What Are *Real* DTDs Like

Byron Choi
University of Pennsylvania
kkchoi@gradient.cis.upenn.edu

Abstract

DTDs have proved important in a variety of areas: transformations between XML and databases, XML storage, XML publishing, consistency analysis of XML specifications, typechecking, and optimization of XML queries. Much of this work depends on certain assumptions about DTDs, e.g., the absence of recursion and nondeterminism. With this comes the need to justify these assumptions against DTDs in the real world. This paper surveys a number of DTDs collected from the Web, and provides statistics with respect to a variety of criteria commonly discussed in XML research.

1 Introduction

The first and most widely used method of describing the structure of an XML document is the Document Type Definition (DTD). It is part of the XML standard [3], and more sophisticated tools for structuring XML, such as XML-Schema [21, 1] are based on DTDs. Recently DTDs have been found useful in implementing efficient storage systems for XML [19] and in typechecking programming and query languages for XML [12, 15, 7]. Software such as XML storage systems may only work well when the structure of the DTD has certain properties. For this reason, as well as to see whether DTDs "make sense", we collected a number of DTDs and analyzed their structure. An earlier survey [18] on DTDs discusses some of their limitations but does not study the properties that we analyze in this paper.

Our statistics were collected by the "DTD Inquisitor" [5], a program that reads a DTD, attempts to identify problems with it, and computes a number of graph-theoretic properties of it. An online demo can be found at [5]. The statistics it collects fall into two categories: *local* – describing the kinds of content models found at individual element declarations, and *global* – describing the graph-theoretic structure of the DTDs and the documents that conform to them. For example: saying that there are no elements with *mixed* content is a local property, while saying that the maximum path length allowed by this DTD is 4 is a global property. Clearly these two properties are not entirely independent: a DTD that contains complex content models (a local property) is likely to allow a large number of distinct paths of a given depth (a global property).

1.1 The DTD Sample

The DTDs were recently extracted from the XML.org DTD repository [23]. We attempted to clean up all the DTDs in initial sample by hand; however some were too full of errors to allow us to do this. We were left with a total of 60 DTDs that formed the basis of our survey.

One may expect that the structure of a DTD depends on how it is intended to be used. For this reason the DTDs are divided into three broad categories: *app*, *data* and *meta*. DTDs that are primarily designed for data interchange between applications are called *app*, e.g. bookmarks exchange [17]. DTDs for *data* are those one might imagine easily putting in a structured database – baseball statistics [11], for example. Finally DTDs whose function is to describe the structure of document markup, such as that for Shakespeare's plays [2], are termed *meta*. The boundaries between these three kinds of DTD are not clear. For example a BIOML DTD [20] which describes an annotation framework for biopolymer sequences might be put in more than one categories. They certainly describe data; they also are designed as a common syntax for exchanging the data between scientists. However, for the most part there was little doubt about the classification. The element and attribute names usually tell how DTDs are used. DTDs with names *bold*, *italic* or *paragraph* are likely belong to the *meta* category. One can also tell from the structure of the DTDs – mixed content for example. Of our 60 DTDs, 7 were *app*, 13 were *data* and 40 were *meta*. A complete listing of the DTD sample can be found in [6].

1.2 The Statistics

The statistic are grouped into (1) local and (2) global properties.

- *Local properties.* In addition to noting the presence of mixed or *any* content, we extracted a number of

Class	*app*	*data*	*meta*
pcdata	13% (32)	58% (1147)	26% (685)
ϵ	13% (32)	2% (33)	16% (415)
any	0% (0)	0% (4)	1% (26)
Mixed content	4% (9)	1% (21)	19% (488)
"|" only (not mixed)	22% (55)	2% (32)	11% (285)
"," only	14% (34)	28% (557)	11% 296
Complex content	19% (48)	7% (132)	7% (188)
List	13% (33)	2% (31)	6% (161)
Single	3% (7)	1% (29)	2% (63)

Table 1: The percentage and the number of the content models in the nine classes.

statistics concerning the structure and complexity of the content models involved in the DTDs. We also looked for properties that would affect the parsing of the DTD or its ambiguity in being used as a type [15, 13]

- *Global properties.* One can come up with a never-ending sequence of graph-theoretic properties to analyze. We chose some that might be important in the mapping of the XML into some database format. For example, if the presence of a Kleene star indicates the need for a table or some other collection type [19], one is interested in how deeply nested these can be. One is also interested in what kind of recursion can occur in DTDs. If one is interested in finding structure for optimization or type-checking as in Dataguides [10] one would like to know about the number and length of paths that are allowed by the DTD.

