An X-Ray on Web-Available XML Schemas

Alberto H. F. Laender, Mirella M. Moro, Cristiano Nascimento and Patrícia Martins[*]
Department of Computer Science
Federal University of Minas Gerais
Belo Horizonte, Brazil
{laender, mirella, crist}@dcc.ufmg.br, patricia.martins@gmail.com

ABSTRACT
XML has conquered its place as the most used standard for representing Web data. An XML schema may be employed for similar purposes of those from database schemas. There are different languages to write an XML schema, such as DTD and XSD. In this paper, we provide a general view, an X-Ray, on Web-available XSD files by identifying which XSD constructs are more and less frequently used. Furthermore, we provide an evolution perspective, showing results from XSD files collected in 2005 and 2008. Hence, we can also draw some conclusions on what trends seem to exist in XSD usage. The results of such study provide relevant information for developers of XML applications, tools and algorithms in which the schema has a distinguished role.

1. INTRODUCTION
A large volume of data is currently represented as XML documents [8]. With such a widespread use, XML has conquered its place as the most employed standard for representing Web data. Web applications frequently need to specify a schema (or a set of schemas) to their documents in order to fulfill their requirements. An XML schema defines a class of documents, i.e., the constraints that all documents must follow in order to be valid. As the complexity of the applications grow, so does the complexity of their schemas.

There are different languages to write an XML schema [12], such as Document Type Definition (DTD) [1] and XML Schema [16]. An XML schema may be employed for similar purposes of those from database schemas. For example, it defines the data structure, extracting information about the data organization, which in turn is essential to store the data in a DBMS. Also, a query optimizer may use facts from the schema for improving its performance. Without such an information, it needs to infer a schema from a dataset, which is a different problem [14]. Finally, other research fronts, such as information integration [5], schema evolution [13], and web services [19], may also benefit from knowing how exactly the schema definitions are used in the real world.

Considering all those schema-oriented applications, some previous work has presented how real DTDs and XML Schema Definitions (XSD) look like. One of the first ones is [10], which provides statistics about the structure of Web-collected DTDs. Another study compared the power expression of real DTDs and XSDs [6]. It concluded that, considering power expression only, most of the XSDs could be written as a similar DTD file. Then, the authors in [3] studied how publicly available XML documents are. They presented statistics on document distribution, schema usage, document internal features, among others.

The goal of the present paper is to provide a general view, an X-Ray, on Web-available XSD files. Specifically, we want to identify which XSD constructs are more and less frequently used. Also, we provide an evolution perspective, showing results from XSD files collected in March 2005 and in November 2008. We can also draw some conclusions on what trends seem to exist in XSD usage. The results of such study provide relevant information for developers of XML applications and tools in which the schema has a defining role.

In summary, this paper presents a snapshot of Web-available XSDs. It works like the initial X-Ray a doctor ask for a patient, trying to grasp a general view of a problem. As the doctor may request an MRI or a CT for enhanced investigation, so can we. For example, we could further examine XSDs by performing an application-based clustering in which we would group XSDs related to bio-data, math-data, and so on. Then, we could evaluate how the frequency of XSD constructs vary from one to another cluster. We leave this deeper study of XSDs for future work, since we are interested in a broader perspective with more general purposes. Finally, as an X-Ray may have different interpretations, so can our study. Hence, we do not exhaustively interpret our results, leaving them open for further discussion as well.

This paper is organized as follows. Section 2 summarizes some concepts about XML and XML Schema. Section 3 presents the methodology employed to perform our evaluations. The results of our experiments are presented and discussed in Section 4. Section 5 overviews some related work while Section 6 concludes the paper.

2. BACKGROUND
XML (*Extensible Markup Language*) is a meta-markup language that provides a format to describe structured and semi-structured data [1]. XML enables more precise content definition and more efficient document search, working over multiple plataforms. Moreover, it also allows the definition

[*]Research partially supported by projects InfoWeb (CNPq grant number 573871/2008-6) and Amanaje (CNPq grant number 479541/2008-6), Brazil.

