Everything You Ever Wanted to Know About DTDs, But Were Afraid to Ask

(extended abstract)

Arnaud Sahuguet
sahuguet@saul.cis.upenn.edu

University of Pennsylvania

1 Introduction

For the last two years, XML has become an increasingly popular data-format embraced by a lot of different communities. XML is extremely attractive because it offers a simple, intuitive and uniform text-based syntax and is extensible. One can find today XML proposals for messages, text content delivery and presentation, data content, documents, software components, scientific data, real-estate ads, financial products, cooking recipes, etc.

Unfortunately this also means that XML is far too general and if people plan to use it in serious applications (mainly for Electronic Document Interchange, in a broad sense), they will need to provide a specification (i.e. structure, constraints, etc.) for *their* XML, which XML itself cannot offer.

In order to specify and enforce this structure, people have been using Document Type Definitions (DTDs), inherited from SGML.

In this paper, we present some preliminary results that explore how DTDs are being used for specifying the structure of XML documents. By looking at some publicly available DTDs, we look at how people are actually (mis)using DTDs, show some shortcomings, list some requirements and discuss possible replacements.

But before getting further, let us ask the following legitimate question: *why bother?*. And to answer it, let us review what DTDs should be good for.

Historically, DTDs have been invented for SGML. The purpose of a DTD is to permit "*to determine whether the mark-up for an individual document is correct and also to supply the mark-up that is missing because it can be inferred unambiguously from other mark-up present*" [ISO 8879].

The two historical functions of DTDs have been parsing and validation. The parsing function is less relevant with XML (since XML does not permit to omit tags). The validation still plays – and will play – an important role: once an XML document has been validated it can be directly processed by the application. It is crucial to keep in mind that XML documents exist without DTDs: an XML document is *well-formed* if its tag structure is correct, and valid (against a DTD) if moreover it complies with this DTD.

However, with the advent of XML as the data-format for both text and data content, DTDs are promised to be used in a broader scope.

From a database perspective [1] for instance, DTDs can be useful to trigger optimizations for XML query languages. For instance some structural knowledge about an XML document permits to resolve path-expressions with wild-cards and perform both horizontal and vertical optimization ([11, 5, 13]). This role of DTDs is similar to *dataguides* [6].

More recently, some research about efficient storage [4] and compression [12] of XML showed that some information about the structure of the document can dramatically improve performance. For instance, knowing that an attribute can have a finite number of values permits to encode it more efficiently using a dictionary.

DTDs can also be extremely useful as meta-information about the document they describe.

The rest of this paper is organized as follows. We first give a brief overview of XML DTDs (Section 2). In Section 3, we explain the methodology of our survey and present some preliminary results about how DTDs are being misused to specify the structure of XML documents. In Section 4 we explain what we think is wrong with the current DTDs and what they should offer. In Section 5, we look at some replacements that have been proposed. In Section 6, we present some future directions of research before we offer some conclusions.

2 DTDs in a nutshell

We briefly describe the structure of a DTD, which can consists of the following items listed below.

Elements represent the tag names that can be used in the document. Element declarations are introduced using `<!ELEMENT >`. Elements can contain sub-elements or be empty. Elements can have some attributes. The structure of sub-elements is defined via a *content-model* built out of operators applied to sub-elements. Elements can be grouped as sequences (`a,b`) or as choices (`a|b`). For every element or group of elements, the content-model can specify its occurrence by using regular expression operators (`?,*,+`). There are also some special case of the content-model: `EMPTY` for an element with no sub-elements; `ANY` for an element that can contain any sub-element; `#PCDATA` for an element that can contain only text. When the element can contain sub-elements mixed with text, the content-model is called *mixed-content*.

Attribute definitions are introduced using `<!ATTLIST >`. Attributes can be of various types such as `ID` for a unique identifier, `CDATA` for text or `NMTOKEN` for tokens. They can be optional (`#IMPLIED`) or mandatory (`#REQUIRED`). Attributes can also be of an arbitrary type defined by a notation, introduced using `<!NOTATION >`. Optionally, attributes can have a default or a constant value (`#FIXED`).

