DTDs versus XML Schema: A Practical Study

Geert Jan Bex
Limburgs Universitair Centrum
Diepenbeek, Belgium
gjb@luc.ac.be

Frank Neven
Limburgs Universitair Centrum
Diepenbeek, Belgium
frank.neven@luc.ac.be

Jan Van den Bussche
Limburgs Universitair Centrum
Diepenbeek, Belgium
jan.vandenbussche@luc.ac.be

ABSTRACT

Among the various proposals answering the shortcomings of Document Type Definitions (DTDs), XML Schema is the most widely used. Although DTDs and XML Schema Definitions (XSDs) differ syntactically, they are still quite related on an abstract level. Indeed, freed from all syntactic sugar, XML Schemas can be seen as an extension of DTDs with a restricted form of specialization. In the present paper, we inspect a number of DTDs and XSDs harvested from the web and try to answer the following questions: (1) which of the extra features/expressiveness of XML Schema not allowed by DTDs are effectively used in practice; and, (2) how sophisticated are the structural properties (i.e. the nature of regular expressions) of the two formalisms. It turns out that at present real-world XSDs only sparingly use the new features introduced by XML Schema: on a structural level the vast majority of them can already be defined by DTDs. Further, we introduce a class of simple regular expressions and obtain that a surprisingly high fraction of the content models belong to this class. The latter result sheds light on the justification of simplifying assumptions that sometimes have to be made in XML research.

1. INTRODUCTION

As Document Type Definitions where historically the first means to describe the structure of XML documents, a large number of them can be found on the Web. The growing success of XML, combined with certain shortcomings of DTDs, generated a large number of alternative proposals for the description of schemas, such as RELAX [12], TREX [6], Relax NG [7], DSD [11], and XML Schema [1, 9, 17].
Judging from the number of schemas one can find on the Web, XML Schema seems the most accepted one. The definition of XML Schema is nevertheless quite complicated and the necessity of various constructs is not always very clear. For this reason, we investigate a number of XSDs collected from the Web, and try to determine to what extent the features of XML Schema not occurring in DTDs are used in practice. In the second part of the paper we look at structural properties of schemas. In particular, we show that the vast majority of content models occurring in practice belong to a well-defined class of simple regular expressions.
To facilitate a comparison between the two formalisms, we first describe DTDs and XSDs on a structural level.

Seventh International Workshop on the Web and Databases (WebDB 2004), June 17-18, 2004, Paris, France.

1.1 A structural view of DTDs and XSDs

When dealing with the structure of XML documents only, it is common to view XML documents as finite ordered trees with node labels from some finite alphabet Σ. We refer to such trees as Σ-trees.

DEFINITION 1. *A* DTD *is a pair (d, s) where d is a function that maps Σ-symbols to regular expressions over Σ, and $s \in \Sigma$ is the start symbol. A tree satisfies the DTD if its root is labeled by s and for every node u with label a, the sequence $a_1 \cdots a_n$ of labels of its children matches the regular expression $d(a)$.*

The class of tree languages definable by DTDs is usually referred to as the *local tree languages* [4, 13]. A simple example of a DTD defining the inventory of a store is the following:

$$\begin{array}{lcl} \text{store} & \rightarrow & \text{dvd dvd}^* \\ \text{dvd} & \rightarrow & \text{title price} \end{array}$$

For clarity, in examples we write $a \rightarrow r$ rather than $d(a) = r$.

We next recall the definition of a specialized DTD [15].

DEFINITION 2. *A* specialized DTD *(SDTD) is a 4-tuple $(\Sigma, \Sigma', \delta, \mu)$, where Σ' is an alphabet of* types*, δ is a DTD over Σ' and μ is a mapping from Σ' to Σ. Note that μ can be applied to a Σ'-tree as a relabeling of the nodes, thus yielding a Σ-tree. A Σ-tree t then* satisfies *the SDTD if t can be written as $\mu(t')$ where t' satisfies the DTD δ.*

As SDTDs are equivalent to unranked tree automata [4], the class of tree languages definable by SDTDs is the class of *regular tree languages*. The XML equivalent of that class is captured by the schema language Relax NG [7].

For ease of exposition, we always take $\Sigma' = \{a^i \mid 1 \leq i \leq k_a, a \in \Sigma, i \in \mathbb{N}\}$ for some natural numbers k_a and set $\mu(a^i) = a$.