We next define some notation to be used throughout the paper. A DTD [3] can be described as a collection of element declarations of the form $e \rightarrow \alpha$ where e is the element name (type) and α is the content model. The content model is defined by: $\alpha ::= \epsilon \mid pcdata \mid e \mid \alpha,\alpha \mid \alpha|\alpha \mid \alpha* \mid \alpha+ \mid \alpha?$, where ϵ denotes the empty content model, *pcdata* denotes string, e denotes an element name, "," and "|" stand for concatenation and union, and "*", "+" and "?" stand for zero or more, one or more and optional occurrences.

In the following sections we present the results of our analysis. Most of the figures in this paper are as follows unless otherwise specified. DTDs are grouped by their categories. Two vertical lines partition the visualization into three areas. The leftmost, middle and rightmost areas show the result from *app* DTDs, *data* DTDs and *meta* DTDs respectively. Each bar represents a DTD. The bars are sorted within each area for presentation purpose.

2 Local Properties

2.1 Content Model Classification

We classified the content model of the three categories. The classification of the content models is follow: (1) *pcdata*, (2) ϵ, (3) *any*, (4) mixed content, (5) "|" only but not mixed content, (6) "," only content, (7) complex content, (8) list and (9) single. The first two classes are string and empty content. The class *any* denotes no restriction on the subelements. Mixed contents are mixture of text and elements, For example, $p \rightarrow (bold|italic|pcdata)*$ means a paragraph contains a mixture of normal, bold and italic text. The fifth class is union of elements, e.g. $dna\_type \rightarrow (A|T|G|C)$. The sixth class, "," only content models, are tuple-like structure, e.g., $address \rightarrow addrline?, city?, country?$. Complex content is a content model with both ","s and "|"s. List is the content model with one element name followed by a Kleene star or a plus. Single is the content model with one element name followed by an optional "?".

The breakdown is shown in table 1. The structure of the DTDs is closely related to their category. DTDs in the *app* category have diverse structures. However, we see a large number of data string and the tuple-like structures in *data* DTD. The *meta* DTDs are dominated by the data string and the mixed content.

Note that the presence of *any* content makes it impossible to measure some graph-theoretic properties in DTD. Thus we ignored such content in our analysis.

2.2 Syntactic Complexity

The syntax for XML DTD content model allows one to write arbitrarily complex expressions. We investigate the complexity of content model in this section. A *depth* function is defined as a rough measure of the content model complexity.

Definition 1. *The inductive definition of the* depth *of the content model:*
$depth(\epsilon) = 0$; $depth(e) = 1$;

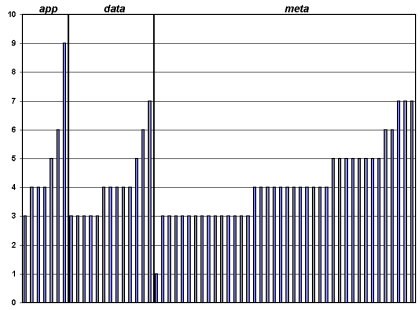

Figure 1: Maximum depth of the content model in DTDs

$depth(\alpha*) = depth(\alpha+) = depth(\alpha?) = depth(\alpha) + 1;$
$depth(pcdata) = 1;$
$depth(\alpha_1, \alpha_2, ..., \alpha_n) = depth(\alpha_1|\alpha_2, ...|\alpha_n) =$
$max(depth(\alpha_i)) + 1, where\ 1 \leq i \leq n.$

Figure 1 shows that the mode of the maximum depth of the regular expression is 3 and the most complex expression has the depth value 9. This indicates that content models are usually not complex. The most complex content model (with the depth 9) is found in the W3C's Scalable Vector Graphics DTD [8], shown below. We expanded its entities and show the element declaration below. The content model encodes the semantic of allowing at most one of *desc*, *title*, and *metadata* element in *any order*. Specifically, the portion of the content model in line one has a depth value 6. We underline a *metadata* element name and trace on how the depth value increases. The meaning of a complex content model can easily become non-trivial.

$$path \rightarrow ((((desc, ((title, \underline{metadata}?)|(metadata, title?))?)|$$
$$(title, ((desc, metadata?)|(metadata, desc?))?)|$$
$$(metadata, ((desc, title?)|(title, desc?))?))?),$$
$$(animate|set|animateMotion|animateColor|$$
$$animateTransform))$$

2.3 Determinism

The XML Standard [3] defines *deterministic* content models to be those do not require look ahead when parsing. Non-deterministic content model is not allowed in XML DTDs [3]. Certain typed XML query language implementation [7] assumes determinism. An example of non-deterministic content model is *(a, b | a, c)* and that of deterministic content model is *a, (b | c)*. The Inquisitor detects 5 non-deterministic content models and they are found in 4 DTDs. For example, the following non-deterministic content model is found in the DTD from workflow management coalition [22], $WF\_XML \rightarrow (request|(request, response))$.