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<!DOCTYPE SigmodRecord SYSTEM SigmodRecord.dtd">
<SigmodRecord>
 <issue>
  <volume>11</volume>
  <number>1</number>
  <articles>
     <article>
     <title>Annotated Bibliography on Data Design.</title>
     <initPage>45</initPage>
     <endPage>77</endPage>
    <authors>
     <author position="00">Anthony I. Wasserman</author>
     <author position="01">Karen Botnich</author>
    </authors>
  </article>
  <article>
    <title>Architecture of Future Data Base Systems.</title>
    <initPage>30</initPage>
    <endPage>44</endPage>
    <authors>
     <author position="00">Lawrence A. Rowe</author>
     <author position="01">Michael Stonebraker</author>
    </authors>
  </article>
  ...
 </issue>
</SigmodRecord>
```

Figure 1: XML document for SIGMOD Record

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified">
 <xs:element name="SigmodRecord">
  <xs:complexType>
   <xs:sequence>
    <xs:element maxOccurs="unbounded" ref="issue"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="issue">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="volume"/>
    <xs:element ref="number"/>
    <xs:element ref="articles"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 ...
 <xs:element name="author">
  <xs:complexType mixed="true">
   <xs:attribute name="position" use="required"
                                   type="xs:integer"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figure 2: An XSD file for SIGMOD Record

of a new generation of applications to handle and visualize Web data. While its counterpart HTML has a fixed set of tags to define the format of characters and paragraphs, XML provides a system to define an *infinite* number of tags (markups). Specifically, an XML document is composed of element definitions and text values. For example, the XML document for the SIGMOD Record[1] issues is composed of elements that define each issue with volume and number, its articles, and the information of each article (title, pages, and authors). Figure 1 shows a sample of this document.

An XML schema may be defined to guarantee that each document of an application or domain follows the same structural constraints. The most common schema languages are DTD [1] and XML Schema [16]. This paper focuses on analyzing how the actual XSD files are used on the Web. Therefore, this section briefly overviews some XML Schema definition features.

An XSD file is an XML document that allows to define: the elements and attributes that may appear in an XML document, the order and the number of child elements, whether the element is empty or has text content, the data types for elements and attributes, default and pattern values. For example, Figure 2 illustrates a part of an XSD file for the SIGMOD Record document from Figure 1.

Note that this is *one* option for the schema definition of the SIGMOD Record document, a very generic XSD file. We could make it more specific by adding some constraints. For example, SIGMOD Record has four issues per year. We could include such restriction by specifying the element *number* as a *simpleType* with constraints *minInclusive* equal to 1 and *maxInclusive* equal to 4.

Furthermore, elements may be specified as simple types (such as *string* and *positiveInteger*) or complex types. Complex types allow more richer specifications. For example, we can use *group* to define a set of elements. Then, we can add one of three restrictions: (i) *sequence*, those elements must appear in the document in the sequence that they are defined in the XSD; (ii) *choice*, only one of those elements may appear in the document; (iii) *all*, either all elements appear (in any order) or none of them appear in the document.

3. EVALUATION SETUP

This section overviews our evaluation methodology. The process has three steps: it gets data from the Web, validates them, and parses the validated data.

1. Get XSD files from the Web. We consider XML schemas available on the Web for the following reasons. First, those schemas are publicly available, making it easier to reproduce our evaluation. Second, they represent currently in-use XSD files, providing a good snapshot of the real world scenario. Third, those files are not specialized in a limited set of fields (e.g., bio or math data) such as those found in some XSD repositories.

The schemas were selected through a search engine - in our case we have used Google[2]. The query considered on the search engine was "*schema filetype:xsd*". With such a query, we expected to get links only to XSD files that instantiate an element *schema*.

[1]http://www.sigmod.org/record

[2]http://www.google.com, last access in November 2008.

