eXtensible Markup Language

CS 368 — Web Programming — Ben Liblit

Meet our Cast of Characters

- XML: eXtensible Markup Language
  - Can add lightweight semantic info to plain text
  - Can describe arbitrarily complex structured data
  - Just data; doesn't really *do* anything by itself
- DTD: Document Type Definition
  - Structure of some XML format you plan to reuse
  - One DTD ↔ many XML documents
  - Just like HTML syntax ↔ many HTML pages

2

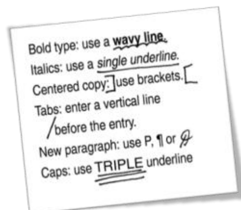
Meet our Cast of Characters

- XPath
  - Compact syntax for grabbing fragments of XML data
- XSLT: eXtensible Stylesheet Language Transformations
  - Programming language for transforming XML
    - It does stuff!
    - Arbitrary calculations, logic, conditional branches, etc.
  - Uses XPath extensively

3

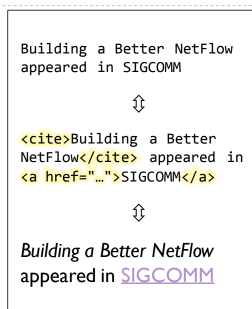
Markup Languages

- Give structure and meaning to plain text
- Lightweight overlay
  - Erase and you're back to plain text
- Markup "vocabulary" agreed-upon by users
  - Writer & editor
  - Web designer & browser



Markup Languages

- Give structure and meaning to plain text
- Lightweight overlay
  - Erase and you're back to plain text
- Markup "vocabulary" agreed-upon by users
  - Writer & editor
  - Web designer & browser



Creating Your Own Markup Language

- HTML is one markup language
  - Pretty good for describing web pages
  - Vocabulary includes links, headings, paragraphs, images, etc.
- But what if that's not the information you're interested in?
  - Mark ingredients in recipes so I can use up all of my basil
  - Mark prices in catalogs so I can find a good deal
  - Mark characters in a play so I know who needs to be on stage
  - Mark dictionary words by rarity so I can build shorter editions
- Make up your own markup vocabulary!
  - Apply whatever meaning you want, as long as everyone agrees
- XML: a generic syntax for custom markup languages

XML: Consistent Generic Syntax

- Elements (a.k.a. tags) in angle brackets
  - <ingredient>basil</ingredient>
- Elements have optional attributes
  - <word level="rare">…</word>
  - No duplicate attribute names allowed!
- Abbreviated syntax for empty elements
  - <sale reduction="10%"></sale>
  - <sale reduction="10%" />
  - (Optional space before closing slash has no meaning)
- Make up any element and tag names you want!
  - Will this lead to complete chaos? Maybe, but DTDs will help…

XML: Consistent Generic Syntax

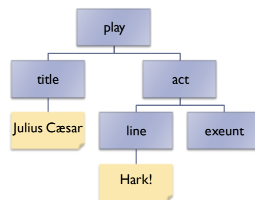
- Some special characters represented as escaped entities
  - “<” and “>” become “<” and “>”
  - “&” becomes “&”
  - “☺” can optionally become “☺” or “&x263A;”
- Elements must be strictly nested and explicitly closed
  - Think {of (elements) [(as)] nested, {matching} parentheses}
  - Which of the following are well-formed XML?
    1. <p>plain **bold** <i>***bold italic*** plain? italic?</p>
    2. <p>plain **bold** <i>***bold italic***</i> **just bold again**</p>
    3. <p>plain **bold** <i>***bold italic***</i> <i>*just italic*</i></p>
- Exactly one top-level root element

XML: Consistent Generic Interpretation

- XML document is … a tree!
  - Strict nesting determines parent/child relationships
  - Elements are nodes
    - Elements may have zero or more ordered children
  - Runs of original text become leaf nodes
    - Cannot have any children
- Attributes are extra info on elements
  - Collection of (name: value) pairs
  - Unordered, unlike child nodes
  - No extra parsing or interpretation of attribute values

XML: Consistent Generic Interpretation

```
<play
  author="Shakespeare">
  <title>Julius Cæsar</title>
  <act setting="hall">
    <line who="Brutus">
      Hark!
    </line>
    <exit who="Brutus"/>
  </act>
</play>
```



(Slight fib: I have omitted whitespace-only text nodes, as is common.)

10

XML Beyond Text Markup

- Remember that idea about erasing the markup to recover plain text?
  - What if we discard this?
- Use XML as a syntax for any tree-structured data
  - Or even non-tree data, though a bit awkward
- Very popular data format
  - Especially for web stuff

```
<recipe>
  <ingredient id= "ingr_01"
    count= "3" units="leaves">
    Basil
  </ingredient>
  …
  <instructions>
    <mix>
      <item ref="ingr_01" />
      <item ref="ingr_02" />
    </mix>
  …
</recipe>
```

11

Total Markup Anarchy?