2.4 Ambiguity

We follow the definition of ambiguity in [4]. An expression R is ambiguous if and only if there exists some string s in $L(R)$ such that there can be distinct ways to parse string s. It is straightforward to show that any ambiguous content model is non-deterministic.

Very often, a data source exports its data in XML format and the receiver maps the XML into some objects [17]. Ambiguity in DTDs means that the mapping is not unique. An ambiguous content model from a DTD for open application [16] is shown below:

$$partner \rightarrow (\underline{name}?, onetime?, partnrid?, partnrtype?,$$
$$syncind?, active?, currency?, descriptn?,$$
$$dunsnumber?, glentitys?, \underline{name}*, parentid?,$$
$$partnridx?, partnrratg?, partnrrole?,$$
$$paymethod?, taxexempt?, taxid?, termid?,$$
$$userarea?, address*, contact*)$$

Consider the underlined element names. When the XML document is streamed over a network, the *partner* element arrived followed by a *name* element. The receiver cannot decide if it is the first optional *name* or the sequence of *name*s. Mapping it to either *name* can form a parse. The Inquisitor detected 2 ambiguous content models.

3 Global Properties

3.1 Reachability

The definition of reachable element name is given below.
Definition 2. *An element name e' is* reachable *from e, denoted by $e \Rightarrow e'$, if either $e \rightarrow \alpha$ and e' occurs in α, or $e \Rightarrow e''$ and $e'' \Rightarrow e'$.*
Similarly, a content model α is derivable *from an element name e, denoted by $e \Rightarrow \alpha$, if either $e \rightarrow \alpha$, or $e \Rightarrow \alpha'$, $e' \rightarrow \alpha''$, and $\alpha = \alpha'[e'/\alpha'']$, where $\alpha'[e'/\alpha'']$ denotes the content model obtained by substituting α'' for all occurrences of e' in α'.*
Definition 3. *An element name e is* reachable *if $r \Rightarrow e$, where r is the name of the root element. Otherwise element name e is called unreachable or useless.*

We assume that the root of the sample DTD is not known, i.e. all element names can be the root element name. Figure 2 shows the number of unreachable element names in DTDs. These unreachable element names are either the root element name or useless. The mode of the number of unreachable element name is 1. Thus DTDs very often have a clear notion of the root element. An *app* DTD with 20 unreachable element name encodes an application, and different kinds of small request and response messages in XML syntax. Each of these small messages are not reachable from all other element names.

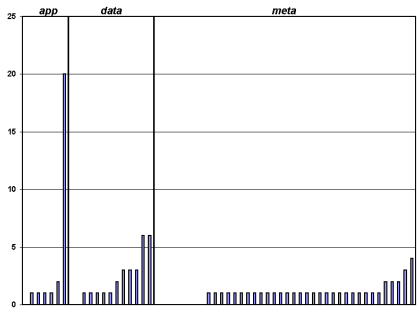

Figure 2: Unreachable element names in DTDs

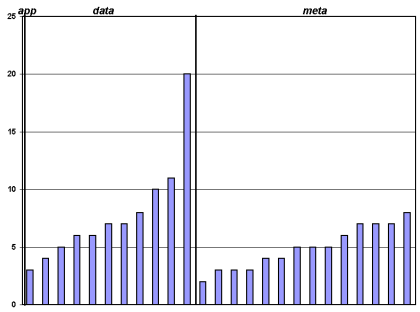

Figure 3: Longest simple path in non-recursive DTDs

Separating the unreachable parts in DTD into smaller DTDs appear to be a better design.

3.2 Recursions

We define recursive DTDs and two kinds of recursions as follow.

Definition 4. *A DTD is* recursive *if and only if it has an element name e such that* $e \Rightarrow e$ *and e is reachable.*

Definition 5. *A DTD is* linear recursive *if and only if it is recursive and for any reachable element name e and any* $e \Rightarrow \alpha$, *e occurs at most once in* α *and the occurrence is not enclosed in "*" or "+". A DTD is said to be non-linear recursive if it is recursive but is not linear recursive.*

Consider an example of a non-linear recursive element DTD [17]. *folder* → *title?, info?, desc?, (bookmark|folder|alias|separator)**. A folder has some optional information and, among other things, a list of folders. *folder* does not occur in *bookmark*, *alias* and *separator*. However, *folder* is enclosed in a "*".

