
DTD+RE

François PERRAD <francois.perrad(at)gadz.org>

translated by Martin Fuzzey, the original version was published at XMLfr.

Table of Contents

Introduction	1
Specification	2
Entity handling	2
Implicit extensions to the regular expression	3
Modifier	3
Predefined parameter entities	3
Example 1	4
Example 2	6
Example 3	8
Conclusion	9
References	10

Introduction

The idea of adding regular expressions to XML DTDs occurred to me some time ago. However it was my recent reading of two books : *Mastering Regular Expressions* by Jeffrey E. F. Friedl and *XML Schema* by Eric van der Vlist which encouraged me to formalise it in this article.

This idea occurred naturally as I have a very grammatical view of DTDs, certainly due to memories of **lex** and **yacc**.

[XML] is a powerful language for describing many types of data. But it above all a means of exchanging them, hence a transfer syntax. To be understandable, this data must respect a schema, that is, comply with an abstract syntax. The DTD was the first type of schema. DTDs do not themselves use XML.

I find it hard to understand why new schema types such as [WXS] and [RELAX NG] whose purpose is to describe an abstract syntax do so using XML which is a transfer syntax. The result is lengthy and, although human readable (one of the strengths of XML), the *design* is unintelligible. These two approaches demonstrate one of XML's limits, it can represent anything, but not always efficiently. Furthermore, James Clark has recently added a compact syntax to [RELAX NG] with the main aim of maximising readability.

My aim is not to replace these new types of schema. They are based on an object oriented approach and are a good fit for some requirements.

My aims are :

- retain the simplicity of the DTD and its grammatical approach.
- increase the validation level

Using the grammatical model, a DTD allows good validation of the syntax structure (the role of **yacc**), but offers very few possibilities for the tokens (the role of **lex**) : either a set of predefined values or freeform or almost freeform text like NMTOKEN, but nothing comparable to a regular expression pattern.

Hence the simple idea of adding regular expression to a DTD and the name DTD+RE.

However, this approach does not meet the third objective of XML's design, that is:

- XML shall be compatible with [SGML]

Specification

Only three production rules of the XML grammar are modified. They are production rule 46 ([XML] 3.2 Element Type Declarations) and production rules 55 and 56 ([XML] 3.3 Attribute-List Declarations).

They are modified as follows :

```
[46] contentspec ::= 'EMPTY' | 'ANY' | Mixed | children
                  / 'REGEX' S RegexType

[55] StringType  ::= 'CDATA' / RegexType

[56] TokenizedType ::= 'ID' / 'ID_REGEX' S RegexType | 'IDREF' | 'IDREFS'
                    | 'ENTITY' | 'ENTITIES' | 'NMTOKEN' | 'NMTOKENS'
```

The following production rules are added :

```
RegexType      ::= '/' regex '/' RegexModifier ?
RegexModifier  ::= 'i'
```

An attribute of type ID_REGEX must respect the same validity constraints as an attribute of type ID.

regex is an extended regular expression (ERE) as defined by [POSIX] (volume Base Definitions, Chapter 9 - Regular Expressions). The internationalisation properties defined by [POSIX] must be taken into account.

If the '/' character, which is used as a delimiter, occurs inside a regular expression it must be escaped as "\/". This escaping must be removed before passing the expression to the regular expression engine.

Entity handling

All parameter-entity references must be replaced by their value before passing the expression to the regular expression engine (without adding a space, as for a literal).

```
<!ENTITY % oct_dec "([0-1]?[0-9][0-9]?|2[0-4][0-9]|25[0-5])" >
      <!-- range [0..255] -->
<!ENTITY % ip_dec "%oct_dec;\.%oct_dec;\.%oct_dec;\.%oct_dec;" >
<!ATTLIST mach ip /%ip_dec;/ "127.0.0.1" >
```

Use of parameter entities allows the regular expression to be split up into fragments, which has two advantages :

- improved readability of the regular expression (this has always been a problem with regular expressions)
- simplified reuse of the expression

However, these references can only be used in external DTDs (well-formedness constraint [XML] section 2.8).

The predefined XML entities (< > & ' and ") must not be used within a regular expression (nor, more generally within the DTD section). The characters < > & ' " must be used directly.