Next task is to get the actual XSD files. We have implemented a crawler using Perl facilities (an HTML parser[3] and a library for WWW in Perl[4]). The crawler gets the URLs that are on the resulting pages of the query "*schema filetype:xsd*" (on the search engine). We have considered the URLs from the first 50 pages, where each page has 10 results. The crawler then downloads the XSD files from those URL links. At the end of this step, we had approximately 500 XSD files, since some of URL links were broken.

2. Validate retrieved XSD files. Once we have a dataset of XSD files, we needed to validate them. We have used a Java Validation API for XML documents[5], and validated schemas against XML Schema's schema[6]. We performed our evaluation in March 2005 and again in November 2008. From the 500 URL links, we got 199 valid XSD files in 2005, and 223 valid XSD files in 2008.

3. Parse validated XSD files. In order to get our statistics, we have also implemented a parser that captures the specific information we are interested in. Specifically, we have defined a script that uses a Perl XML DOM API[7] to parse documents. Its input is a list of XSD files, and its output is the frequency of each XML schema construct. Finally, we have not considered XSD files with *namespace*, *include* or *import* constructs, because we do not have control of them.

4. RESULTS

In this paper, we apply two strategies to evaluate the frequencies in which the XSD constructs appear in the files. The first one, called *general usage* (GEN), counts how many **documents** (from the XSD files dataset) use each construct at least once. This measure tells us which constructs are more/less frequent within the **set** of documents. The second one, called *internal usage* (INT), measures the frequency of each construct over all constructs within each **single** document. In other words, it evaluates how important certain constructs are for each document.

4.1 General Overview

We start our evaluation by presenting a general overview of the constructs considered in Table 1. The first column lists the name of the construct being evaluated. The second and third columns list the general usage on both datasets from 2005 and 2008. For example, the construct *all* appears in 10.55% of all XSD files collected in 2005, and in 19.82% of those collected in 2008. Finally, the last two columns list the internal usage from the same datasets. For example, the construct *all* represents 0.36% of all constructs used within one single document, considering the average of all 2005 files, and in 0.49% of all 2008 files.

Note that GEN represents the frequency in which each construct appears in the overall dataset. On the other hand, INT reflects the internal structure of the XSD files, individually. Therefore, we analyze those results separetely.

[3]http://search.cpan.org/dist/HTML-Parser

[4]http://search.cpan.org/~gaas/libwww-perl-5.800

[5]http://java.sun.com/j2se/1.5.0/docs/api/javax/xml/validation/package-summary.html

[6]http://www.w3.org/2001/XMLSchema.xsd

[7]http://search.cpan.org/dist/XML-DOM

Table 1: Usage of XML Schema elements (%)

Construct	GEN-05	GEN-08	INT-05	INT-08
all	10.55	19.82	0.36	0.49
annotation	59.80	81.98	8.79	10.37
any	15.07	31.53	0.08	0.33
anyAttribute	6.53	10.81	0.03	0.19
appinfo	5.53	9.91	0.08	0.09
attribute	79.40	81.08	7.89	9.34
attributeGroup	8.54	11.71	1.01	0.60
choice	46.23	42.34	0.94	0.84
complexContent	19.09	30.63	0.71	1.38
complexType	91.46	92.79	5.77	8.83
documentation	58.80	81.98	9.40	10.59
element	92.96	93.69	25.66	26.57
enumeration	56.78	72.97	19.35	8.68
extension	47.24	53.15	1.46	2.04
field	10.05	16.22	1.40	0.45
fractionDigits	2.51	3.60	0.16	0.04
group	8.04	12.61	0.47	0.55
key	7.54	9.91	0.29	0.12
keyref	5.53	7.21	0.27	0.13
length	4.52	8.11	0.02	0.04
list	5.53	7.21	0.04	0.06
maxExclusive	2.51	4.50	0.04	0.01
maxInclusive	15.07	28.83	0.40	0.35
maxLength	11.56	20.72	0.52	0.42
minExclusive	2.51	6.31	0.27	0.03
minInclusive	17.59	32.43	0.41	0.35
minLength	11.56	20.72	0.14	0.41
notation	0.50	0.00	0.00	0.00
pattern	24.62	36.04	0.93	1.14
restriction	68.34	82.88	3.31	3.44
selector	10.05	16.22	0.89	0.42
sequence	87.94	89.19	3.50	5.79
simpleContent	36.18	37.84	0.76	0.72
simpleType	68.84	82.88	3.66	3.63
totalDigits	2.51	3.60	0.17	0.05
union	5.53	15.32	0.31	0.19
unique	5.53	13.51	0.32	0.17
whiteSpace	4.02	4.50	0.07	0.09

General Usage

We now discuss some of the most interesting results presented in the first three columns of Table 1. The constructs that are most used are: *complexType*, *element*, *sequence*, *simpleType*, and *restriction*. The least used ones are *notation*, *totalDigits*, *fractionDigits*, *maxExclusive*, and *whiteSpace*. There is an interesting difference between those two sets: the former is more related to the document structure, while the latter to the values inside the elements.