No linear recursive DTD is found in our sample. The sample contains 7, 2 and 26 non-linear recursive DTDs in the *app*, *data* and *meta* category respectively. The analysis shows that there are a few (25) non-recursive DTDs in our sample. *data* DTDs are usually non-recursive.

3.3 Simple Path and Simple Cycle

For the non-recursive DTDs, we investigate their longest simple path. The result is shown in figure 3. The leftmost area is empty since all DTDs in the first category are recursive. The result shows that the longest path in such DTDs is not too long (mostly less than 8). The DTD for exchanging financial data [14] allows the longest simple path with the length 20. The structure in the DTD is dominated by tuples and strings. We do not have an explanation on permitting such a long path in this DTD.

We also investigate the number of simple paths in non-recursive DTDs and the number of simple cycles in recursive DTDs. Simple cycle is a path in the form e_1, e_2, ... e_k, e_1, where $e_1, e_2, \ldots e_k$ are distinct element names. The range of the number of simple paths in the DTD sample is wide. For the number is larger than 512, we treat it as "too many" and do not show the entire bar. Figure 4 shows the number of simple path allowed by non-recursive DTDs. DTDs appear either allowing few or a large number of simple paths.

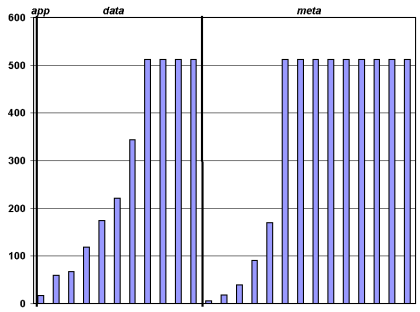

Figure 4: The number of simple path in non-recursive DTDs

The number of simple cycle allowed by recursive DTDs is shown in figure 5. For the number is larger than 50, we do not show the entire bar but the actual value is shown on top of it. Most DTDs with large number of simple cycles are designed for document markup. The short bars in *meta* category are the DTDs for marking up small messages. Recursive *data* and *app* DTDs have small number of simple cycles, with some exceptions.

3.4 Chain of Stars

A real example [9] of a chain of 2 stars is follow: *entity* → *name*, contact*, location*, phone*, fax*, email** and *location* → *address*, city?, region?, postal?, country?*. A business entity has a list of locations. Each location has some address lines. A star/plus in a cycle leads to a chain of stars with infinite length. Such chains are not included in this analysis. The result of this analysis is shown in figure 6. The longest chain of stars is often

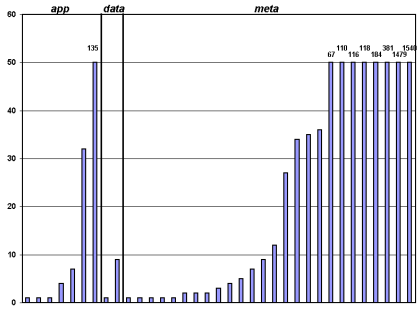

Figure 5: The number of simple cycles in recursive DTDs

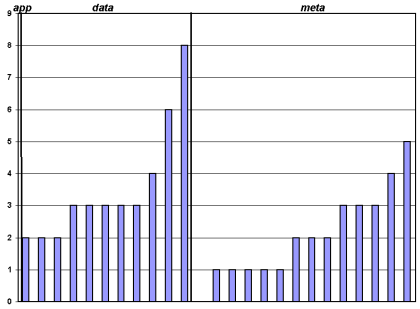

Figure 6: Longest chain of stars in non-recursive DTDs

small (the mode is 3). One star-free non-recursive DTD is detected. The extreme case is found in the DTD for exchanging financial data mentioned in section 3.3.

3.5 Hubs

Fan-in of an element name, e, is the cardinality of the set $\{e'|e' \rightarrow \alpha'$ and e occurs in $\alpha'\}$. An element name with a large fan-in value is called *hub*. We present the fan-in of the elements in *data* and *meta* in figure 7 and figure 8 respectively. We skip the result for the DTDs for *app* DTDs for space constraint. Each line in the figure represents a DTD. A cardinal number (element name ID) is assigned to each element name. The fan-in values are sorted in descending order (all lines are monotonically decreasing). For DTD with few element names, the line is short. The figure shows the largest 52 fan-in values. Our survey shows that hubs exist in DTDs in all categories. In some cases, documents contain many elements whose name is a hub. Fast access to hubs can be in high priority.

An observation is that many content models are exactly the same, a prefix or a suffix of another. These content models cause some horizontal lines at high fan-in values.