A simple example of an SDTD is the following:

$$\begin{array}{lcl} \text{store} & \rightarrow & (\text{dvd}^1 + \text{dvd}^2)^*\text{dvd}^2(\text{dvd}^1 + \text{dvd}^2)^*\text{dvd}^2 \\ & & \quad \cdot (\text{dvd}^1 + \text{dvd}^2)^* \\ \text{dvd}^1 & \rightarrow & \text{title price} \\ \text{dvd}^2 & \rightarrow & \text{title price discount} \end{array}$$

Here, dvd^1 defines ordinary DVDs while dvd^2 defines DVDs on sale. The rule for store specifies that there should be at least two of the latter.

The following restriction on SDTDs corresponds to the expressiveness of XML Schema [13]:

DEFINITION 3. *A* single-type SDTD *is an SDTD* $(\Sigma, \Sigma', (d, s), \mu)$ *with the property that no regular expression* $d(a)$ *has occurrences of types of the form* b^i *and* b^j *with the same* b *but with different* i *and* j*.*

The example SDTD above is not single type as both dvd^1 and dvd^2 occur in the rule for store. It is shown by Murata et al [13], that the class of trees defined by single-type SDTDs is strictly between the local and the regular tree languages. An example of a single-type grammar is given below:

$$\begin{array}{lcl} \text{store} & \rightarrow & \text{regulars discounts} \\ \text{regulars} & \rightarrow & (\text{dvd}^1)^* \\ \text{discounts} & \rightarrow & \text{dvd}^2\ \text{dvd}^2\ (\text{dvd}^2)^* \\ \text{dvd}^1 & \rightarrow & \text{title price} \\ \text{dvd}^2 & \rightarrow & \text{title price discount} \end{array}$$

Although there are still two element definitions dvd^1 and dvd^2, they can only occur in a different context, regulars and discounts respectively.

1.2 Related work

In 2000, while the XML Schema specification was still under development, Sahuguet [16] investigated a sample of DTDs to determine the shortcomings of the Document Type Definition specification. What he found missing has been remedied in XML Schema. Moreover, XML Schema introduces many features not envisioned by Sahuguet. One of the goals of this paper is to investigate to what measure these features are used in real world XSDs currently available.

Choi [5] has tried to identify features that are characteristic for DTDs used to describe three types of schemas: application, data and meta-data related. He created a content model classification based on syntactic features and considered several measures for the complexity of a DTD. In this paper we extend his classification of content models and consider XSDs beside DTDs.

1.3 Overview

Based on the characterizations given above, it is clear that DTDs and XSDs are both grammar based where XML Schema in addition is extended with a restricted typing mechanism. In Section 3, we inspect the use of that typing mechanism in practice together with another notion added to XML Schema: derived types. In Section 4, we compare the properties of the grammars underlying real DTDs and XSDs. First, we discuss in the next section our dataset and methodology.

2. DATASET AND METHODOLOGY

We have tried to gather a representative sample of DTDs and XSDs. The XML Cover pages [8] have proved to be an excellent repository so almost all schemas in our sample have been obtained automatically from that source by using a simple web crawler. To ensure that the sample contains a base set of quality DTDs and XSDs, a number of the W3C standards have been included. Among those are the DTDs for MathML, SVG, XHTML, XML Schema and SMIL and the XSD for RDF and XML Schema.
All in all, 109 DTDs and 93 XSDs have been obtained. Although some 600 DTDs and XSDs are mentioned on the Cover Pages, only these $109 + 93$ were actually available for download, thus illustrating once again the transient nature of the Internet and its various technologies. All 93 XSDs have been used for the analysis in Sections 3.2 and 3.3 while unfortunately only 30 of the 93 XSDs can be used for most of the analysis in Sections 3.1 and 4, due to various errors discussed in Section 6 below. In the appendix, we provide a list of some of the XSDs we used.

3. EXPRESSIVENESS OF XML SCHEMA

3.1 Single-type

The formal taxonomy presented in Section 1, elicits the following question: is the expressive power of single-type SDTDs actually used in real-world XSDs, and if so, to what extent? Given that XSDs tend to be a bit unwieldy due to their inherent verbosity, it is interesting to identify use cases for the distinctive features of single-type SDTDs versus DTDs. Those use cases might suggest a simpler formalism that finds an appropriate balance between designer-friendliness and expressive power.

