Principles of Information and Database Management 198:336 Week 8 – Mar 28

Matthew Stone

XML – Motivations

Semi-structured data
- Relaxing traditional schema
- Storing more complex objects

Standardized data
- Using reference schemas for interoperability
- “Meta-data” – language for data description

Web data
- Supported in protocols for information exchange

Outline

XML – overview

XML data representations

XML and standardization
- XML namespaces
- XML resource description framework

XML and the web
- XHTML
- Cascading style sheets and XSLT

XML

eXtensible Markup Language
- “File format” for giving partial structure to text documents.
- Based on the use of **paired tags** to give a **tree structure** to the document.

Tags in XML

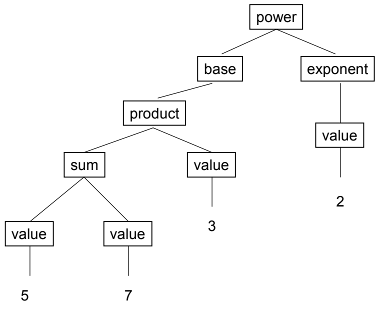
Work like parentheses…

$[(5 + 7) * 3]^2$

But make category of structure explicit

power(product(sum(5,7), 3), 2)

Tree visualization



Basic tag syntax

<tag>
- open a tag

</tag>
- close a tag

Example becomes

```
<power>
   <base>
        <product>
                <sum>
                        <value>5</value>
                        <value>7</value>
                </sum>
                <value>3</value>
        </product>
   </base>
   <exponent><value>2</value></exponent>
</power>
```

Storing data in XML

Relational data
- Combines schema and tuples together

Example
- Schema
  student(id:integer, name:string, email:string)
- Tuple
  (65, “Teddy Salad”, tds@mp.com”)

Storing relational data in XML

In XML, encode table

```
<student>
…
</student>
```

Storing relational data in XML

Then columns…

```
<student>
<id> … </id>
<name> … </name>
<email> … </email>
</student>
```

Storing relational data in XML

Then values…

```
<student>
<id>65</id>
<name>Teddy Salad</name>
<email>tds@mp.com</email>
</student>
```

Storing relational data in XML

For whole tables, just repeat

```
<tableOfStudents>
   <student>
        <id>64</id>
        <name>Anne Elk</name>
        <email>ae@bronto.mp.com</email>
   </student>
   <student>
        <id>65</id>
        <name>Teddy Salad</name>…
```

Storing data in XML

Text data

- Elements can be freeform text
- Elements can be further “marked up” to indicate presentation or structure

Storing text data in XML the basics

```
<text>
Elk: Yes, well you may well ask me what is my
   theory.
Presenter: I am asking.
Elk: Good for you.  My word yes.  Well Chris, what
   is it that it is – this theory of mine. Well, this is
   what it is – my theory that I have, that is to say,
   which is mine, is mine.
</text>
```

Storing text data in XML markup

```
<drama>
<line><player>Elk</player>
<content>Yes, well you may well ask me what is my
   theory.</content></line>
<line><player>Presenter</player>
<content>I <loud>am</loud> asking.</content></line>
<line><player>Elk</player>
<content>Good for you.  My word yes.  Well Chris, what is
   it that it is – this theory of mine. Well, this is what it is –
   my theory that I have, that is to say, which is mine, is
   mine.</content></line>
</drama>
```

Storing data in XML

Mix – partly well-defined, partly open-ended

- Example: product descriptions
- Name, description – formatted text
- Nutrition information – content FDA requires

Storing mixed data in XML

```
<product>
<info><name>California trail mix</name>
<description>We mix sweet <loud>ripe</loud> fruit with
   <loud>premium</loud> nuts to bring you the taste of <loud>pure
   energy</loud>…</description></info>
<nutrition><servings><size>1/4 cup</size>
                    <per>about 27</per></servings>
           <calories><total>120</total>
                    <fat>25</fat></calories>
        … </nutrition>
</product>
```

Describing data

DTDs – “document type definitions”

- – Original proposal for XML
- – Describes possible patterns of elements
- – Grammar with regular expression syntax

DTD examples

```
<!ELEMENT loud (#PCDATA) >
<!ELEMENT description (#PCDATA | loud)* >
<!ELEMENT name (#PCDATA) >
<!ELEMENT info (name, description) >
```

DTDs

Not very specific

- – Don’t constrain types of values
- – Don’t indicate links to standards
- – Can only see one layer of structure at a time

XML Schema

Give a template for a document

- – as more XML!
- – Complicated syntax, but powerful.

