XML Data Management

3. Document Type Definitions (DTDs)

Werner Nutt

based on slides by Sara Cohen, Jerusalem

Document Type Definitions

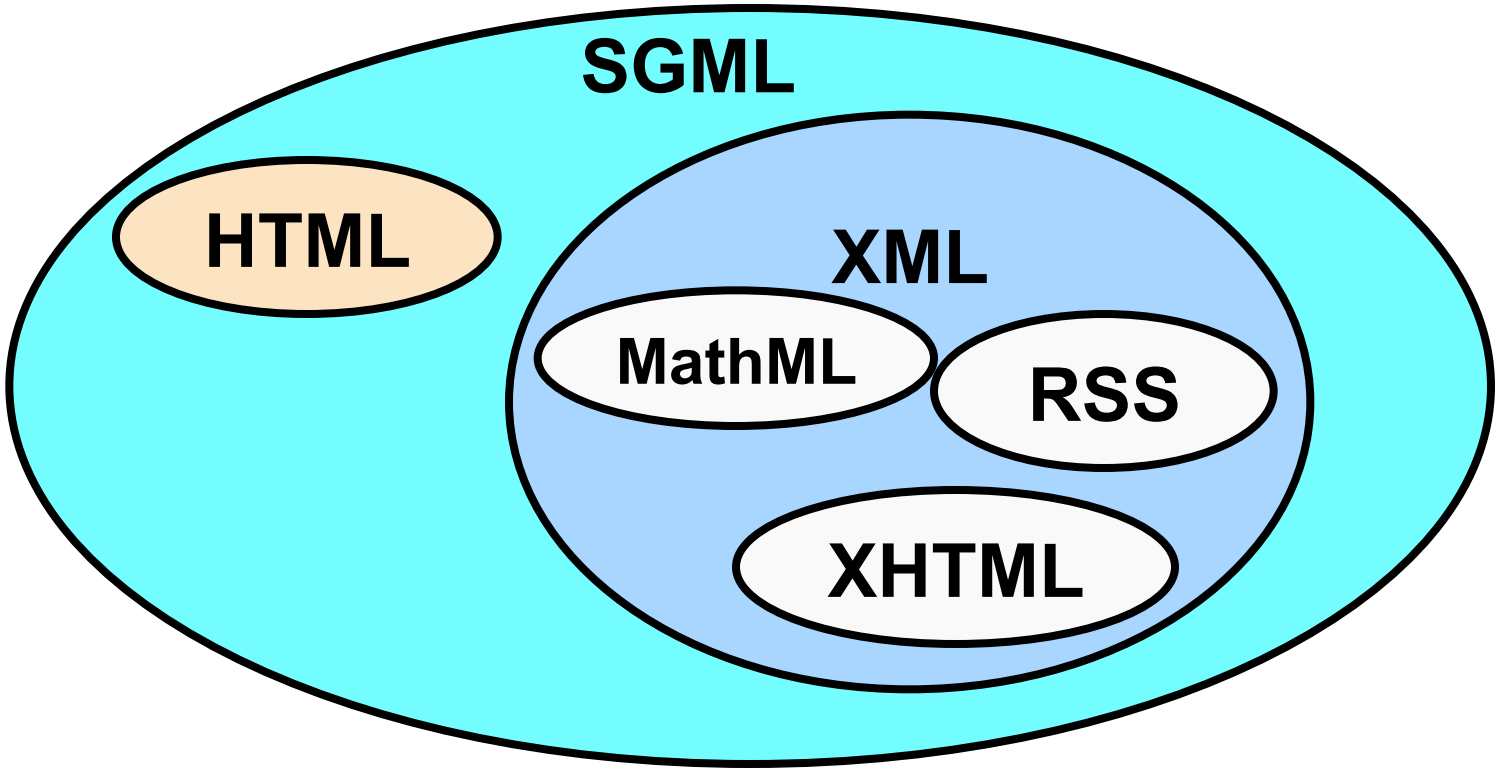
- Document Type Definitions (DTDs)
  impose structure on an XML document
- Using DTDs, we can specify
  what a "valid" document should contain
- DTD specifications require more than being well-formed,
  e.g., what elements are legal, what nesting is allowed
- DTDs have limited expressive power,
  e.g., one cannot specify types

What is This Good for?