Surprisingly, most XSDs in our sample turn out to define local tree languages, that is, can actually be defined by DTDs. Only 5 out of 30 are true single-type SDTDs, which corresponds to approximately 15%. There might be several possible reasons for this low percentage. A first possibility is that expressiveness beyond local tree languages is simply rarely needed. Another explanation might be that due to the relatively new nature of XML Schema and its complicated definition most users have no clear view on what can be expressed.

All five examples we found are of the following form:

$$\begin{array}{lcl} p & \rightarrow & \ldots a^1 \ldots \\ q & \rightarrow & \ldots a^2 \ldots \\ a^1 & \rightarrow & expr_1 \\ a^2 & \rightarrow & expr_2 \end{array}$$

The meaning of a^1 and a^2 is the following: when the parent of an A is P (resp. Q) use the rule for A^1 (resp. A^2). No other use cases have been found in the sample.

3.2 Derived types

Two kinds of types are provided by XML Schema: simple and complex types. The former describes the character data an element can contain (cfr. #PCDATA in DTDs) while the latter specifies which elements may occur as children in a given element.

	simple type (%)	complex type (%)
extension	27	37
restriction	73	7

Table 1: Relative use of derivation features in XSDs

XML Schema facilitates derivation of new types from existing types via two mechanisms: extension and restriction. Both simple and complex types can be extended or restricted. The four cases are introduced below; for a thorough discussion, we refer to the W3C specification [9, 17].

- A complex type can be derived from a simple type by extension to add attributes to elements.
- A complex type can be extended to add a sequence of additional elements to its content model or to add attributes.
- Restricting a simple type limits the acceptable range of values for that type. For example, one can enforce that a phone number should consist of three digits, a dash, followed by six more digits.
- Restricting a complex type is similar to restricting a simple type in that it limits the set of acceptable subtrees.

Table 1 lists the number of XSDs using a particular derivation feature. Note that in this section we used all 93 XSDs we retrieved since conformance was not an issue for this analysis.

Approximately one fifth of the XSDs considered do not construct new types through derivation at all. Extension is mostly used to define additional attributes (58%); elements are added to a content model to a lesser degree (42%).
As expected, restriction of complex elements is hardly used (7%). A typical example of the latter mechanism is the modification of the multiplicity of an element:
maxOccurs ="unbounded" to maxOccurs="1".
The statistics also show that only just over a third of the XSDs (37%) use extension of complex types, a feature that parallels inheritance in the object orientation paradigm. This might indicate that the data modeled by these XSDs is often too simple to merit such a (relatively) sophisticated approach. It might also be "underused" due to the relative novelty of XML Schema, since many data architects are trained to think in terms of relational data rather than object orientation. Extension of simple types occurs in 27% of the XSDs.
Restriction of simple types is most heavily used (73%), which comes as no surprise since it allows a much more fine-grained control over the content of an element, rather than the unrestrictive #PCDATA DTDs are limited to, thus alleviating one of the more glaring shortcomings of DTDs.

Several mechanisms have been defined to control type creation by derivation. The final attribute for type definitions indicates that the type can not either be restricted, extended or both. Only 6 out of the 93 XSDs use this feature. As opposed to finalizing a type definition, it can also be declared abstract implying that one should derive new types from it. Although slightly more common, it is only used in 11 XSDs in our sample.

As a general rule, derived types can occur anywhere in the content model where the original type is allowed. However, this can be prevented by applying the block attribute to the original type. As for the final attribute, replacement by either restricted, extended or both types can be blocked. Blocking is used in 2 out of the 93 XSDs.

The fixed attribute that is usually used to indicate that an element or an attribute is restricted to a specific value also serves a purpose in the context of derivation from simple types. It can be applied to fix a facet of a simple type (e.g. the length of a xsd:string) in a restrictive type derivation. Only a single XSD uses the fixed attribute in this sense.

Although not directly related to derivation, the substitution group feature nevertheless deserves to be mentioned here. Elements are declared members of a substitution group using the substitutionGroup attribute with an element name as value and may occur instead in the content model akin to derived types. Substitution groups are used in 10 out of 93 XSDs.