On the other hand, the '%' character must be doubled within a regular expression to avoid confusion with the start of a parameter entity.

```
<!ENTITY % pe "PE" >
<!ATTLIST elt attr1 /%pe;/ #FIXED "PE" > <!-- regex: PE -->
<!ATTLIST elt attr2 /%%pe;/ #REQUIRED > <!-- regex: %pe; -->
<!ATTLIST elt attr3 /%%pe;/ #IMPLIED > <!-- regex: %PE -->
```

Implicit extensions to the regular expression

If `xml:space='default'`, the prefix `"^[:space:]*"` and the suffix `" [:space:]*$"` are added to the regular expression.

If `xml:space='preserve'`, the prefix `"^"` and the suffix `"$"` are added to the regular expression.

It is unnecessary to explicitly specify an anchor in a regular expression. Furthermore, the anchor `"$"` is interpreted as the end of the element or the value of an attribute and not as the end of line. End of line characters may be used in a regular expression and are matched by the expressions `"."` and `" [:space:]"` .

Modifier

The value `'i'` may be used to make the regular expression case insensitive.

```
<!ELEMENT bool REGEX /(true|false)/i >
```

Predefined parameter entities

These entities correspond to some of the predefined types of [WXS].

```
<!ENTITY % re._boolean "(true|false|1|0)" >
<!ENTITY % re.boolean "/%re._boolean/" >
<!ENTITY % re._integer "[+\\-]?[:digit:]+" >
<!ENTITY % re.integer "/%re._integer/" >
<!ENTITY % re._decimal "[+\\-]?([:digit:]+(\\.[:digit:]+)?|\\.[:digit:]+)" >
<!ENTITY % re.decimal "/%re._decimal/" >
<!ENTITY % re._float "%re._decimal;([Ee][+\\-]?[:digit:]+)?" >
<!ENTITY % re.float "/%re._float/" >
<!ENTITY % re._language "([:lower:]{2,3}(-[:upper:]{2})?|[ix]-.*)" >
<!ENTITY % re.language "/%re._language/" >
<!ENTITY % re.__date "[:digit:]{4}-[:digit:]{2}-[:digit:]{2}" >
<!ENTITY % re.__time "[:digit:]{2}:[:digit:]{2}:[:digit:]{2}" >
<!ENTITY % re.__tz "Z|[+\\-][:digit:]{2}:[:digit:]{2}" >
<!ENTITY % re.__datetime "%re.__date;T%re.__time;(%re.__tz)?" >
<!ENTITY % re.datetime "/%re.__datetime/" >
```

```

<!ENTITY % re._date "%re.__date;(%re.__tz;)" >
<!ENTITY % re.date "/%re._date;/" >
<!ENTITY % re._time "%re.__time;(%re.__tz;)" >
<!ENTITY % re.time "/%re._time;/" >
<!-- URI : RFC2396 -->
<!ENTITY % re.__escaped "[%:xdigit:]{2}" >
<!ENTITY % re.__mark "[\_\.\!~\*'\\(\)]" >
<!ENTITY % re.__unreserved "([:alnum:]|%re.__mark;)" >
<!ENTITY % re.__reserved "[;\./\?:@&=\+\$,]" >
<!ENTITY % re.__uric
    "(%re.__reserved;|%re.__unreserved;|%re.__escaped;)" >
<!ENTITY % re.__uric_no_slash
    "(%re.__unreserved;|%re.__escaped;|[;\?:@&=\+\$,])" >
<!ENTITY % re._pchar "(%re.__unreserved;|%re.__escaped;|[:@&=\+\$,])" >
<!ENTITY % re._param "%re._pchar;*" >
<!ENTITY % re._segment "%re._pchar;*(;%re._param;)*" >
<!ENTITY % re._path_segments "%re._segment;(\/%re._segment;)*" >
<!ENTITY % re._abs_path "\/%re._path_segments;" >
<!ENTITY % re._opaque_part "%re.__uric_no_slash;%re.__uric;*" >
<!ENTITY % re._path "(%re.__abs_path;|%re._opaque_part;)" >
<!ENTITY % re._port "[:digit:]*" >
<!ENTITY % re._IPv4address
    "[:digit:]+\.[:digit:]+\.[:digit:]+\.[:digit:]" >
<!ENTITY % re._toplabel "([:alpha:]|[:alpha:]([:alnum:]-)*)[:alnum:]" >
<!ENTITY % re._domainlabel
    "([:alnum:]|[:alnum:]([:alnum:]-)*)[:alnum:]" >
<!ENTITY % re._hostname "(%re._domainlabel;\.)*%re._toplabel;[\.]?" >
<!ENTITY % re._host "(%re._hostname;|%re._IPv4address;)" >
<!ENTITY % re._hostport "%re._host;(:%re._port;)" >
<!ENTITY % re._userinfo
    "(%re.__unreserved;|%re.__escaped;|[:;&=\+\$,])*" >
<!ENTITY % re._server "((%re._userinfo;@)?%re._hostport;)" >
<!ENTITY % re._reg_name
    "(%re.__unreserved;|%re.__escaped;|[\$,;:@&=\+])+" >
<!ENTITY % re._authority "(%re._server;|%re._reg_name;)+" >
<!ENTITY % re._scheme "[:alpha:]([:alnum:]|[\-\.\_])*" >
<!ENTITY % re._rel_segment
    "(%re.__unreserved;|%re.__escaped;|[;@&=\+\$,])+" >
<!ENTITY % re._rel_path "%re._rel_segment;(%re._abs_path;)" >
<!ENTITY % re._net_path "\/\/%re._authority;(%re._abs_path;)" >
<!ENTITY % re._query "%re.__uric;*" >
<!ENTITY % re._hier_part
    "(%re._net_path;|%re._abs_path;)(\?%re._query;)" >
<!ENTITY % re._absoluteURI
    "%re._scheme;:(%re._hier_part;|%re._opaque_part;)" >
<!ENTITY % re._relativeURI
    "(%re._net_path;|%re._abs_path;|%re._rel_path;)(\?%re._query;)" >
<!ENTITY % re._fragment "%re.__uric;*" >
<!ENTITY % re._uri
    "(%re._absoluteURI;|%re._relativeURI;)(#%re._fragment;)" >
<!ENTITY % re.uri "/%re._uri;/" >