It is important to notice that there is a general increase in the use of all constructs over the last three years. The exceptions are the constructs *choice* and *notation*. Such an increase appears in the most frequently used constructs and in the least ones as well. Also, the most frequent ones (i.e., *element*, *complexType*, and *attribute*) have increased less in relation to the others. This can indicate the stability on the use of such constructs.

Considering the constructs that had a high increase from 2005 to 2008 (such as *documentation*, *annotation*, *any*, *enu-*

Table 2: Content model distribution in complexType construct

Content Model	Internal Usage	
	2005 (%)	2008 (%)
Simple	13.11	13.45
Complex	86.89	86.55

meration, *minInclusive*, and *restriction*), one possible explanation would be that the users are more familiar with them now. Also, as more tools for handling XML schemas become available, the users start to acquire confidence in using more complex structures as well. Moreover, as XML applications become more specific, so do their requirements and data restrictions. For example, instead of using an element of type *xs:int*, it may be necessary to employ a *simpleType* with restrictions in its minimum and maximum values. Particularly expressive is the use of the construct *documentation*, which occurred in 53.80% of the 2005 XSD files and in 81.92% of the 2008 ones. This is a clear indication that XSDs are, in general, very complex and need to be properly documented.

Another explanation for the general increasing trend would be the cascade effect. In other words, the increase of using one construct may have affected the increasing of others. Such a consequence is expected because, most of the times, the constructs allow nesting. Therefore, if the use of an external construct increases, the same will happen to its internal constructs. For example, the constructs *maxExclusive*, *maxInclusive*, *minExclusive*, *minExclusive*, *restriction*, *pattern*, *enumeration*, *totalDigits*, and *fractionDigits* are usually associated to the construct *simpleType*. Hence, *simpleType* contains the other constructs, such that increasing the usage of *simpleType* will also increase the usage of those constructs. According to Table 1, the usage of *simpleType* has grown around 20% and its associated constructs have grown even more. Despite the general usage inscrease of most constructs, surprinsingly, the use of *key* and *keyref* still remains low (less than 10%). This might be an indication that, in most XML applications, user-defined keys are still the usual design choice.

Finally, the construct *notation* specifies the format of non-XML data. It was the least used one in 2005 and it does not appear in any of the documents in 2008. As already pointed out by van der Vlist [17], notations were very rare in real world applications. Now, we have just shown, empirically, that notations are not used at all.

Internal Usage

Here, we discuss some of the most interesting results presented on the last two columns of Table 1. Half of the constructs had their presence within XSD files decreased (while the other half increased). One of the possible reasons is, again, the users becoming more familiar with different constructs. One possible scenario is more people using different constructs with low internal usage, which is common in a learning phase. Since we calculate the average, the low usage in new files may have caused the overall decreasing. Another explanation, still related to learning, is that some constructs were replaced by others to better attend the applications requirements. For example, the lower usage of *enumeration* (which defines a list of possible values) may have been influenced by the higher usage of *pattern* (which defines regular expressions). Note that defining a list of possible values seems to be easier than defining a pattern. However, the latter is more powerful and requires more time to be mastered, while the former can be quickly learned.