Entity references are constants that can be used inside XML documents[1]. Entity references are introduced using `<!ENTITY name>` and referred to using `&name;`. Entity references can be used inside the DTD itself – to define other entities – and inside documents.

Entity parameters can be seen as text macros that can be used internally inside the DTD: they have no meaning outside of the DTD. They are introduced using `<!ENTITY % name >` and are referred to inside the DTD using `%name;`.

A given XML document can refer to its DTD in 4 different ways, defined by the required mark-up declaration (RMD) `<?XML version="1.0" RMD=''?>`. First it can point to no DTD[2] and corresponds to `RMD='NONE'`. Second, it can point to an external DTD, as a remote resource. Third, it can include an internal DTD, in-lined inside the document (`RMD='INTERNAL'`). Fourth, it can use a combination of both (`RMD='ALL'`).

In the previous description, we have omitted on purpose some details that are not relevant for the results of the survey.

[1]Like `&lt;` and `&gt;` that represent `<` and `>` in HTML.
[2]In this case, only well-formedness matters

3 The DTD survey

We present the details of the survey we have conducted on DTDs and we first describe the methodology. The first step is the **harvesting** of DTDs. Fortunately, repositories are emerging such as Microsoft BizTalk. For this paper, we have been using `http://www.xml.org`. Harvesting is done by hand since the repository points to the web page of the corresponding project and not the DTD itself. In some cases, access requires a registration. We have selected DTDs from different domains (see Figure 3) in order to get a representative sample of XML applications. Unexpectedly, the second step is the **cleaning** of the DTDs. Our experience proved that most of the DTDs are incorrect, with some missing declarations or some typos. This is a paradoxical discovery since DTDs are made to validate XML documents: *Quis custodiet custodes ipsos!* The third step is to **normalize**, by expanding parameter entities and translating the DTD structure into a convenient data-structure. The next step is the **mining** of the DTDs. The term *mining* is actually misleading since in most cases we know what we are looking for. The final step is **reporting and visualization** of the results.

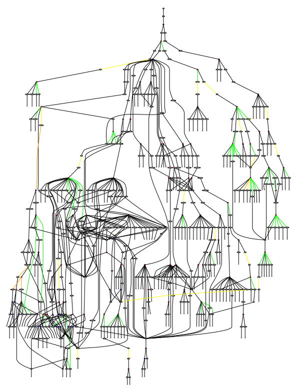

Figure 1: View of a DTD for financial transactions

The experiments were done using `sgrep`[3], Perl, shell scripts, and Java programs based of the IBM XML API. The entity expansion was performed using a modified version of `xmlproc`[4], by Lars M. Garshol. We used Graphviz[5] from AT&T to generate DTD graphs.

[3]`http://www.cs.helsinki.fi/~jjaakkol/sgrep.html`
[4]`http://www.stud.ifi.uio.no/~lmariusg/download/python/xml/xmlproc.html`
[5]`http://www.research.att.com/sw/tools/graphviz`

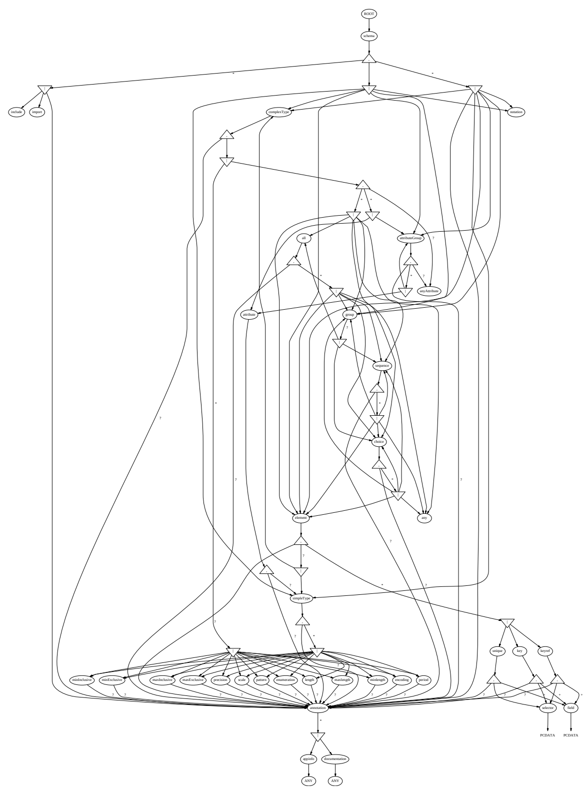

Figure 2: View of the XML-Schema DTD

3.1 What we have been looking at

We are interested in various properties of a DTD. First we look at its size, in terms of number of elements, attributes and entity references.

