Everything you never wanted to know about DTDs but were afraid to ask

Arnaud Sahuguet
Penn Database Research Group
University of Pennsylvania

!! Self Promotion !!

- XMill* XML compression
- XDuce XML Programming Language
- Silk Route* XML views on top of relational data
- XML-QL* XML query language
- W4F HTML to XML translation
What this talk is all about

There are two kinds of people. Those who look at the world as it is and wonder “why?”. And those who look at the world as it could be and wonder “why not?”.

(R. Kennedy)

For a change, it would be nice -- when one speaks about XML -- for him/her to look at the world as it is.

This on-going work simply looks at how people are expressing the structure of XML documents using DTDs.

The XML map

![XML Map Diagram]

- Industry
- Research
- XML Users
Everything you ever wanted to know about ....

$ how to create your XML start-up
$ raise funds
$ go public
$ and make money

1. Pick a cool name

www.mind-your-own-dtd.com

and a cool logo

www.mind-your-own-dtd.com
2. Build the right team

(XML knowledge is a +)

- Arnaud Sahuguet
  Founder, CEO, XML Inquisitor

- Byron Choi
  Chief Technology Officer

- Peter Buneman, PhD
  Chief Scientist, XML Evangelist

- Sanjeev Khanna, PhD
  VP Algorithms

3. Next steps (left as an exercise)

- Raise funds
- Spend it all in advertising
- Raise funds again (second round)
- Go public
- Sell your stock
- Quit and repeat from step 1

single atomic transaction
More seriously: outline

• What are DTDs
• Why bother
• The survey
• What’s wrong with DTDs
• Replacements
• Future Work

SGML DTDs [ISO 8879]

A document type definition specifies:
• the generic identifiers (GIs) of elements that are permissible in a document of this type
• for each GI, the possible attributes, their range of values and defaults
• for each GI, the structure of its contents, including:
  – which element can occur and in what order
  – whether text characters can occur
  – whether non character data can occur

Bottom line: the purpose of a DTD is to permit to determine whether the mark-up for an individual document is correct and also to supply markup that is missing, because it can be inferred unambiguously from other mark-up present.
XML DTDs

Entity Parameters:

```xml
<!ENTITY % url.att
  "href" CDATA #REQUIRED
  "visited" CDATA #IMPLIED
  "modified" CDATA #IMPLIED
  "%local.url.att;">
```

Entity parameters are referenced using %entityName;

Entity parameters are like macros that can be used only in the DTD itself. They are not available to XML documents.

XML DTDs

Entity References:

```xml
<!ENTITY amp CDATA "&amp;">
<!ENTITY lt CDATA "&lt;">
<!ENTITY gt CDATA "&gt;">
<!ENTITY diff CDATA "&lt;&gt;">
```

Entity references are referenced using &entityName;

Entity references are like constants that can be used in the XML document.
**XML DTDs**

Element declarations:

```xml
<!ELEMENT title (#PCDATA)>  
<!ELEMENT metadata EMPTY>  
<!ELEMENT bookmark (title?,info,desc?,(bookmark|folder))*>
```

Attribute declarations:

```xml
<!ATTLIST alias ref IDREF #REQUIRED>  
<!ATTLIST folder folded (yes|no) #FIXED "yes">  
```

Other stuff I will deliberately and happily ignore:

- Processing instruction: `<? ....?>`
- Notations

---

**XML DTDs (cont’d)**

- Elements are defined by a content-model
  - sequence (".") or alternation ("|") of sub-elements
  - "?" (optional), "*" (zero-or-more), "+" (one-or-more)
  - EMPTY (no sub-elements), ANY (any), PCDATA
  - mixed-content (alternation of PCDATA and something else)
- Attributes can be CDATA (text), NMTOKEN (tokens) or enumeration.
- Attributes can be optional (#IMPLIED) or mandatory(#REQUIRED).
- Attributes can be constant (#FIXED) and can be assigned a default value (#DEFAULT)
A DTD for bookmarks (XBel)

```xml
<!DOCTYPE jg PUBLIC "-//JG" >
<!ENTITY folder SYSTEM "folder.jpg" %DATA jg >
<!ENTITY bookmark SYSTEM "bookmark.jpg" %DATA jg >
<!ENTITY % local.node.att "">
<!ENTITY % local.url.att "">
<!ENTITY % node.att "">
<!ENTITY % url.att "">
<!ENTITY % nodes.mix "">
<!ENTITY xbel (title?, info?, desc?, (%nodes.mix*))>
<!ATTLIST xbel %node.att;
%node.url.att>
<!ATTLIST %nodes.mix>
"bookmark|folder|file|separator">
```

```xml
<ELEMENT title (PCDATA)>
<ELEMENT info (PCDATA)>
<ELEMENT metadata EMPTY>
<ATTLIST metadata
owner CDATA #REQUIRED>
<ELEMENT folder (title?, info?, desc?, (%nodes.mix*))>
<ATTLIST folder
%node.att;
folded (yes|no) #FIXED 'yes' >
<ELEMENT bookmark (title?, info?, desc?)>
<ATTLIST bookmark
%node.att;
%url.att>
<ELEMENT desc (PCDATA)>
<ELEMENT separator EMPTY>
<ELEMENT alias EMPTY>
<ATTLIST alias
ref IDREF #REQUIRED>
```