Table 3: Usage of constructs all, choice, sequence and group in complex content model

Construct	Internal Usage	
	2005(%)	2008(%)
All	8.61	6.77
Choice	11.52	14.43
Sequence	79.35	75.77
Group	0.51	3.03

Other Considerations

The results for general and internal usages may change when we analyze schemas of particular areas. For example, we can expect more use of *fractionDigits* in commercial and scientific applications. Hence, our explanations for the results must not be taken as true for any type of file collection.

Moreover, general and internal usage represent different aspects of a schema collection and do not have any direct relation. They also follow the 20/80 rule. Specifically, on the general usage columns, we can see that only 20% of the constructs appear in more than 80% of the XSD files. Likewise, on the internal usage columns, we find that 20% of the constructs represent 80% of all usage. It shows that usage of XML Schema constructs has been highly concentrated.

4.2 Complex Constructs

As we can see from Table 1, *complextType* is the second most used construct. It allows to create more sophisticated structures, to expand basic types, to provide more flexibility, and so on. Due to its importance and intricate structure, we provide a detailed analysis of its components. A closer view of other construct is left for future work.

Complex types are formed by either the simple or the complex content model. Simple content defines element attributes, whereas complex content describes the markup structure. Table 2 presents the distribution of simple and complex contents (internal usage). This table shows that the distribution has almost not changed along the last three years. This table also shows that a major part (more than 86%) of the *complexType* models in the schema contain complex contents. Therefore, we detail how those complex contents are actually employed next.

Group, All, Choice, and Sequence

In order to create complex content, the following constructs are available: *all*, *choice*, *sequence*, and *group*. Table 3 presents the usage of those constructs. The results on that table show a small variation from 2005 to 2008. The con-

Table 4: Usage of compositors with only one child-element

Construct	Internal Usage		General Usage	
	2005 (%)	2008 (%)	2005 (%)	2008 (%)
All	14.05	12.02	4.02	3.59
Choice	17.94	23.26	11.05	13.45
Sequence	42.75	48.08	77.39	77.58

Table 5: Nesting in composition constructors

Construct	Nesting	
	2005(%)	2008 (%)
Choice	8.86	5.46
Sequence	9.62	7.36

struct *sequence* (ordered elements) is still the most used, followed by *choice*, *all*, and *group*. The increase in using *group* may explain the decrease in using *all* and *sequence*. It is important to notice that *group* is defined outside *complexContent*, while the other three constructs can be embedded within it, which then may justify those numbers. We can also suppose that *all* and *sequence* were more used in *group* than in *choice*, since Table 3 shows that the usage of all those constructs has increased, but *choice*. Considering only Table 3, we could infer wrongly that *sequence* and *all* were being replaced by *choice*.

Table 4 presents the usage of compositors (*all*, *choice*, and *sequence*) with only *one* child-element. Those constructs define lists of elements. Hence, we could expect that a list would have more than one element. However, in our dataset, the reality is a little bit different. In 2005, it was common to include an only child-element in those compositors. In 2008, we confirmed such an interesting tendency. In the *sequence* construct we can observe that the usage of an only child-element is highly common (48% of internal usage). Perhaps the proportion observed in here can be a consequence of the usage found in Table 3. In many cases, we can use any of those constructs to build the same structure with one child-element. However, as we usually apply *sequence*, we tend to use it even when it is not the only option.

Nesting

Another interesting information is the presence of nested constructions within complex types. The usage of nesting shows how complex the schema structure is. Table 5 presents usage of nesting among compositors, except *all* which cannot be used as a particle [17]. According to Table 5, nesting is not really used in practice and its usage has decreased. Table 6 shows the usage of nesting in XSD files (at least one compositor using nesting). We can observe that most of the files do not use nesting among compositors as well.