Second, we look at its structure, in terms of number of root elements, depth and the complexity of the the content-model. In figure 3, CM represents the maximum depth of the content model: 0 for `EMPTY`; 1 for a single element, a sequence or a choice; ...; n for an alternation[6] of sequences and choices of depth n.

Third, we look at some specific aspects such as the use of mixed-content (MX), `ANY`, `IDREF`s, and the kind of attribute decorations used (I,R,F for implied, required and fixed).

Finally, we draw graphs (Figures 1 and 2) to get some insights about DTD patterns. Graphs only display elements; $\triangle$ corresponds to sequences; ∇ to choices.

3.2 Results

Some quantitative results from the survey are presented in Figure 3. The detailed results will be available in the full version of this paper.

As mentioned previously, the first striking observation is that most published DTDs are not correct, with missing elements, wrong syntax or incompatible attribute declarations. This might prove that such DTDs are being used for documentation purposes only and are not meant to be used for validation at all. The reason might be that because of the ad-hoc syntax of DTDs (inherited from SGML), there are no standard tools to validate them. This issue will be addressed by proposals that use XML as the syntax to describe DTDs themselves (see Section 5).

The second remark is that a DTD is not always a connected graph. This is not an issue for validation since the XML document will mention the root that needs to be used for validation. But it is not clear why in such cases the DTD is not split into multiple DTDs.

A third really interesting observation – that we got from visual representations of DTDs (see Figure 4) concerns the encoding tuples. Unlike SGML that offer the `&` operator to create unordered sequences, XML only offer sequences (`","`) or choices (`"|"`). In order to encode a tuple `<a,b,c>`, where SGML would use `(a & b & c)`, XML requires `(a,b,c)|(a,c,b)|(b,c,a)|(b,a,a)|(c,a,b)|(c,b,a)`. The *official workaround* [7] seems to be `(a|b|c)*`, which will validate the tuple but has a totally different meaning!

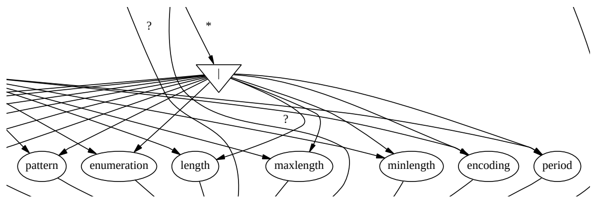

Figure 4: The tuple encoding issue

We also notice that DTD authors try to mimic inheritance and capture modularity via entity references. Unfortunately, this inheritance is purely syntactic and sometimes leads to some cascading mistakes.

The use of mixed-content is a good indication of the kind of application intended for the DTD. No mixed-content implies pure data processing (BSML, OSD) while the presence of mixed content suggests more textual content (XHTML, XMI/UML, Adex).

An intriguing fact is the presence of very complex and *deep* content models (up to depth 5 !).

Some DTDs contain some really strange patterns where for instance empty elements are systematically used instead of attributes.

[6] `(a,(b|(c,d)))` has depth 3

[7] This way to model tuples is actually recommended by some people and is being used for the description of the XML-Schema DTD.