```

The following entity allows the same DTD to be compatible with parsers regardless of their support or otherwise for DTD+RE

```

<!ENTITY % xml-dtd-regex "INCLUDE" >

```

Example 1

For this first example, I reuse the one from the book *XML Schema* by Eric van der Vlist.

Here is the document instance :

```

<library>

```

```

<book id="b0836217462" available="true">
  <isbn>0836217462</isbn>
  <title xml:lang="en">Being a Dog Is a Full-Time Job</title>
  <author id="CMS">
    <name>Charles M Schulz</name>
    <born>1922-11-26</born>
    <dead>2000-02-12</dead>
  </author>
  <character id="PP">
    <name>Peppermint Patty</name>
    <born>1966-08-22</born>
    <qualification>bold, brash and tomboyish</qualification>
  </character>
  <character id="Snoopy">
    <name>Snoopy</name>
    <born>1950-10-04</born>
    <qualification>extroverted beagle</qualification>
  </character>
  <character id="Schroeder">
    <name>Schroeder</name>
    <born>1951-05-30</born>
    <qualification>brought classical music
      to the Peanuts strip</qualification>
  </character>
  <character id="Lucy">
    <name>Lucy</name>
    <born>1952-03-03</born>
    <qualification>bossy, crabby and selfish</qualification>
  </character>
</book>
</library>

```

And here is the XML 1.0 DTD :

```

<!ELEMENT library (book+) >
<!ELEMENT book (isbn, title, author+, character*) >
<!ATTLIST book id ID #REQUIRED >
<!ATTLIST book available (true|false) #REQUIRED >
<!ELEMENT isbn (#PCDATA) >
<!ELEMENT title (#PCDATA) >
<!ATTLIST title xml:lang NMTOKEN #IMPLIED >
<!ELEMENT author (name, born?, dead?) >
<!ATTLIST author id ID #REQUIRED >
<!ELEMENT name (#PCDATA) >
<!ELEMENT born (#PCDATA) >
<!ELEMENT dead (#PCDATA) >
<!ELEMENT character (name, born?, dead?, qualification) >
<!ATTLIST character id ID #REQUIRED >
<!ELEMENT qualification (#PCDATA) >

```

And now the DTD+RE version :

```

<!ENTITY % date "[[:digit:]]{4}-[:digit:]{2}-[:digit:]{2}/" >
<!ENTITY % _isbn "[[:digit:]]{10}" >
<!ELEMENT library (book+) >
<!ELEMENT book (isbn, title, author+, character*) >
<!ATTLIST book id ID_REGEX /b%_isbn;/ #REQUIRED >
<!ATTLIST book available %re.boolean; #REQUIRED >
<!ELEMENT isbn REGEX /%_isbn;/ >
<!ELEMENT title (#PCDATA) >
<!ATTLIST title xml:lang %re.language; #IMPLIED >
<!ELEMENT author (name, born?, dead?) >
<!ATTLIST author id ID #REQUIRED >
<!ELEMENT name (#PCDATA) >