- DTDs can be used to define special languages of XML, i.e., restricted XML for special needs
- Examples:
  - MathML (mathematical markup)
  - SVG (scalable vector graphics)
  - XHTML (well-formed version of HTML)
  - RSS ("Really Simple Syndication", news feeds)
- Standards can be defined using DTDs, for data exchange and special applications

now, often replaced by XML Schema

Alphabet Soup



Example: MathML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE math PUBLIC "-//W3C//DTD MathML 2.0//EN"
  "http://www.w3.org/Math/DTD/mathml2/mathml2.dtd">
<math>
  <mrow>
    <msup>
      <mi>x</mi>
      <mn>2</mn>
    </msup>
    <mo>&InvisibleTimes;</mo>
    <mi>y</mi>
  </mrow>
</math>
```

Example: SVG

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
     "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="250px" height="250px"
          xmlns="http://www.w3.org/2000/svg">
  <g fill="red">
    <text font-size="32" x="45" y="60">
         Hello, World!
    </text>
  </g>
  <g fill="blue">
    <text font-size="32"  x="50" y="90">
         Hello, World!
    </text>
    <text font-size="32"  x="58" y="98">
         Hello, World!
    </text>
  </g>
</svg>
```

Address Book DTD

- Suppose we want to create a DTD
  that describes legal address book entries
- This DTD will be used to exchange
  address book information between programs
- How should it be written?
- What is a legal address?

Example: An Address Book Entry

```
<person>
   <name>Homer Simpson</name>
   <greet>Dr. H. Simpson</greet>
   <addr>1234 Springwater Road</addr>
   <addr>Springfield USA, 98765</addr>
   <tel>(321) 786 2543</tel>
   <fax>(321) 786 2544</fax>
   <tel>(321) 786 2544</tel>
   <email>homer@math.springfield.edu</email>
</person>
```

exactly one name

at most one greeting

as many address lines as needed

mixed telephones and faxes

at least one email

Specifying the Structure

How do we specify exactly what must appear in a person element?

- A DTD specifies for each element the permitted content
- The permitted content is specified by a

  regular expression

- Our plan:
  - first, regular expression defining the content of person
  - then, general syntax

What's in a `person` Element?

Exactly one name,

followed by at most one greeting,

followed by an arbitrary number of address lines,

followed by a mix of telephone and fax numbers,

followed by at least one email.

regular expression

Formally:

```
name, greet?, addr*, (tel | fax)*, email+
```

What's in a `person` Element? (cntd)

```
name, greet?, addr*, (tel | fax)*, email+
```

`name` = there must be a name element

`greet?` = there is an optional greet element
(i.e., 0 or 1 greet elements)

`name, greet?` = the name element is followed
by an optional greet element

`addr*` = there are 0 or more address elements

What's in a `person` Element? (cntd)

```
name, greet?, addr*, (tel | fax)*, email+
```

`tel | fax` = there is a tel *or* a fax element

`(tel | fax)*` = there are 0 or more repeats of tel or fax

`email+` = there are 1 or more email elements

What's in a `person` Element? (cntd)

```
name, greet?, addr*, (tel | fax)*, email+
```

Does this expression differ from:

```
name, greet?, addr*, tel*, fax*, email+
name, greet?, addr*, (fax|tel)*, email+
name, greet?, addr*, (fax|tel)*, email, email*
name, greet?, addr*, (fax|tel)*, email*, email
```

Element Content Descriptions

a	element a
e1?	0 or 1 occurrences of expression e1
e1*	0 or more occurrences of expression e1
e1+	1 or more occurrences of expression e1
e1,e2	expression e2 after expression e2
e1|e2	either expression e1 or expression e2
(e)	grouping
#PCDATA	parsed character data *(i.e., after parsing)*
EMPTY	no content
ANY	any content
(#PCDATA | a_1 | … | a_n)*	mixed content

addressbook as Internal DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE addressbook [
  <!ELEMENT addressbook (person*)>
  <!ELEMENT person  (name, greet?, address*,
  (fax | tel)*, email+)>
  <!ELEMENT name    (#PCDATA)>
  <!ELEMENT greet   (#PCDATA)>
  <!ELEMENT address (#PCDATA)>
  <!ELEMENT tel     (#PCDATA)>
  <!ELEMENT fax     (#PCDATA)>
  <!ELEMENT email   (#PCDATA)>
]>
```

Exercise

Requirements

- A country must have a name as the first node.
- A country must have a capital city as the following node.
- A country may have a king.
- A country may have a queen.

What about the following?