We believe that the reasons for results from Table 5 and 6 are twofold. First, real world applications do not demand complex nestings. Second, users are not prepared enough to make an elaborate use of those kinds of constructs.

Table 6: General Usage of nesting in composition constructors

Type	General Usage	
	2005 (%)	2008 (%)
Nesting	32.66	33.18
No nesting	67.34	66.82

Table 7: General Usage of extension and restriction in simple and complex types

Derivation	Simple Type		Complex Type	
	2004 (%)	2008 (%)	2004(%)	2008 (%)
extension	27.00	37.84	37.00	34.53
restriction	73.00	82.88	7.00	7.62

Derivation: Extension and Restriction

In 2004, a study presented in [6] compared the features from DTDs and XSDs, with focus on expressive power. One important feature (from the point-of-view of language power) is the derivation of new types. Both simple and complex types may be derived by *extension* and *restriction*. In summary, there are four possibilities: (*i*) we can derive a complex type from a simple type by extension, and then add attributes to elements; (*ii*) we can extend a complex type and then add sequence of elements to its content model or add attributes; (*iii*) we can restrict a simple type and limit its acceptable range of values; and (iv) we can restrict a complex type and limit its acceptable range of subtrees.

That study considered 93 XSDs (collected in 2004) and it counted within how many files those four types of derivation were defined. We measured those as well in 2008, and Table 7 presents both results (from their study in 2004 and from ours in 2008). Note that there is an increase in the derivation of simple types, while there is practically no change in the derivation of complex types.

5. RELATED WORK

In this section, we present some related work divided in two parts. First, we overview some research papers on XML data management that depend on the information stored on the schemas. Second, we discuss other publications that evaluated actual schemas in use.

5.1 XML Data Management and Schemas

The management of XML data by a native or an XML-enabled DBMS has been widely discussed. Two central questions are how to store and how to query the data (for example [2, 4, 7, 9, 11, 15, 18], just to cite a few). We can divide those approaches into two categories: (*i*) those that do not depend on schema definition, such as [9] and [11]; and (*ii*) those that depend on schema definitions. The second category can be further divided in those that employ DTDs [4, 15] and XSDs [2, 7, 18].

Our empirical evaluation can be of major value to research and industry work similar to those aforementioned, as well as to those on information integration [5], schema evolution [13], and web services [19]. We offered a snapshot, an X-Ray, on what features are currently most and least used in XSDs. Therefore, one can decide to invest more time on optimizing the most frequently used constructs, leaving the least used ones for a second moment. Furthermore, we also show a trend in using more sophisticated constructs. Such a trend can have a positive impact on design decisions of new applications as well.

5.2 Actual Schemas in Use

An early study discussed how real DTDs were like [10]. Such a study evaluated files from a DTD repository and presented different statistics considering local and global properties, such as syntactic complexity, ambiguity, determinism, reachability, recursions, path sizes, among others. It is very similar to our study in spirit, since it wanted to provide an overview of currently in-use DTDs. However, ours deals with a more complex, powerful schema definition language.

A later study compared the features from DTDs and XSDs [6]. It considered DTDs and XSDs files from the Web and focused on finding out which features from XML Schema, that are not allowed in DTDs, are more used in practice and how sophisticated the features employed (in both languages) are in practice. The main conclusion is that the expressive power from real world XSDs are mostly equivalent to that of DTDs. However, note that its focus is on expressive power, while ours is on the actual schema structure. Hence, our study is complimentary to that one.

Also related to our work is [3], which presents a study over features of XML *documents* (schema instances) from the Web. It presented statistics on document distribution, schema usage, document internal features, among others. Specifically, the internal features are broken down on node distribution, size, depth, element and attribute fan-out, and recursion. That study focuses on the actual XML documents, while our focuses on the schema definition using XSD. Nevertheless, our research work complements all those aforementioned by providing an X-Ray on XSDs files.