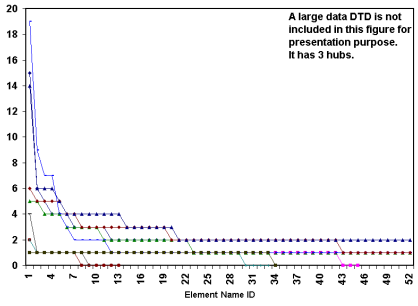

Figure 7: Fan-in of content models in *data* DTDs

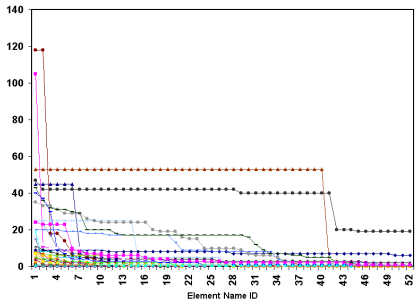

Figure 8: Fan-in of content models in *meta* DTDs

4 Conclusion

Some XML storage system, typechecking and query language depend on certain assumptions on DTDs. We collected real DTDs and analyzed the structures that may be assumed in XML research. The paper provides statistics on some structures of real DTDs.

Acknowledgement. Thanks to Peter Buneman and Wenfei Fan for their invaluable comment and encouragement. The author would also like to thank many members of Penn Database Research Group for suggestions on this paper.

References

[1] P. V. Biron and A. Malhotra. XML Schema Part 2: Datatypes. W3C Recommendation. Available at `http://www.w3.org/TR/xmlschema-2`, May 2001.

[2] J. Bosak. The Plays of Shakespeare in XML. Available at `http://www.oasis-open.org/cover/bosakShakespeare200.html`, July 1999.

[3] T. Bray, J. Paoli, and C. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. W3C Recommendation. Available from `http://www.w3.org/TR/1998/REC-xml-19980210`, Feb. 1998.

[4] A. Brueggemann-Klein and D. Wood. The Validation of SGML Content Models, 1994.

[5] B. Choi. DTD Inquisitor 2. Available at `http://db.cis.upenn.edu/~kkchoi/DTDI2/`, Sept. 2001.

[6] B. Choi. What are Real DTDs Like. Technical Report, 2002.

[7] B. Choi, M. Fernandez, J. Simeon, and P. Walder. Galax Demo. Available at `http://db.bell-labs.com/galax/`, 2002.

[8] Chris Lilley and Dean Jackson. Scalable Vector Graphics (SVG). Available at `http://www.w3.org/Graphics/SVG/`, Feb. 2002.

[9] Financial Services Technology Consortium. The Bank Internet Payment System (BIPS): Leading the Way to Electronic Commerce. Available at `http://www.fstc.org/projects/bips/`.

[10] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *The 23rd International Conference on Very Large Data Bases*, pages 436–445, 1997.

[11] E. R. Harold. Baseball Source Code and Examples from The XML Bible. Available at `http://www.ibiblio.org/xml/examples/baseball/`, Apr. 1999.

[12] H. Hosoya and B. Pierce. XDuce: A Typed XML Processing Language. In *The 3rd International Workshop on the Web and Databases*, pages 226–244, May 2000.

[13] H. Hosoya, J. Vouillon, and B. Pierce. Regular expression types for XML. In *The International Conference on Functional Programming (ICFP)*, pages 11–22, Sept. 2000.

[14] IFX Forum, Inc. IFX Home. Available at `http://www.ifxforum.org`, 2002.

[15] M. Makoto. RELAX (REgular LAnguage description for XML). Available at `http://www.xml.gr.jp/relax/`, July 2000.

[16] Open Application Group. Open Application Group. Available at `http://www.openapplications.org/`.

[17] Python XML Special Interest Group. The XML Bookmark Exchange Language Resource Page. Available at `http://pyxml.sourceforge.net/topics/xbel/`.

[18] A. Sahuguet. Everything You Ever Wanted to Know About DTDs, But Were Afraid to Ask. In *WebDB-2000*, 2000.

[19] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *The 25th International Conference on Very Large Data Bases*, pages 302–314, 1999.

[20] D. States and P. Gordon. The BIOML home page. Available at `http://www.bioml.com/BIOML/bioml.dtd`, Mar. 1999.

[21] H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures. W3C Recommendation. Available at `http://www.w3.org/TR/xmlschema-1`, May 2001.

[22] Workflow Management Coalition. Workflow Management Coalition, WF-XML specification. Available at `http://www.oasis-open.org/cover/WFXML10a-Alpha.html`, Apr. 1999.

[23] XML.org. XML.org Registry. Available at `http://www.xml.org/xml/registry.jsp`, 2002.