```
<!ELEMENT country (name,capital?,king*,queen)>
```

Exercise

Requirements for binary trees

- A node has two children,
  which can be nodes or leaves

- A leaf contains text.

Exercise

Requirements:

- A country must have
  - a president or
  - a king or
  - a king and a queen or
  - a queen.

Let’s Validate This

E.g., as an internal DTD!

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE country[
  <!ELEMENT country
    (president | king | (king,queen) | queen)>
  <!ELEMENT president   (#PCDATA)>
  <!ELEMENT king        (#PCDATA)>
  <!ELEMENT queen       (#PCDATA)>
]>
<country><king>Luis</king></country>
```

The Error

Validation Output: 1 Error

Line 3, Column 66: **content model is ambiguous: when no tokens have been matched, both the 1st and 2nd occurrences of "king" are possible**

```
<!ELEMENT country     (president | king | (king,queen) | queen) >
```

What's the problem?

How can we fix it?

Deterministic DTDs

SGML requires that a DTD is deterministic, that is,

when parsing a document, a parser only needs to look at the next element to know at which point it is in the regular expression

1-step lookahead

Is this DTDs deterministic?

```
<!ELEMENT a  ((b,c) | (b,d))>
```

Try `<a><b/><d/></a>` !

Can we fix this one?

Deterministic DTDs

E Deterministic Content Models (Non-Normative)

As noted in **3.2.1 Element Content**, it is required that content models in element type declarations be deterministic. This requirement is for compatibility with SGML (which calls deterministic content models "unambiguous"); XML processors built using SGML systems may flag non-deterministic content models as errors.

For example, the content model ((b, c) | (b, d)) is non-deterministic, because given an initial b the XML processor cannot know which b in the model is being matched without looking ahead to see which element follows the b. In this case, the two references to b can be collapsed into a single reference, making the model read (b, (c | d)). An initial b now clearly matches only a single name in the content model. The processor doesn't need to look ahead to see what follows; either c or d would be accepted.

…

From: Extensible Markup Language (XML) 1.0 (Fifth Edition)
W3C Recommendation 26 November 2008

Research Questions

What are research questions to ask about
non-deterministic and deterministic DTDs?

1. Is there an algorithm to check
   whether a DTD is (non-)deterministic?
2. Is there an algorithm running in polynomial time?
   (Or is this problem NP-hard?)
3. What is the exact runtime of the best algorithm?
4. Is there for every (nondeterministic) DTD
   an equivalent deterministic DTD?

Answers by Anne Brüggemann-Klein (1993):

1) yes, 2) yes, 3) quadratic for DTDs, linear for expressions,

4) yes, but it may be exponential in the size of the input

Exercise: Payments

Requirements:

- Customers at the till may pay with a combination of credit cards and cash.
- If cards and cash are both used the cards must come first.
- There may be more than one card.
- There must be no more than one cash element.
- At least one method of payment must be used.

Task:

- Construct a deterministic DTD with the elements `card` and `cash`

Attributes

How can we define the possible attributes of elements in XML documents?

General Syntax:

```
<!ATTLIST element-name
  attribute-name1 type1 default-value1
  attribute-name2 type2 default-value2
                  …
  attribute-namen typen default-valuen>
```

Example:

```
<!ATTLIST height  dim  CDATA  "cm">
```

Attributes (cntd)

```
<!ATTLIST element-name
  attribute-name1 type1 default-value1
                  …                     >
```

type is one of the following:

CDATA	character data *(i.e., the string as it is)*
(en1 | en2 | …)	value must be one from the given list
ID	value is a unique id
IDREF	value is the id of another element
IDREFS	value is a list of other ids

… there are more possibilities (e.g., ENTITY or NMTOKEN), which we don't discuss)

Attributes (cntd)

```
<!ATTLIST element-name
  attribute-name1 type1 default-value1
                  …                     >
```

default-value is one of the following:

value	default value of the attribute
#REQUIRED	attribute must always be included in the element
#IMPLIED	attribute need not be included
#FIXED *value*	attribute value is fixed