6. CONCLUDING REMARKS

In this paper, we provided an X-Ray on the structural features of XSD files available on the Web. We considered files from the Web mainly because they are not specialized in a limited set of areas (e.g., bio or math data) such as those files found in some XSD repositories. Our evaluation showed the most and the least frequently used XSD constructs. We also presented an evolution on the usage of those constructs, by considering XSD files available on the Web in March 2005 and in November 2008. From such an evolution, we would like to emphasize three findings: (*i*) there is a general increase in the use of all constructs; (*ii*) the construct *notation* has disappeared from XSDs files; and (*iii*) the 20/80 rule, where 20% of the constructs represent more than 80% of all internal usage, and 20% of the constructs appear in more than 80% of all XSD files. Finally, we did not exhaustively interpret our results, leaving them open for further discussion as well. Nevertheless, our empirical evaluation provides relevant information for developers of XML applications, tools, and algorithms in which the schema has a distinguished role.

7. REFERENCES

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.

[2] S. Amer-Yahia, F. Du, and J. Freire. A Comprehensive Solution to the XML-to-Relational Mapping Problem. In *Procs. of WIDM*, pages 31–38, 2004.

[3] D. Barbosa, L. Mignet, and P. Veltri. Studying the XML Web: Gathering Statistics from an XML Sample. *World Wide Web Journal*, 8(4):413–438, 2005.

[4] M. Benedikt, W. Fan, and F. Geerts. XPath Satisfiability in the Presence of DTDs. In *Procs. of PODS*, pages 25–36, 2005.

[5] P. A. Bernstein and L. M. Haas. Information Integration in the Enterprise. *Commun. ACM*, 51(9):72–79, 2008.

[6] G. J. Bex, F. Neven, and J. V. den Bussche. DTDs versus XML Schema: A Practical Study. In *Procs. of WebDB*, pages 79–84, 2004.

[7] P. Bohannon et al. From XML Schema to Relations: A Cost-Based Approach to XML Storage. In *Procs. of ICDE*, pages 64–75, 2002.

[8] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. M. F. Yergeau. *Extensible Markup Language (XML) 1.0*. W3C, 5th edition, November 2008. http://www.w3.org/TR/REC-xml/.

[9] D. Che, K. Aberer, and T. Özsu. Query Optimization in XML Structured-Document Databases. *The VLDB Journal*, 15(3):263–289, 2006.

[10] B. Choi. What are real DTDs like? In *Procs. of WebDB*, pages 43–48, 2002.

[11] D. Florescu and D. Kossmann. Storing and Querying XML Data using an RDMBS. *IEEE Data Eng. Bull.*, 22(3):27–34, 1999.

[12] D. Lee and W. W. Chu. Comparative Analysis of Six XML Schema Languages. *SIGMOD Record*, 29(3):76–87, 2000.

[13] M. M. Moro, S. Malaika, and L. Lim. Preserving XML Queries during Schema Evolution. In *Procs. of WWW*, pages 1341–1342, 2007.

[14] S. Nestorov, S. Abiteboul, and R. Motwani. Infering Structure in Semistructured Data. *SIGMOD Record*, 26(4):39–43, 1997.

[15] J. Shanmugasundaram et al. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Procs. of VLDB*, pages 302–314, 1999.

[16] H. S. Thompson et al. *XML Schema Part 1: Structures*. W3C, 2nd edition, October 2004. http://www.w3.org/TR/xmlschema-1/.

[17] E. van der Vlist. *XML Schema*. O'Reilly and Associates, 1st edition, June 2002.

[18] I. Varlamis and M. Vazirgiannis. Bridging XML-schema and relational databases. A system for generating and manipulating relational databases using valid XML documents. In *Procs. of ACM DocEng*, pages 105–114, 2001.

[19] Q. Yu et al. Deploying and Managing Web Services: Issues, Solutions, and Directions. *The VLDB Journal*, 17(3):537–572, 2008.