DTD name	Domain	Elem/Attr	MX	CM	ID/IDREF	I/R/F	ANY	Ent. Ref/Para	
Adex	classifieds	365/5194	7	2	366 / 4	4550 / 13 / 4	0	0/17	
BSML	DNA sequencing	111 / 2595	0	3	88 / 105	2490 / 58 / 6	0	5/36	
EcoknowMics	economics	105 / 264	0	2	89 / 0	0 / 93 / 171	0	0/0	
HL7	medical informatics	109 / 252	69	2	102 / 7	115 / 14 / 119	0	1/24	
ICE	content syndication	48 / 157	0	3	0 / 0	82 / 63 / 1	1	0/10	
MusicML	music	12 / 17	0	2	0 / 0	6 / 5 / 0	0	0/12	
OSD	software description	15 / 15	0	1	0 / 0	3 / 11 / 0	0	0/0	
PML	web portals	46 / 293	1	2	0 / 0	90 / 203 / 0	0	0/3	
XMI	UML data modeling	398 / 1636	23	1	119 / 122	1587 / 46 / 1	22	0/2	
XHTML	HTML	77 / 1373	44	5	70 / 3	1344 / 12 / 4	0	252/59	
XML-Schema		37 / 91	0	4	12 / 0	43 / 23 / 2	2	0/46	
Xbel	bookmarks	9 / 13	0	2	3 / 1	8 / 3 / 2	0	0/6	

Figure 3: Some quantitative results for DTDs from `http://www.xml.org`.

Finally, it is clear that most of the features (from the SGML legacy) of DTDs are not being used, such as notations and fancy attribute types. More surprisingly, `ID` and especially `IDREF`s are very infrequently considered (see Figure 3): since `ID` and `IDREF`s are not typed (an `IDREF` can point to any `ID`, people prefer not to use them. These mechanism can always be implemented at the application level.

Some conclusions from this preliminary survey are that: (1) DTD have all sort of shapes and sizes and are used for many diverse purposes; (2) DTD features are not properly understood (some features are never used, and some are misused); (3) there are many ways to do the same things and it is not clear why people are using one solution rather than another; (4) people use *hacks* to solve DTD shortcomings, sometimes for better, sometimes for worse.

A more practical conclusion from our database perspective is that relying on DTDs for storage, compression and optimization seems foolish since DTDs are often a misleading approximation of the intended structure (e.g. tuple encoding example). Moreover, a DTD offers only fuzzy guarantees about the structure of the XML document: `a*` might mean 0, 1 or 20 repetitions. ‘We would therefore recommend to use the actual structure of documents like in [4, 12] rather to bet[8] on the DTD [14].

4 Shortcomings & requirements

The conclusions of the survey show that DTDs – as they are – are not suitable for what we want to do with them.

DTDs are suffering from the heavy SGML ancestry. Even though some complex features have been removed, XML DTDs are still too complex, with text processing rather than data processing in mind.

[8]The author has to acknowledge that before conducting the survey, he was heavily betting on DTDs too.

DTDs are also suffering from a big lack of understanding. For instance, people are still confused about element vs attributes.

But more critically, DTDs seem to suffer from the same shortcomings as the first programming languages. In Figure 5, we ”compare” DTD with the `C` programming language.

XML DTD	Programming language
validation	type-checking
entity references	constants
entity parameters	macros
ANY	`void`
IDREF	`void*`
DTD	header file
|	variant
,	tuple (with order)
?	`nil`
+/*	`list`

Figure 5: DTDs vs programming languages

All these analogies are good news because it means we can get some inspiration from modern programming languages. We briefly list some features that DTDs could benefit from.

From an engineering point of view, DTDs would strongly benefit from a convenient system for modularity. The actual system of entities and catalogue is clumsy and does not really scale. The use of namespaces[9] and packages would be a big improvement.

XML DTDs would also benefit from a more expressive content-model that is suitable for both text and data processing.

As clearly shown by the survey, in the absence of typed references, people are reluctant to use of them. Being able to specify keys and foreign keys is crucial.

Finally, we think that DTDs will soon face the critical problem of versioning, including backward and

[9]Not part pf the XML 1.0 specification

forward compatibility. To the best of our knowledge, this issue has not been elegantly resolved in the S-GML world. Database schemas are well known for evolving and DTDs should be expected to do exactly the same.

5 Replacement for DTDs

After denigrating DTDs and writing down the *wish list* for the *New Document Type Descriptors*, let us look at the current proposals. XML users have realized very early the shortcomings of DTDs and have come up with some extensions such as SOX, DCD, XML-Data, etc. In the rest of this section, we present various approaches to replace DTDs and some recent corresponding proposals. It is worth noting that most of these proposals use XML itself as the syntax, which permits to use the XML tools to check them.