3.3 Additional features

XML Schema defines various additional features with respect to DTDs, see [9, 18] for an introduction.

One feature of SGML DTDs that was lost to XML DTDs is the &-operator that specifies that all elements must occur but that their order is not significant. Obviously this can be simulated in an XML DTD by explicitly listing all orders (e.g. $a_1 \& a_2 \& a_3 \equiv a_1a_2a_3 \mid a_2a_3a_1 \mid \ldots \mid a_3a_2a_1$, so a choice between 6 cases), but this doesn't exactly improve the clarity of the content model. XML Schema restores this feature by defining the xsd:all element. However, only 4 out of 93 XSDs use this operator.

Elements in an XML document can be identified using ID attributes and referred to by IDREF or IDREFS. This feature is part of the XML 1.0 specification [2] and is supported by DTDs. These IDs are unique throughout a document and are the only attributes with such a restriction for DTDs. In XML Schema, any element or attribute can be declared to require a unique value by selecting the relevant nodes using an XPath expression and specifying the list of fields that combined should have a unique value. In our sample, 6 XSDs out of 93 use this feature, all applied to a single field.

Referring to elements can also be accomplished by key/keyref pairs. Using a reference to a key implies that the element with the corresponding key should exist in the document. This feature is reminiscent of the foreign key concept in relational databases. It is used in 4 XSDs in our sample.

An interesting feature introduced in XML Schema is the use of namespaces for modularity. This allows to use elements and types defined in the current XSD that are defined elsewhere without fear of name clashes. Apart from the obvious inclusion of the XML Schema namespace, 20 XSDs in our sample used this mechanism.

A last feature to discuss is the ability to redefine types and groups. It should be noted that W3C's primer on XML Schema cautions against the use of this feature since it may break type derivation without warning. It turns out that the authors of the XSDs in our sample set heeded this advice and avoided redefine altogether.

4. REGULAR EXPRESSION CHARACTERIZATION

The second question we try to answer is how sophisticated regular expressions tend to be in real world DTDs and XSDs. If simple expressions make up the vast majority of schema definitions, it is worthwhile to take this into account when developing implementations of XML related applications and fine-tune algorithms to take advantage of this simplicity whenever possible.

In order to facilitate the analysis some preprocessing was performed. For the DTDs parsed entities were resolved and conditional sections included/excluded as appropriate. Since we are only concerned with the schema structure, the DTD element definitions were extracted and converted to a canonical form, which abstracts away the actual node labels and replaces them by canonical names c_1, c_2, c_3, ... For example,

```
<!ELEMENT lib ((book | journal )*)>
```

is represented by a canonical form $(c_1 \mid c_2)^*$ to preserve only the structure related DTD information.

The XSDs were preprocessed using XSLT to the canonical representation mentioned above for DTDs. To capture multiplicity constraints, '?' is used, e.g. for an element a with minOccurs="1", maxOccurs="3", \lstinlinea a? a?— is substituted. This approach allows us to reuse much of the software developed to analyze DTDs for XSDs as well.

For all DTDs, there is a total of 11802 element definitions which reduce to 750 distinct canonical forms. The 1016 element definitions in the XSDs yield 138 distinct canonical forms, totaling 838 for both types of schemas combined. The majority of these can be classified in one of the categories of "simple expressions", which are subclasses of the expressions studied by Martens, Neven, and Schwentick [14].

DEFINITION 4. *A* base symbol *is a regular expression a, $a?$, or a^* where $a \in \Sigma$; a* factor *is of the form e, e^*, or $e?$ where e is a disjunction of base symbols. A* simple regular expression *is ε, $\emptyset$, or a sequence of factors.*

The following is an example of a simple regular expression: $(a^* + b^*)(a + b)?b^*(a + b)^*$.

We introduce a uniform syntax to denote subclasses of simple regular expressions by specifying the allowed factors. We distinguish base symbols extended by ? or $*$. Further, we distinguish between factors with one disjunct or with arbitrarily many disjuncts; the latter is denoted by $(+ \cdots)$. Finally, factors can again be extended by $*$ or ?. For example, we write $\mathrm{RE}((+a)^*, a?)$ for the set of regular expressions $e_1 \cdots e_n$ where every e_i is $(a_1 + \cdots + a_n)^*$ for some $a_1, \ldots, a_n \in \Sigma$ and $n \geq 1$, or $a?$ for some $a \in \Sigma$. Table 2 provides an overview.