Example: Attributes

```
<!ELEMENT height (#PCDATA)>

<!ATTLIST height
        dimension   (cm|in)    #REQUIRED
        accuracy     CDATA     #IMPLIED
        resizable    CDATA     #FIXED "yes"
>
```

Need not appear in the doc,
will be automatically added by the XML processor

Typical usage:

```
xmlns CDATA #FIXED "http://spam.com"
```

Specifying ID and IDREF Attributes

```
<!DOCTYPE family [
  <!ELEMENT family   (person)*>
  <!ELEMENT person   (name)>
  <!ELEMENT name     (#PCDATA)>
  <!ATTLIST person
       id         ID        #REQUIRED
       mother     IDREF     #IMPLIED
       father     IDREF     #IMPLIED
       children   IDREFS    #IMPLIED>
]>
```

Specifying ID and IDREF Attributes (cntd)

Attributes **`mother`** and **`father`**
are references to IDs of other elements

However,

- those elements are not necessarily **`person`** elements
- the **`mother`** attribute is not necessarily
a reference to a female person

References to IDs have no type!

ID, IDREF, and IDREFS in a Document

```
<family>
  <person  id="lisa"  mother="marge" father="homer">
      <name> Lisa Simpson </name>
  </person>
  <person  id="bart"  mother="marge" father="homer">
      <name> Bart Simpson </name>
   </person>
  <person id="marge" children="bart lisa">
      <name> Marge Simpson </name>
  </person>
  <person id="homer" children="bart lisa">
      <name> Homer Simpson </name>
  </person>
</family>
```

Consistency of ID and IDREF Attribute Values

- If attributes are declared as ID

  their associated values must all be distinct

  (no confusion)

  That is, no two ID attributes can have the same value

- If an attribute is declared as IDREF

  the associated value must exist as the value of some ID attribute *(no dangling "pointers")*

- Similarly for all the values of an IDREFS attribute

Which parallels do you see to relational databases?

Is this Legal?

```
<family>
  <person id="superman" mother="lara" father="jor-el">
      <name> Clark Kent </name>
  </person>

  <person  id="kara"  children="laura" >
      <name> Linda Lee </name>
   </person>
</family>
```

Relational Keys vs. IDs in DTDs

- Relational keys may be multi-valued,
  while IDs are always single-valued

```
enroll (sid: string, cid: string, grade:string)
```

- A db relation may have multiple keys,
  while an element can have at most one ID
- A foreign key points always to tuples of the same relation,
  while an IDREF can point to arbitrary elements
- Every db relation has a schema (which defines keys),
  while XML data may come w/o a DTD (XML schema)

Due to Wenfei Fan, U Edinburgh

Adding a DTD to a Document

- A DTD can be *internal*
  - the DTD is part of the document file
- … or *external*
  - the DTD and the document are on separate files
- An external DTD may reside
  - in the local file system (where the document is)
  - in a remote file system (reachable using a URL)

Connecting a Document with its DTD

- Internal DTD:

```
<?xml version="1.0"?>
<!DOCTYPE db [<!ELEMENT ...> … ]>
<db> ... </db>
```

- DTD from the local file system:

```
<!DOCTYPE db SYSTEM "schema.dtd">
```

- DTD from a remote file system:

```
<!DOCTYPE db SYSTEM
        "http://www.schemaauthority.com/schema.dtd">
```

Connecting a Document with its DTD

Combination of external and internal DTD

```
<?xml version="1.0"?>
<!DOCTYPE db SYSTEM "schema.dtd"
 [
   <!ATTLIST db vendor CDATA #REQUIRED >
    …
 ]
>
<db> ... </db>
```

internal subset

DTD Entities

Entities are XML macros. They come in four kinds:

- Character entities: stand for arbitrary Unicode characters, like: <, ; , & , ©, …
- Named (internal) entities: macros in the document, can stand for any well-formed XML, mostly used for text
- External entities: like named entities, but refer to a file with with well-formed XML
- Parameter entities: stand for fragments of a DTD

… and are referenced in a DTD

Character Entities

Macros expanded when the document is processed.