Insightful Analogy

<table>
<thead>
<tr>
<th>XML</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>validation</td>
<td>type-checking</td>
</tr>
<tr>
<td>entity reference</td>
<td>constants</td>
</tr>
<tr>
<td>entity parameter</td>
<td>macros</td>
</tr>
<tr>
<td>ANY</td>
<td>void*</td>
</tr>
<tr>
<td>IDREF</td>
<td>void*</td>
</tr>
<tr>
<td>DTD</td>
<td>header file</td>
</tr>
<tr>
<td>conditional section</td>
<td>#ifdef</td>
</tr>
<tr>
<td>key entities</td>
<td>standard library</td>
</tr>
<tr>
<td>namespace</td>
<td>namespace</td>
</tr>
</tbody>
</table>

It is interesting to remark that features like inheritance, type inference, polymorphism or modules are missing.
Why bother about DTDs?

- XML exists without DTDs (well-formed/valid)
- DB people do not pay attention to them anyway

Projects (as far as I know)

<table>
<thead>
<tr>
<th>STORED</th>
<th>Ignored. Data-mining instead.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wisc. (VLDB99)</td>
<td>Used after simplification.</td>
</tr>
<tr>
<td>XML-QL</td>
<td>Not used for optimization.</td>
</tr>
<tr>
<td>XMIL</td>
<td>Not used.</td>
</tr>
<tr>
<td>CWI’s paper</td>
<td>Not used.</td>
</tr>
<tr>
<td>Xyleme</td>
<td>for document clustering + query formulation.</td>
</tr>
<tr>
<td>Niagara</td>
<td>for document clustering + query formulation.</td>
</tr>
<tr>
<td>Excelon</td>
<td>“DTD agnostic” (off-the-record)</td>
</tr>
<tr>
<td>Lore</td>
<td>Dataguides.</td>
</tr>
<tr>
<td>W4F</td>
<td>small DTD subset supported</td>
</tr>
</tbody>
</table>

Information from DTD IS useful.

- Validation
  - e.g. runtime guarantees
- Query optimization
  - e.g. resolution of wildcards in regular path expressions (role similar to dataguides)
- Storage/Compression
  - e.g. enum values for attributes, #elements, etc.
- Meta-information
  - documentation, retrieval, clustering
- DTD as a type system
  - language binding
The survey: methodology

- Over 60 DTDs reviewed.
- + Penn Web site where you can submit your own.

What we have been looking at

- DTD well-formedness
- Features being used
- Structure (size): depth, #elements, #attributes
- Structure (redundancy)
- Content-model (type)
- Content-model (complexity)
- Recursivity
- Number of roots
- and many other features

Computing features is (relatively) easy.
Finding out what to compute and finding a meaning to them is the hard part.
DTD “well-formedness”

• Result
  – most DTDs are incorrect: typos, missing declarations, invalid declarations, duplicate declarations

• Possible explanations
  – DTDs seem to be used more for documentation than for validation or anything else.
  – No good tools available to help in DTD authoring. Tools are now appearing (XML-Spy, XML-Authority).

• Issue
  – DTDs supposedly used to validate documents

  *Quis custodiet custodes ipsos?*

Well-formededness

- □ Typos
- □ Syntax error
- □ Lack of knowledge on DTD
- □ Variant of DTD
- □ External entities needed
- □ Cannot be corrected
DTD unused features

- Results
  -notations and fancy attribute types are almost never used
  -ID and IDREF are rarely used (more later on)

  36 out of 60 are not using IDREFs

  -“+” is not often used

- Possible explanations:
  -DTD specification is CRYPTIC
  -DTD features are used as needed
  -un-typed references are not useful to people

- Issue
  -why not get rid of useless features?