Factor	Abbr.	Factor	Abbr.
a	a	$(a_1 + \cdots + a_n)^*$	$(+a)^*$
a^*	a^*	$(a_1 + \cdots + a_n)?$	$(+a)?$
$a?$	$a?$	$(a_1^* + \cdots + a_n^*)$	$(+a^*)$
$(a_1 + \cdots + a_n)$	$(+a)$	$(a_1^* + \cdots + a_n^*)^*$	$(+a^*)^*$

Table 2: Possible factors in simple regular expressions and how they are denoted ($a, a_1, \ldots, a_n \in \Sigma$).

We analyze the DTDs and XSDs to characterize their content models according to the subclasses defined above. The result is represented in Table 3 that lists the non-overlapping categories of expressions having a significant population (i.e. more than 0.5%). Two prominent differences between DTDs and XSDs immediately catch the eye: XSDs have (1) more simpleType elements (denoted by #PCDATA) and (2) less expressions in the category $\mathrm{RE}(a, (+a)^*)$. The first difference is due to the fact that it pays to introduce more distinct simpleType elements in XSD since thanks to type restriction, it is now possible to fine tune the specification of an element's content (cfr. the discussion in Section 3.2). The second distinction is most probably due to the nature of the XSDs in the sample since those describing data are overrepresented with respect to those describing meta documents [5]. The latter tend to have more complex recursive structures than the former.

To gauge the quality of our sample of XSDs, we compared DTDs and XSDs using several of the measures proposed by Choi [5]. No significant differences between the two samples are observed, which is confirmed by an additional measure in Figure 1, the density of XSDs. The density of a schema is defined as the number of elements occurring in the right hand side of its rules divided by the number of elements. DTDs and XSDs do not fundamentally differ in this respect. Several other measures such as the width and depth of canonical forms viewed as expression trees show no significant differences.

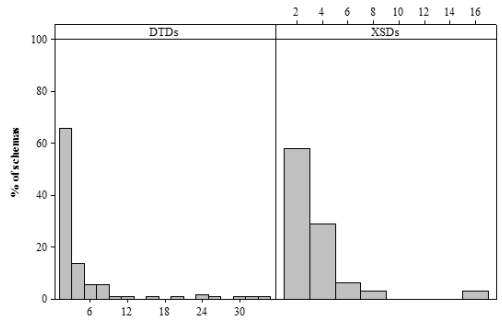

Figure 1: Fraction of DTDs (left) and XSDs (right) versus their density

More importantly though, it is clear that the vast majority of expressions are simple, i.e. 92% and 97% of all element definitions in DTDs and XSDs respectively. Figure 2 shows the fraction of DTDs and XSDs versus the fraction of their

	DTDs (%)	**XSDs** (%)
`#PCDATA`	34	48
`EMPTY`	16	10
`ANY`	1	0
RE(a)	5	5
RE$(a, a?)$	2	10
RE(a, a^*)	8	10
RE$(a, a?, a^*)$	1	4
RE$(a, (+a))$	3	3
RE$(a, (+a)?)$	0	1
RE$(a, (+a)^*)$	20	2
RE$(a, (+a)?, (+a)^*)$	0	1
RE$(a, (+a^*)^*)$	0	2
total simple expr.	92	97
non-simple expr.	8	3

Table 3: Relative occurrence of various types of regular expressions given in % of element definitions

simple content models: the majority of documents have 90% or more simple content models.

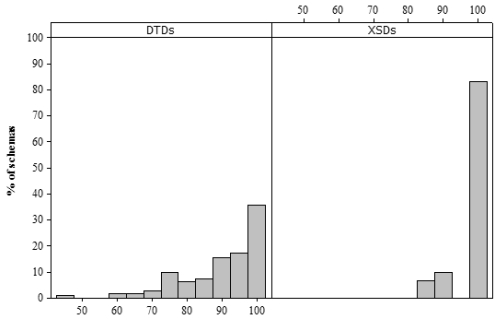

Figure 2: Fraction of DTDs (left) and XSDs (right) having a given % of simple expression content models