- You can make up any elements and attributes you want
  - Can any element appear anywhere?
  - Can any attribute appear on any element, with any value?
- Yes and no
  - How carefully do you want to check your XML document?
- Well-formed XML
  - Requires only proper syntax, nesting, entity escaping, etc.
  - Sufficient to ensure you can construct an unambiguous tree
- Validated XML
  - Document tree obeys extra rules about what appears where
  - Rules provided by designer of markup vocabulary (e.g., you!)

12

DTD: Document Type Definition

- Gives the general format of a family of XML documents
  - What are the known element names?
  - Which attributes can each element have?
    - And what are the possible values?
  - What children can each element have?
    - And how many?
    - And what order can they appear in?
- Validating XML parser checks tree against DTD
  - Non-validating parser only checks for well-formed XML
  - Cannot even try to validate a non-well-formed XML document
  - Many parsers offer both validating and non-validating modes

13

Simplified Fragment of HTML DTD

```
<!ELEMENT html (head, body)>

<!ELEMENT body (h1 | h2 | h3 | p | table | ul | hr)*>

<!ELEMENT p (#PCDATA | a | img)*>
<!ATTLIST p style CDATA #IMPLIED>

<!ELEMENT img EMPTY>
<!ATTLIST img src CDATA #REQUIRED>

<!ELEMENT table (caption?, thead?, tfoot?, (tbody+ | tr+))>
```

14

DTD Element Properties

- Ordering of child nodes, if any are allowed
  - Specific order: foo, bar, baz
  - Mixed in any order: foo | bar | baz
- How many repetitions?
  - Zero or one: foo?
  - Zero or more: bar*
  - One or more: baz+
- Special kinds of content: EMPTY, #PCDATA
- Marking up a Shakespearean play
  - <!ELEMENT play (title, prologue?, act+, epilogue?)>
  - <!ELEMENT act (line | enter | exit)+>

15

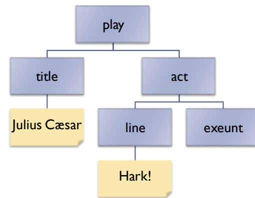
DTD Attribute Properties

- Each element has a list of allowed attributes
  - <!ATTLIST line
    who IDREF #REQUIRED
    mood (happy | sad | neutral) #IMPLIED>
- Each attribute has name, type, and default value
  - Types include CDATA, NMTOKEN, ID, IDREF, enum, …
    - Pretty limited, actually; cannot even require a valid number
  - Default value
    - *value*
    - #REQUIRED
    - #IMPLIED
    - #FIXED *value*

16

OK, I built my XML tree. Now what?

- A data definition language is only useful if you can get data back out of it
- Use tree paths to describe the data you want
  - /play
  - /play/title
  - /play/act/line
    - (Which line?)
- Welcome to XPath!



17

XPath: XML Data Extraction Patterns

- Paths are slash-delimited, each level naming an element
  - /play/title
  - /html/body/table/tr/td
  - tr/td/p
  - ../act/enter
- Wildcards
  - * matches any one node: /play/*/line
  - // matches zero or more nodes: /html/body//table//a/*/img
- Attributes available at the leaves using @name
  - /recipe/ingredient/@units
  - //@lang

18

Being More Selective

- If pattern matches multiple parts of tree, get all of them
  - /play/act/line: every line in every act, in document order
  - But what if you only want some of them?
- Restrictions in square brackets anywhere along the path
  - /play/act/line[@who = "Brutus"]
- XPath functions give more info about current node
  - /play/act[position() = 2]/line[text() = "Hark!"]/@who
- Special syntax simplifies some common cases
  - Number is treated as position check: /play/act[3]/line[last()]
  - Node set matches if non-empty: /play[epilogue]/title

19

XPath Can Get Pretty Fancy

- Text of the last line in the play
  - /play//line[not preceding::line]/text()
- Schools that the Badgers played against
  - /scores/game[team = "Badgers"]/team[. != "Badgers"]
- Extract TV listing from HTML page (screen-scraping)
  - //div[@class = "times"]//dt[text() = "Scrubs"]/../ul/li[2]
- However, it's still just a one-time query
  - Good start, but not enough for complex data transformations