Example: Special characters from XHTML1.0 DTD

```
<!ENTITY mdash   "&#8212;"> <!-- em dash, U+2014 ISOpub -->
<!ENTITY lsquo   "&#8216;"> <!-- left single quotation mark,
                                 U+2018 ISOnum -->
<!ENTITY copy    "&#169;">  <!-- copyright sign,
                                 U+00A9 ISOnum -->
```

Can be specified in decimal (above) and in hexadecimal, e.g.,

```
<!ENTITY mdash   "&#x2014;">
```

(`x` stands for hexadecimal)

Named Entities

Declared in the DTD (or its local fragment, the "internal subset")

- Entities can reference other entities
- … but must not form cycles (which the parser would detect)

Example:

```
<!ENTITY d "Donald">
<!ENTITY dd "&d; Duck">
```

Using `dd` in a document expands to

Donald Duck

External Entities

Represent the content of an external file.
Useful when breaking a document down into parts.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE book SYSTEM book.dtd
 [
  <!ENTITY chap1 SYSTEM "chapter-1.xml">
  <!ENTITY chap2 SYSTEM "chapter-2.xml">
  <!ENTITY chap3 SYSTEM "chapter-3.xml">
 ]>
<!-- Pull in the chapters -->
<book>
   &chap1;&chap2;&chap3;
</book>
```

internal subset

location of the file

Parameter Entities

- Can only be used in DTDs and the internal subset
- Indicated by percent (%) symbol instead of ampersand (&)
- Can be named or external entities

→ Modularization of DTDs

Pattern:

```
<!ENTITY % name "Text to be inserted">
```

Parameter Entities in the XHTML 1 DTD

```
<!--========== Generic Attributes ==========-->
<!-- core attributes common to most elements -->
<!ENTITY % coreattrs
 "id          ID              #IMPLIED
  class       CDATA           #IMPLIED
  style       %StyleSheet;    #IMPLIED
  title       %Text;          #IMPLIED"
  >

<!-- internationalization attributes -->
<!ENTITY % i18n
 "lang        %LanguageCode;  #IMPLIED
  xml:lang    %LanguageCode;  #IMPLIED
  dir         (ltr|rtl)       #IMPLIED"
  >
…
<!ENTITY % attrs "%coreattrs; %i18n; %events;">
```

Parameter Entities in the XHTML 1 DTD

```
<!--============= Document Body =============-->
<!ELEMENT body %Block;>
<!ATTLIST body
  %attrs;
  onload          %Script;   #IMPLIED
  onunload        %Script;   #IMPLIED
  >

<!ENTITY % block
     "p | %heading; | div | %lists; | %blocktext; |
                          fieldset | table">

<!ENTITY % Block "(%block; | form | %misc;)*">
```

Valid Documents

A document with a DTD is *valid* if it conforms to the DTD,

that is,

- the document conforms

  to the regular-expression grammar,

- types of attributes are correct,
- constraints on references are satisfied.

DTDs Support Document Interpretation

```
<?xml version="1.0" encoding="UTF-8"?>
<a>
  <b/>
</a>
```

How many children of the node <a>
will a DOM parser find?

DTDs Support Document Interpretation

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE a [
<!ELEMENT a (b)>
<!ELEMENT b EMPTY>
 ]>
<a>
  <b/>
</a>
```

How many children of the node <a>
will a DOM parser find now?

Not Every DTD Makes Sense

```
<DOCTYPE genealogy [
  <!ELEMENT genealogy (person*)>
  <!ELEMENT person (
      name,
      dateOfBirth,
      person,          <!-- mother -->
      person   )>      <!-- father -->
  ...
]>
```

Is there a problem with this?

Not Every DTD Makes Sense (cntd)

```
<DOCTYPE genealogy [
  <!ELEMENT genealogy (person*)>
  <!ELEMENT person (
      name,
      dateOfBirth,
      person?,          <!-- mother -->
      person?   )>      <!-- father -->
  ...
]>
```

Is this now okay?

Weaknesses of DTDs

- DTDs are rather weak specifications by DB & programming-language standards
  - Only one base type: PCDATA
  - No useful “abstractions”, e.g., sets
  - IDs and IDREFs are untyped
  - No constraints, e.g., child is inverse of parent
  - Tag definitions are global
- Some extensions impose a schema or types on an XML document, e.g., XML Schema

Weaknesses of DTDs (cntd)

Questions:

- How would you say that element `a` has exactly the children `b`, `c`, `d` in any order?

- In general, can validity of documents with respect to such definitions be checked efficiently?