The relative simplicity of most DTDs and XSDs is further illustrated by the star height that is given in Table 4. The star height of a regular expression is the maximum nesting depth of Kleene stars occurring in the expression, e.g. 2 for the last example given below, 1 for all others. Content models with star height larger than 1 are very rare. No significant differences are observed between DTDs and XSDs, except for the star height but this is consistent with the relative abundance of RE$(a, (+a)^*)$ type of expressions in DTDs with respect to XSDs.

star height	**DTDs** (%)	**XSDs** (%)
0	61	78
1	38	17
2	1	4
3	0	≈ 0

Table 4: Star height observed in DTDs and XSDs

In a sense this should not come as too great a surprise: DTDs and XSDs model data that reflect real world entities. Mostly those entities are subject to simple relations among one another such as `is-a`, or `is-part` relations (pertainym[1], holonym/meronym relations) that are very often quite simple to express.

[1]meaning relating to or pertaining to

Some randomly chosen examples of non-simple regular expressions that we encountered follow:

$$c_1{}^+ \mid (c_2?{c_3}^+)$$
$$(c_1c_2?c_3?)?c_4?(c_5 \mid \ldots \mid c_{18})^*$$
$${c_1}?({c_2}{c_3}?)?(c_4 \mid \ldots \mid c_{44})^*{c_{45}}^+$$
$${c_1}?{c_2}{c_3}?{c_4}?({c_5}^+ \mid ((c_6 \mid \ldots \mid c_{61})^+{c_5}^*))$$
$$c_1(c_2 \mid c_3)^*(c_4, (c_2 \mid c_3 \mid c_5)^*)^*$$

5. SCHEMA AND AMBIGUITY

The XML 1.0 specification published by the W3C [2] requires schema definitions to be one-unambiguous, i.e. that all regular expressions in the grammar's rules are deterministic in the following sense [3]: a regular expression is *one-unambiguous* iff the corresponding Glushkov automaton is deterministic. Note that the terminology is somewhat confusing in the literature: in the context of SGML 'unambiguous' is used to denote this feature while Choi [5] refers to it as 'deterministic'.

We checked whether the DTDs and XSDs in our sample respect this requirement and find that they almost all do. IBM's XML Schema Quality Checker (SQC) [10] reported 3 out of 93 XSDs as having one or more ambiguous content models (see also Section 6). For DTDs, a first exception is a regular expression of the following type: $(\ldots \mid c_i \mid \ldots \mid c_i \mid \ldots)^*$ that occurred in several DTDs. However, this is merely a typo, not a design feature.

A second type of one-unambiguous regular expression proves to be more interesting: $c_1c_2?c_2?$. The designer's intention is clearly to state that c_2 may occur zero, one or two times.

The latter example illustrates a shortcoming of DTDs that has been addressed in XML Schema. Element definitions in the latter formalism allow the specification of the number of times an element can occur using the minOccurs and maxOccurs attributes. The specification for the example above would be captured by the following snippet of XML Schema (with slight abuse of notation):

```
<xsd:sequence>
  <xsd:element name="c1" type="t1"/>
  <xsd:element name="c2" type="t2"
    minOccurs="0" maxOccurs="2"/>
</xsd:sequence>
```

We found three XSDs defining non-deterministic (or ambiguous) content models. Two canonical forms are found: ${c_1}?(c_1 \mid c_2)^*$ and $(c_1c_2) \mid (c_1c_3)$.

6. ERRORS

It was a bit disappointing to notice that a relatively large fraction of the XSDs we retrieved did not pass a conformance test by SQC. As mentioned in Section 2, only 30 out of a total of 93 XSDs were found to be adhering to the current specifications of the W3C [17]. We decided to use only conforming XSDs for those parts of the analysis that require

conversion to canonical form to ensure correct processing by our software.
Often, lack of conformance can be attributed to growing pains of an emerging technology: the SQC validates according to the 2001 specification and 19 out of the 93 XSDs have been designed according to a previous specification. Some simple types have been omitted or added from one version of the specs to another causing the SQC to report errors.
Some errors concern violation of the Datatypes part of the specification [1]: regular expressions restricting xsd:string are malformed.
Some XSDs violate the XML Schema specification by e.g. specifying a type attribute for a complexType element or leaving out the name attribute for a top-level complexType element.