DTD sizes vary a lot

- Result
  -a huge spectrum

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>max</th>
<th>avg</th>
<th>med</th>
</tr>
</thead>
<tbody>
<tr>
<td>#elements</td>
<td>4</td>
<td>590</td>
<td>74</td>
<td>36</td>
</tr>
<tr>
<td>#attributes</td>
<td>0</td>
<td>5,700</td>
<td>390</td>
<td>82</td>
</tr>
<tr>
<td>#entities</td>
<td>0</td>
<td>190</td>
<td>15</td>
<td>2</td>
</tr>
</tbody>
</table>

- Possible explanations
  -people are using DTDs to model almost everything

- Issues
  -the size of the DTD potentially influences the size of the XML document
  -a “big” DTD makes validation more expensive
  -a “big” DTD implies a big corresponding type
**DTDs are redundant**

- What we mean by redundancy
  - CM redundant: same content-model
  - Redundant: CM redundant + same attributes

- Result
  - Many DTDs are highly redundant
  - Different elements can have the exact same content-model

- Possible explanations
  - Since DTDs do not offer any mechanism for inheritance, people are forced to re-declare things many times.

<table>
<thead>
<tr>
<th></th>
<th>CM Redundancy</th>
<th>Redundancy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Max</td>
<td>95%</td>
<td>95%</td>
</tr>
<tr>
<td>Avg</td>
<td>55%</td>
<td>34%</td>
</tr>
<tr>
<td>Med</td>
<td>57%</td>
<td>37%</td>
</tr>
</tbody>
</table>

**Tuple encoding**

- Result
  - People are misusing CM to encode DTDs
  - Tuple(a, b, c)
    - In SGML: (a & b & c)
    - In XML: (a,b,c) \(\mid\) (a,c,b) \(\mid\) (b,c,a) \(\mid\) (b,a,c) \(\mid\) (c,b,a) \(\mid\) (c,a,b)
    - In practice: \((a|b|c)^*\)

- Possible explanations
  - DTDs do not offer a tuple construct and people had to find a workaround.

- Issue
  - \((a|b|c)^*\) is a misleading representation of tuples: the type is much larger than needed.
Entities to model inheritance

- **Result**
  - inheritance is captured by entity parameters, but this is purely syntactic

- **Possible explanations**
  - this is the only way to avoid repeating the same CM
  - this is the only way to make CM extensible

- **Issues**
  - this notion of inheritance is misleading, specially when tuples are encoded using \((a_1|a_n)^*\)
What's wrong with DTDs?

- Too much document oriented (interface with text processing)
- Too simple and too complicated at the same time
- No notion of record
- IDREFs are not typed
- No notion of inheritance/sub-typing
- No notion of constraints
- No obvious way to offer versioning, extension, evolution
- Content-model ambiguous
- Too many ways to represent the same thing

Replacement Candidates

- Grammar-based approaches
  - XML-Data/BizTalk, SOX, DCD
  - XML-Schemas
  - DSD
- Constraint-based approaches
  - Schematron
  - Relax
- Type-based approaches
  - XDuce
  - Haskell
- Other
  - UML
XML-Schemas

- The name itself is misleading
- Official goal: to replace DTDs and XML-Data
- Supported by all the major players
- XML syntax
- More expressive than DTDs
  - atomic types
  - better semantic for the content-model (tuples are back!)
  - mechanisms to extend and restrict schemas
  - constraints (limited, based on XPath)
- XML-Schema = data types + structures + constraints*

Schematron

- Idea: encoding structure using tree constraints
- Not based on grammars but on tree patterns
- Semantics
  - find a context node in the document
  - check for constraints (i.e. XPath expressions)
- Features
  - in the spirit of XSL (patterns, rules)
  - based on XPath
- Benefits
  - a "schema" specification can have more or less refined
  - supports variations of the schema (versions, etc.)
You can for instance “type” ID/IDREF relationships.
**XML Bindings**

- There is only a “gentlemen’s agreement” between the application and its XML environment.

- Why do we need to go beyond that?
  - performance
  - static guarantees

- How do we create a tight contract between the application and its XML environment?

**Conclusions**

- XML is just syntax.
- For serious applications, XML documents need a specification (a “soul”).
- DTDs are simply not adequate for the job.
- Even worse. People have been hacking DTDs, which means that they often provide misleading information.
  - DTD inference is one way to solve the problem.
Conclusion (cont’d)

- By looking at these hacks, we can get a pretty clear idea of what is needed.
- The good news: most issues are addressed by replacement candidates.
- Bad news: important issues still not addressed:
  - versioning
  - constraints
  - bindings
- Future work:
  - relation between DTDs and documents they describe
  - DTD categorization using NLP
  - metrics for DTD (what is a good DTD)

Questions

Stay tuned:

http://xml.cis.upenn.edu/DTDi
Use of structural information
(from XML Schema Requirements)

- Publishing and syndication
- Electronic commerce transaction processing
- Supervisory control and data acquisition
- Traditional document authoring/editing governed by schema constraints
- Use schema to help query formulation and optimization
- Open and uniform transfer of data between applications, including databases
- Metadata Interchange

- Language binding