Grammar-based, parsing based approaches: These approaches define the structure of an XML document in terms of production rules, starting from the root of the document, just like DTDs.

XML-Schemas[10] is the official W3C replacement for DTDs. Their goal is to "*constrain and document the meaning, usage and relationships of their* [XML documents] *constituent parts*". The proposal is split into two specifications: one for data-types [18] (parsing oriented) and one for structures [19].
Even though the proposal does not address all the issues pointed out by the survey, it is interesting to note that it brings back a tuple construct (SGML `&`, but with some syntactic restrictions) and offers a replacement for `ANY`. It also addresses the issue of modeling by offering grouping, modularity and restriction/extension mechanisms for data-structures. The proposal also permits to define *identity constraints*, where keys (including composite keys) can be defined using elements and/or attributes.

Another proposal is the *Document Structure Definition* (DSD) [10] which among other things offer context dependent descriptions, constraints and typed references. It also support evolution via *document inclusion and redefinition.*

Constraints: A different approach exemplified by the Schematron [8], is to describe the structure of an XML document using only constraints and pattern-matching. Such constraints are defined by a *context* where they apply and a *predicate* they must satisfy. Both are expressed as XPath expressions[11]. An interesting aspect is that constraints can be used to describe the structure with different granularities. Schematron constraints are much more expressive than the ones offered by XML-Schema's. For instance, it is possible to type references by using a constraint that states that the node reached by following an IDREF is of a given type. It is not clear though if all constraints expressed in the language are decidable.

Type systems: Another proposed approach is to use type systems borrowed from programming languages in order to enforce some structure on XML documents. A type-system for semi-structured data has been proposed in [2]. Embedding XML into functional languages has also been looked at in [7, 21]: in this case the emphasis has been to offer a framework to validate programs written against a given DTD rather than validate documents. Various toolkits also offer some ad-hoc mappings from XML into Java classes.

Other data-models: Finally, there are some proposals to use other representation to describe DTDs such as description logics or UML [9]. For the latter, a mapping from DTDs to UML has been defined and – not surprisingly – the main glitches occur at the level of the content-model.

In most cases, these approaches are complementary and do not address the same kind of issues. It is already interesting to remark that constraint-based and grammar-based approaches are being put together by the XML-Schemas proposals.

6 Future Work

We present some future work that has arisen, sometimes unexpectedly, from this survey.

Systematic mining: As we mentioned above, the term mining is misleading because we somehow knew what we were looking for. The advent of XML gives us the unique opportunity to analyze and compare data modeling strategies[12]. This would hopefully lead to the discovery of XML patterns and the creation of XML modeling *recipes*.

Metrics for DTD complexity: For a given domain are there some DTDs better than others? In the object-oriented community, metrics [3] have been defined and applied to define complexity of programs. Can they be adapted to DTDs?

The meaning of DTDs: XML is claimed to be a self-describing format. By using NLP techniques and ontologies, it would be interesting to see if DTDs (el-

[10]See [17] for a tutorial.

[11]The Schematron is similar to an XSL-T transform that would output validation messages instead of content.

[12]This is to be contrasted with database schema that remain the secret property of organizations.

ement and attributes names) are useful to characterize the domain of a given document. This exercise is not as trivial as it appears: elements and attributes are not always words but acronyms, abbreviations including namespaces that will require smart tokenization programs.

7 Conclusion

By its extensible nature, the XML language imperatively needs a constraint structure that is represented today by DTDs. Unfortunately, DTDs have been designed for a specific domain (text processing applications) which represents a small part of the scope of XML. As a specification tool for XML, DTDs are simply inadequate.

In this paper we have presented the preliminary results of a survey we have started in Fall 1999. Its primary motivation was to better understand DTDs by looking at *how they are actually being used* to describe the structure of XML documents. Surprisingly, not unlike living organisms, XML DTDs have mutated from SGML DTDs into something that tries to fit the requirements of XML (both text and data processing). Because of their inherited shortcomings, XML DTDs have been *hacked* by users, in order to resolve serious issues such as tuple encoding and modularity. Other issues such as reference typing and versioning have simply been postponed, for lack of immediate workarounds.