7. CONCLUSION

Our analysis has shown that many features defined in the XML Schema specification are not widely used yet, especially those that are related to object oriented data modeling such as derivation of complex types by extension. Most importantly, it turns out that almost all XSDs are local tree grammars, i.e. proper single type grammars are rarely used. The expressive power encountered in real world XSDs turns out to be mostly equivalent to that of DTDs. Hence it seems that — barring some exceptions — the current generation of XSDs could just as well have been written as DTDs from the point of view of structure. This might change in the future, as acceptance of a relatively new technology increases, or it might be a symptom that the level of sophistication offered by XML Schema is simply unnecessary for many applications.
The data type part of the XML Schema specification is heavily used though since it alleviates a glaring shortcoming of DTDs, namely the ability to specify the format and type of the text of an element. This is accomplished through restriction of simple types.
The content models specified in both real world DTDs and XSDs tend to be very simple. For XSDs, 97% of all content models can be classified in the categories of simple expressions we identified. This observation can guide software engineers when developing new implementations of XML related tools and applications, for instance by avoiding optimizations for complex cases that rarely occur in practice.

8. REFERENCES

[1] P. Biron and A. Malhotra. *XML Schema part 2: datatypes*. W3C, May 2001, `http://www.w3.org/TR/xmlschema-2/`

[2] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0*. W3C, 3 edition, February 2004, `http://www.w3.org/TR/2004/REC-xml-20040204/`

[3] A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and computation*, 140(2):229–253, 1998.

[4] A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree languages over non-ranked alphabets (draft 1). Unpublished manuscript, 1998.

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

[6] J. Clark. *TREX - Tree Regular Expressions for XML: language specification*, February 2001, `http://www.thaiopensource.com/trex/spec.html`

[7] J. Clark and M. Murata. *RELAX NG Specification*. OASIS, December 2001, `http://www.oasis-open.org/committees/relax-ng/spec-20011203.html`

[8] R. Cover. The cover pages, 2003, `http://xml.coverpages.org/`

[9] D. Fallside. *XML Schema part 0: primer*. W3C, May 2001, `http://www.w3.org/TR/xmlschema-0/`

[10] IBM corp. *XML Schema Quality Checker*, 2003. `http://www.alphaworks.ibm.com/tech/xmlsqc`

[11] A. Møller. *Document Structure Description 2.0*. BRICS, 2003, `http://www.brics.dk/DSD/dsd2.pdf`

[12] M. Murata. Document description and processing languages – regular language description for XML (RELAX): Part 1: RELAX core. Technical report, ISO/IEC, May 2001.

[13] M. Murata, D.Lee, M. Mani, and K. Kawaguchi. Taxonomy of xml schema languages using formal language theory. To be submitted to ACM TOIT, 2003.

[14] W. Martens, F. Neven and T. Schwentick Complexity of Decision Problems for Simple Regular Expressions. Submitted.

[15] Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *PODS proceedings*, pages 35–46, 2000.

[16] A. Sahuguet. Everything you ever wanted to know about DTDs, but were afraid to ask. In *Proceedings of WebDB 2000*, 2000.

[17] H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. *XML Schema part 1: structures*. W3C, May 2001, `http://www.w3.org/TR/xmlschema-1/`

[18] E. van der Vliet. *XML Schema*. O'Reilly, Cambridge, 2002.

APPENDIX

A list of XSDs used in the regular expression and single-type analysis (with number of definitions in brackets) and a few of the XSDs considered in the other parts of the paper.

DSML v2 (1), EPAL-cs-xacml-schema-policy (34), EPAL-epal-interface (12), epal-interface (12), extensions (13), ipdr 2.5 (14), mets (1), OAI DC (1) OAI GetRecord (9), ODRL-EX v1.0 (25), ODRL-EX v1.1 (23), PersonName v1.2 (8), PIDXLib-2002-02-14 v1.0 (255), PMXML2 (1), PostalAddress v1.2 (16), RIXML2 (1), simpledc (15), TC-1025 schema v1.0 xpdl (91), UKGovTalk-BS7666 v1.2 (68), VRXML 20021204 (43), wsrp v1.0 types (1), WS-Security-Schema-xsd-20020411 (7), wsui (26), xgmml (8), xpdl (91), BPML, GenXML v1.0, GML Base, HEML, LogML, MPEG21, PSTC-CS v1.0, RDF, UDDI v2.0, XML Schema