20

XSLT: XML Transformation Language

- XSLT is a fully general programming language
  - Highly specialized for transforming XML into XML
- Why would you want to do this?
  - To generate HTML pages from other structured data
  - To convert data in one structured format into another format
  - To extract data using more powerful tools than XPath
- So why did we bother with XPath?
  - XSLT uses XPath extensively to match and extract data
  - Think of XSLT as an XPath-based XML reorganizer

21

General Style of an XSLT Script

- XSLT script is a collection of templates
  - Each template has an XPath pattern + commands to run
  - Use XPath pattern to match fragments of XML document tree
  - When a template matches, run the commands
- If no match, default behavior kicks in
  1. Default for text nodes: copy text to result tree
  2. Default for element nodes: recursively descend
- Start by matching document root, "/"
  - Might match an XSLT template, or might recursively descend

22

Warning! Amazingly ugly syntax ahead!

- What syntax should XSLT programming language use?
  - Curly braces and semicolons like Java, C, C++, C#, JavaScript?
  - Nested parentheses like Lisp?
  - Whitespace-delimited commands like Unix shells?
- Of course not. Don't be silly. ☺
- XSLT programs are structured, and we already have a perfectly good (?) syntax for structured data
  - XSLT is represented using … XML!
  - <xsl:if test="…">…</xsl:if>
  - <xsl:for-each select="…">…</xsl:for-each>
  - <xsl:variable name="inches" select="@cm / 2.54" />

23

Partial XSLT Play-to-HTML Converter

```
<xsl:template match="act">
  <h1>Act <xsl:value-of select="position()"></h1>
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="line">
  <p>
    <xsl:value-of select="@who"/>:
    <xsl:apply-templates/>
  </p>
</xsl:template>
```

24

Partial XSLT Play-to-HTML Converter

```
<xsl:template match="exit">
  <xsl:variable name="gone" select="@who" />
  <p>Exit <xsl:value-of select="$gone"/>.</p>
  <xsl:if test="not following::enter[@who = $gone]">
    <p class="stage-direction">
      <xsl:value-of select="$gone" />
      may now leave the theatre.
    </p>
  </xsl:if>
</xsl:template>
```

25

If going to HTML, what was the point?

- Move between XML vocabularies (not just HTML)
  - Data exchange, conversion, mining, migration, etc.
- Rehearsal plans
  - Get list of unique characters appearing in each act
  - Cross-reference with names of actors for each role
  - Convert to OOXML or ODF (XML for word processing)
  - Print "call sheet" saying who needs to be at rehearsal
- Cookbook of just basil recipes
  - "Too Many Tomatoes: A Cookbook for When Your Garden Explodes"
- Rewrite badly-designed web sites within the browser?

26

Why I'm Doing This Lecture

- My Curriculum Vitae is an XML document
  - Automatically convert to HTML, PDF (via LaTeX), plain text
  - Automatically extract conflict-of-interest lists: everyone I've written a paper with in the last five years
- My class schedules are XML documents
  - Generate meeting dates automatically
  - Generate HTML table for posting on class web page
  - Automatically extract calendar records for Google Calendar
- Much of my research data is stored as XML
  - XSLT transformations to HTML for rapid prototyping of alternative ways to explore our results

27

Buyer Beware: Some XML Caveats

- Syntax can be very verbose
  - <date><year>2007</year><month>11</month><date>20…
  - Lisp-like alternative: (date (year 2007) (month 11) (date 20))
- Too much design flexibility and little standardized policy
  - <date><year>2007</year><month>11</month><date>20…
  - <date>2007-11-20</date>
  - <date when="2007-11-20"/>
- Human-readable?
  - In theory, yes
  - In practice, sometimes not

28

Buyer Beware: Some XML Caveats

- Doesn't work too well for arbitrary binary data
  - Need to encode using allowed subset of chars
  - <data base64="iVBORw0KGgoAAANSUhEUgAAAJAAA…"/>
- DTDs have limited expressive power, quirky syntax
  - Might need to allow "valid" documents you don't really like
  - Popular alternatives: W3C Schema, RELAX NG
- XPath cannot do absolutely everything
  - XQuery: SQL for XML
  - Use XPath within more full-featured programming languages
- XSLT: worst programming syntax ever invented
  - Call it selectively from within standard programming languages

29

Summary of What We've Seen

- XML: generic syntax for tree-structured data
  - Well-formed XML must obey some simple rules
- DTD: define grammars for particular ways of using XML
  - Valid (or validated) XML documents obey some given DTD
  - Properly authored HTML is XML and obeys the HTML DTD
- XPath: data extraction based on path matching
  - Compact and usually easy to read
  - Limited expressive power; cannot solve every problem
- XSLT: domain-specific language for XML transformation
  - Systematically match input tree and generate output tree
  - Very powerful tool if your task fits its model

30