XML Schema examples

Loud

<element name=“loud” type=“string” />

Name

<element name=“name” type=“string” />

Hey, what’s all that junk?

XML also has empty tags

<foo></foo>
is the same as
<foo />

Hey, what’s all that junk?

XML also has attributes on opening tags

```
<tag attribute=“value” >
```

Hey, what’s all that junk?

So

```
<element name=“loud” type=“string” />
```

Defines an empty element

```
<element name=“loud”
            type=“string”></element>
```

- Whose name attribute has value “loud”
- Whose type attribute has value “string”

XML Schema Examples

Description

```
<element name=“description”>
 <complexType mixed=“true”>
    <choice minOccurs=“0”
              maxOccurs=“unbounded”>
       <element name=“loud” type=“string” />
    </choice>
  </complexType>
</element>
```

XML Schema Examples

Easier to define your own types

```
<complexType name=“descriptionType” mixed=“true”>
     <choice minOccurs=“0”
                maxOccurs=“unbounded”>
         <element name=“loud” type=“string” />
     </choice>
</complexType>
```

XML Schema Examples

Info

```
<complexType name=“infoType”>
   <sequence>
       <element name=“name” type=“string” />
       <element name=“description”
                  type=“descriptionType” />
   </sequence>
</complexType>
<element name=“info” type=“infoType” />
```

What’s the point?

Even with semi-structured data

- You can check that your data falls in a specific range of possibilities
- Validation

Problems:

What about files created by scripts?

Standardization

What schema are you using?

- Does your element <name> mean the same thing as my element <name>?
- If your license gives me
  <permission action="copy" />
  do I really know what I can do with your data?

Key principle

Need a way to uniquely identify tokens as instances of known concepts.

Compare: UPC codes, ISBN numbers

Solution

Use URLs/URIs

- Uniform resource locators
- Uniform resource identifiers

Build on the existing infrastructure to avoid clashing names on the web.

Example

The official DTD for XHTML 1.0 strict

- A standard for describing hypertext web documents as XML

lives here

http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd

(a URL)

Example

A standard reference for the concepts associated with XHTML is this URI

http://www.w3.org/1999/xhtml

Using this "namespace" means your intended meaning for your document is what is spelled out there.

Using namespaces

```
<tag1 xmlns:ns="URI">
   …. <ns:tag2 … />
</tag1>
```

Declared using xmlns attribute

Used using ":" syntax

Metadata

Data about data

- We've seen one example: schemas
- If you are building a document that respects a particular XML Schema, you can say so

```
<product xmlns="URL"
    xmlns:xsi=http://www.w3c.org/2001/XMLSchema-instance
   xsi:schemaLocation="URL2">
…
</product>
```

Metadata

In general, XML metadata is

- An XML description in a specified language
- That you link to as a specified attribute of designated elements

Resource Description Framework

RDF is a particular set of concepts for describing metadata

- Also known as "the semantic web"

Includes

- "Dublin core" concepts for computer science and representation
- OWL and DAML concepts for services
- eXtensibly linked to other concept sets

Example: Creative Commons

http://www.creativecommons.org

- Develops culture-friendly licenses for distributing web content
- Motto: "some rights reserved"

- Licenses are distributed as RDF files granting specific permissions and reserving rights
- Creative commons maintains an XML namespace and URIs for licences and concepts used in them.

Querying XML

How do you find places in a tree?

By nodes

- Category
- Attributes

By paths

- Location
- Ancestor
- Child
- Sequence

Example: XML stylesheets

Controls the layout of XML data when presented in a web browser.

Rules of the form

Pattern { Actions }

Patterns can be seen as queries over data trees.

Stylesheet patterns

Category

- – Matches any node of type Category

Category.sub

- – Matches any node of type Category whose class attribute has the value “sub”

Stylesheet patterns

ParentType > ChildType

- – Matches any node of ChildType whose parent is a node of type ParentType

ParentType ChildType

- – Maches any node of ChildType that has an ancestor of type ParentType

Stylesheet patterns

Attribute selectors (new)

myElement[myAttribute]
myElement[myAttribute=“myValue”]
myElement[myAttribute~=“myValue”]
myElement[myAttribute|=“myValue”]

Key points

Classic issues in data

- – Design
- – Representation
- – Query
- – Declare
- – Tell
- – Validate