```

```

<!ELEMENT born REGEX %date; >
<!ELEMENT dead REGEX %date; >
<!ELEMENT character (name, born?, dead?, qualification) >
<!ATTLIST character id ID #REQUIRED >
<!ELEMENT qualification (#PCDATA) >

```

Supposing that these two DTDs are stored respectively in the files `library.10.dtd` and `library.dre`, it is possible to define a common entry point for a standard DTD parser and a DTD+RE parser as follows :

```

<!ENTITY % xml-dtd-regex "IGNORE" >
<![%xml-dtd-regex;[
<!ENTITY % library.dtd SYSTEM "library.dre" >
]]>
<!ENTITY % library.dtd SYSTEM "library.10.dtd" >
%library.dtd;

```

Example 2

For the second example, I reuse the documentation for Perl module `PPM::XML::PPD` which describes an XML application of the Perl Package Distribution.

Here is the document instance :

```

<SOFTPKG NAME="Math-MatrixBool" VERSION="4,2,0,0">
  <TITLE>Math-MatrixBool</TITLE>
  <ABSTRACT>Easy manipulation of matrices of booleans
    (Boolean Algebra)</ABSTRACT>
  <AUTHOR>Steffen Beyer (sb@sdm.de)</AUTHOR>
  <LICENSE
    HREF="http://www.ActiveState.com/packages/Math-MatrixBool/license.html" />
  <IMPLEMENTATION>
    <OS VALUE="WinNT" />
    <OS VALUE="Win95" />
    <PROCESSOR VALUE="x86" />
    <CODEBASE
      HREF="http://www.ActiveState.com/packages/Math-MatrixBool/Math-MatrixBool-4.2-1-
        <DEPENDENCY NAME="Bit-Vector" />
    <INSTALL>
    </INSTALL>
    <UNINSTALL>
    </UNINSTALL>
  </IMPLEMENTATION>

  <IMPLEMENTATION>
    <DEPENDENCY NAME="Bit-Vector" />
    <CODEBASE
      HREF="http://www.cpan.org/CPAN/modules/by-module/Math/Math-MatrixBool-4.2.tar.
    <INSTALL>
      system("make"); ;;
      system("make test"); ;;
      system("make install"); ;;
    </INSTALL>
  </IMPLEMENTATION>
</SOFTPKG>

```

Here is the XML 1.0 DTD (this commented version in HTML was generated by the Perl module `ddd2html`) :

```

<!ELEMENT SOFTPKG (ABSTRACT | AUTHOR | IMPLEMENTATION | LICENSE | TITLE)*>

```

```

<!ATTLIST SOFTPKG NAME CDATA #REQUIRED>
<!ATTLIST SOFTPKG VERSION CDATA #IMPLIED>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT ABSTRACT (#PCDATA)>
<!ELEMENT AUTHOR (#PCDATA)>
<!ELEMENT LICENSE EMPTY>
<!ATTLIST LICENSE HREF CDATA #REQUIRED>
<!ELEMENT IMPLEMENTATION (CODEBASE | DEPENDENCY | LANGUAGE | OS |
OSVERSION | PERLCORE | PROCESSOR | INSTALL |
UNINSTALL)* >

<!ELEMENT CODEBASE EMPTY>
<!ATTLIST CODEBASE FILENAME CDATA #IMPLIED>
<!ATTLIST CODEBASE HREF CDATA #REQUIRED>
<!ELEMENT DEPENDENCY EMPTY>
<!ATTLIST DEPENDENCY VERSION CDATA #IMPLIED>
<!ATTLIST DEPENDENCY NAME CDATA #REQUIRED>
<!ELEMENT LANGUAGE EMPTY>
<!ATTLIST LANGUAGE VALUE CDATA #REQUIRED>
<!ELEMENT OS EMPTY>
<!ATTLIST OS VALUE CDATA #REQUIRED>
<!ELEMENT OSVERSION EMPTY>
<!ATTLIST OSVERSION VALUE CDATA #REQUIRED>
<!ELEMENT PERLCORE EMPTY>
<!ATTLIST PERLCORE VERSION CDATA #REQUIRED>
<!ELEMENT PROCESSOR EMPTY>
<!ATTLIST PROCESSOR VALUE CDATA #REQUIRED>
<!ELEMENT INSTALL (#PCDATA)>
<!ATTLIST INSTALL HREF CDATA #IMPLIED>
<!ATTLIST INSTALL EXEC CDATA #IMPLIED>
<!ELEMENT UNINSTALL (#PCDATA)>
<!ATTLIST UNINSTALL HREF CDATA #IMPLIED>
<!ATTLIST UNINSTALL EXEC CDATA #IMPLIED>