However, in this survey we have only scratched the surface of the problem: we not only need a better way to capture the structures of XML documents, but also tools and methodologies to define them properly.

It is encouraging though to note that the current proposals to replace DTDs are taking some of these issues into account and offer cleaner constructs to capture what is needed by XML applications.

Finally, even though DTDs are flourishing, the corresponding XML documents are still nowhere to be found. The next interesting question will be to see how XML documents are being instantiated for a given DTD. This will be of special interest to the database community who will be "responsible" for efficiently storing, indexing, querying, mediating and transforming such documents. But this is another story...

Acknowledgements: The author would like to thank Val Tannen, the members of the *Database Research Group* and the *Programming Languages Club* for stimulating discussions about XML, constraints and types; Henry Thompson for a sneak-preview of XML-Schema; Frank Olken for a sneak-preview of XML-Schema constraints.

References

[1] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web : From Relations to Semistructured Data and Xml* . Morgan Kaufmann, 1999.

[2] Peter Buneman and Benjamin Pierce. Union Types for Semistructured Data. Technical Report MS-CIS-99-09, University of Pennsylvania, Apr 1999.

[3] S. R. Chidamber and C. F. Kemerer. A Metrics suite for Object-Oriented Design. *IEEE Transaction on Software Engineering*, 20(6):476–493, 1994.

[4] Alin Deutsch, Mary F. Fernandez, and Dan Suciu. Storing Semistructured Data with STORED. In *SIGMOD*. ACM Press, 1999.

[5] Mary F. Fernandez and Dan Suciu. Optimizing regular path expressions using graph schemas. In *ICDE*. IEEE Computer Society, 1998.

[6] Roy Goldman and Jennifer Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *VLDB'97*. Morgan Kaufmann, 1997.

[7] Haruo Hosoya and Benjamin C. Pierce. XDuce: An XML Processing Language. Available from `http://www.cis.upenn.edu/~hahosoya/papers/xduce-prelim.ps`, Dec 1999.

[8] Rick Jelliffe. The Schematron. Available from `http://www.ascc.net/xml/resource/schematron`, 1999.

[9] W. Eliot Kimber. Using UML To Define XML Document Types. Available from `http://www.drmacro.com/hyprlink/uml-dtds.pdf`, Dec 1999.

[10] N. Klarlund, A. Moller, and M. Schwartzbach. DSD: A Schema Language for XML. Available at `http://www.brics.dk/DSD/`, Nov 1999.

[11] Hartmut Liefke. Horizontal Query Optimization on Ordered Semistructured Data. In *WebDB*, 1999.

[12] Hartmut Liefke and Dan Suciu. XMill: an Efficient Compressor for XML Data. In *SIGMOD*. ACM Press, 2000.

[13] Jason McHugh and Jennifer Widom. Query Optimization for XML. In *VLDB*. Morgan Kaufmann, 1999.

[14] Jayavel Shanmugasundaram at al. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *VLDB*. Morgan Kaufmann, 1999.

[15] W3C. Extensible Markup Language (XML) 1.0. W3C Recommendation 10-February-1998. Available from `http://www.w3.org/TR/1998/REC-xml-19980210`.

[16] W3C. XML Path Language (XPath) 1.0. W3C Recommendation 16 November 1999. Available from `http://www.w3.org/TR/xpath`.

[17] W3C. XML Schema Part 0: Primer. Working Draft 25 February 2000. Available from `http://www.w3.org/TR/xmlschema-0`.

[18] W3C. XML Schema Part 1: Structures. Working Draft 25 February 2000. Available from `http://www.w3.org/TR/xmlschema-1`.

[19] W3C. XML Schema Part 2: Datatypes. Working Draft 25 February 2000. Available from `http://www.w3.org/TR/xmlschema-2`.

[20] W3C. XSL Transformations (XSL-T) 1.0. W3C Recommendation 16 November 1999. Available from `http://www.w3.org/TR/xslt`.

[21] Malcolm Wallace and Colin Runciman. Haskell and XML: Generic Combinators or Type-Based Translation? In *International Conference on Functional Programming*, Paris, France, Sept 1999.