```

Now using DTD+RE :

```

<!ENTITY % ppd.name "[[:alnum:]]|[\- _]+/" >
<!ENTITY % ppd.version "[[:digit:]]+([[:digit:]]+){3}/" >
<!ELEMENT SOFTPKG (ABSTRACT | AUTHOR | IMPLEMENTATION | LICENSE | TITLE)*>
<!ATTLIST SOFTPKG NAME %ppd.name; #REQUIRED>
<!ATTLIST SOFTPKG VERSION %ppd.version; #IMPLIED>
<!ELEMENT TITLE REGEX %ppd.name; >
<!ELEMENT ABSTRACT (#PCDATA)>
<!ELEMENT AUTHOR (#PCDATA)>
<!ELEMENT LICENSE EMPTY>
<!ATTLIST LICENSE HREF %re.uri; #REQUIRED>
<!ELEMENT IMPLEMENTATION (CODEBASE | DEPENDENCY | LANGUAGE | OS |
OSVERSION | PERLCORE | PROCESSOR | INSTALL |
UNINSTALL)* >

<!ELEMENT CODEBASE EMPTY>
<!ATTLIST CODEBASE FILENAME CDATA #IMPLIED>
<!ATTLIST CODEBASE HREF %re.uri; #REQUIRED>
<!ELEMENT DEPENDENCY EMPTY>
<!ATTLIST DEPENDENCY VERSION %ppd.version; #IMPLIED>
<!ATTLIST DEPENDENCY NAME %ppd.name; #REQUIRED>
<!ELEMENT LANGUAGE EMPTY>
<!ATTLIST LANGUAGE VALUE CDATA #REQUIRED>
<!ELEMENT OS EMPTY>
<!ATTLIST OS VALUE CDATA #REQUIRED>
<!ELEMENT OSVERSION EMPTY>
<!ATTLIST OSVERSION VALUE %ppd.version; #REQUIRED>
<!ELEMENT PERLCORE EMPTY>
<!ATTLIST PERLCORE VERSION %ppd.version; #REQUIRED>
<!ELEMENT PROCESSOR EMPTY>
<!ATTLIST PROCESSOR VALUE CDATA #REQUIRED>
<!ELEMENT INSTALL (#PCDATA)>
<!ATTLIST INSTALL HREF %re.uri; #IMPLIED>

```

```
<!ATTLIST INSTALL EXEC CDATA #IMPLIED>
<!ELEMENT UNINSTALL (#PCDATA)>
<!ATTLIST UNINSTALL HREF %re.uri; #IMPLIED>
<!ATTLIST UNINSTALL EXEC CDATA #IMPLIED>
```

Example 3

The third example concerns [XHTML] and in particular XHTML 1.1 which is the modularised version.

This version simplifies my job since it is only necessary to rewrite a single module to begin to take advantage of DTD+RE.

The file in question is `xhtml-datatypes-1.mod` which specifies the attribute types using "CDATA", "NMOKEN" and "NMOKENS".

The following regular expressions are simplistic versions, my aim here is not to provide the best regular expressions but only to show the advantages of this technique.

Furthermore, if we want to apply the rule :

Be tolerant in what you accept and strict in what you produce

It may be worthwhile defining two versions depending on our role (producer or consumer)

```
<!-- Datatypes
      defines containers for the following datatypes, many of
      these imported from other specifications and standards.
-->

<!-- Length defined for cellpadding/cellspacing -->
<!-- nn for pixels or nn% for percentage length -->
<!ENTITY % Length.datatype "[[:digit:]]+%" >
<!-- space-separated list of link types -->
<!ENTITY % LinkTypes.re
      "(next/prev/head/toc/parent/child/index/glossary)" >
<!ENTITY % LinkTypes.datatype
      "%LinkTypes.re;([[:space:]]+%LinkTypes.re;)*" >
<!-- single or comma-separated list of media descriptors -->
<!ENTITY % MediaDesc.re
      "(screen/tty/tv/projection/handheld/print/braille/aural/all)" >
<!ENTITY % MediaDesc.datatype
      "%MediaDesc.re;(,%MediaDesc.re;)*" >
<!-- pixel, percentage, or relative -->
<!ENTITY % MultiLength.datatype "[[:digit:]]+%" >
<!-- one or more digits (NUMBER) -->
<!ENTITY % Length.datatype "[[:digit:]]+" >
<!-- integer representing length in pixels -->
<!ENTITY % Pixels.datatype "\\+?[[:digit:]]+" >
<!-- script expression -->
```

```

<!ENTITY % Script.datatype "CDATA" >

<!-- textual content -->
<!ENTITY % Text.datatype "CDATA" >

<!-- Imported Datatypes ..... -->

<!-- a single character from [ISO10646] -->
<!ENTITY % Character.datatype "/./" >

<!-- a character encoding, as per [RFC2045] -->
<!ENTITY % Charset.re "([\.\-_:][[:alnum:]]+" >
<!ENTITY % Charset.datatype "%Charset.re;" >

<!-- a space separated list of character encodings, as per [RFC2045] -->
<!ENTITY % Charsets.datatype "%Charset.re;(,%Charset.re;)*/" >

<!-- Color specification using color name or sRGB (#RRGGBB) values -->
<!ENTITY % Color.datatype "%alpha:][:alnum:]*#[[:xdigit:]]{6}"/" >

<!-- media type, as per [RFC2045] -->
<!ENTITY % ContentType.re.type "([[:alnum:]]|-)+" >
<!ENTITY % ContentType.re.subtype "([[:alnum:]]|-)+" >
<!ENTITY % ContentType.re.parameter "[^=]+=[^;]+" >
<!ENTITY % ContentType.re "%ContentType.re.type;%ContentType.re.subtype;(;%ContentType.re.parameter;)*" >
<!ENTITY % ContentType.datatype "%ContentType.re;" >

<!-- comma-separated list of media types, as per [RFC2045] -->
<!ENTITY % ContentTypes.datatype "%ContentType.re;(,%ContentType.re;)*/" >

<!-- date and time information. ISO date format -->
<!ENTITY % Datetime.datatype "%re._datetime;" >

<!-- formal public identifier, as per [ISO8879] -->
<!ENTITY % FPI.datatype "CDATA" >

<!-- a language code, as per [RFC3066] -->
<!ENTITY % LanguageCode.datatype "%re._language;" >

<!-- a Uniform Resource Identifier, see [URI] -->
<!ENTITY % URI.datatype "%re._uri;" >

<!-- a space-separated list of Uniform Resource Identifiers, see [URI] -->
<!ENTITY % URIs.datatype "%re._uri;([[:space:]]+%re._uri;)*/" >

```

This is certainly not the best possible use of DTD+RE, but it is a way to begin migrating the large volume of existing [XHTML].

Conclusion

DTD+RE is based on a simple idea. Which should enable :

- a simplified implementation based on a validating DTD parser, since regular expression engines are available in the languages used to write parsers (A Java prototype has been built based on the

SAX2 parser Saxon-AEfred)

- a reduced learning curve, since there are no new technologies requiring extensive documentation

Furthermore, the regular expression technique is well known and proven in the string processing domain.

DTD modularisation techniques allow reuse of macro components to be achieved, as for [XHTMLMOD]. The use of parameter entities for regular expression fragments will allow reuse of micro components. This reuse begins with the list of predefined parameter entities which will need to be refined and enriched.

The major advantage of this approach is its backwards compatibility, a XML 1.0 DTD is a DTD+RE. This allows application migration by progressively enriching the DTDs with regular expressions.

References

- [RELAX NG] REgular LAnguage description for XML - Next Generation <http://www.oasis-open.org/committees/relax-ng/>
- [POSIX] Portable Operating System Interface (POSIX), IEEE Std 1003.1-2001 http://www.unix.org/version3/ieee_std.html
- [SGML] Standard Generalized Markup Language, ISO 8879:1986
- [WXS] XML Schema Part 2: Datatypes, W3C Recommendation. <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502>
- [XHTML] Module-based XHTML, W3C Recommendation. <http://www.w3.org/TR/2001/REC-xhtml11-20010531>
- [XHTMLMOD] Modularization of XHTML, W3C Recommendation. <http://www.w3.org/TR/2001/REC-xhtml-modularization-20010410>
- [XML] Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation. <http://www.w3.org/TR/2000/REC-xml-20001006>