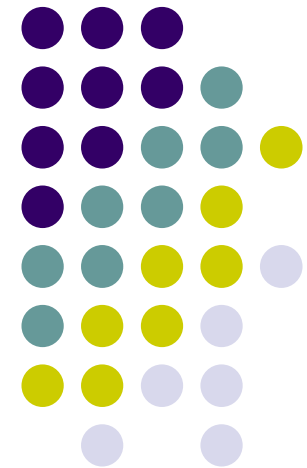


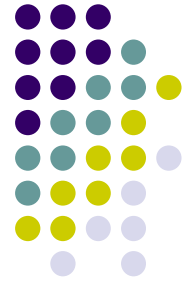
Internet Applications I

Web Data Formats

Chapter 7

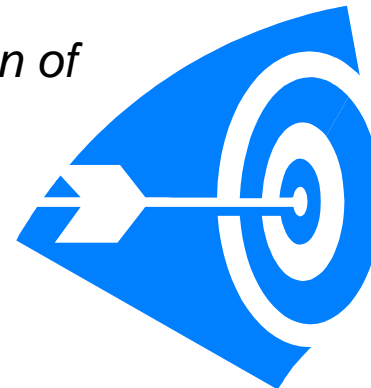


Internet Applications Overview



- **Internet Concepts**
 - **Web data formats**
 - HTML, XML, DTDs
 - **The three-tier architectures**
 - Database Layer
 - Application Logic Layer
 - Presentation Layer
- THIS LECTURE
- NEXT LECTURE

Our Goal: Understanding and Construction of Database-Backed Internet Applications



Uniform Resource Identifiers



- Uniform naming schema to identify *resources* on the Internet
- A resource can be any type of file:
 - curriculumvitae.html
 - pokerface.mp3
 - foobarparty.jpg
 - myvirus.exe
- Example URIs:
<http://www.cs.wisc.edu/~dbbook/index.html>
<mailto:webmaster@bookstore.com>

Structure of URIs



<http://www.cs.wisc.edu/~dbbook/index.html>

- URI has three parts:
 - Naming schema ([http](http://www.cs.wisc.edu/~dbbook/index.html))
 - Name of the host computer ([www.cs.wisc.edu](http://www.cs.wisc.edu/~dbbook/index.html))
 - Name of the resource ([~dbbook/index.html](http://www.cs.wisc.edu/~dbbook/index.html))

- URLs are a subset of URIs

<http://www.w3.org/TR/uri-clarification/>

Hypertext Transfer Protocol



- What is a communication protocol?
 - Set of standards that defines the structure of messages
 - Examples: TCP/IP, **HTTP**
- What happens if you click on www.cs.wisc.edu/~dbbook/index.html?
 1. Client (web browser) sends HTTP request to server
 2. Server receives request and replies
 3. Client receives reply; makes new requests

HTTP (Contd.)

SERVER REPLIES

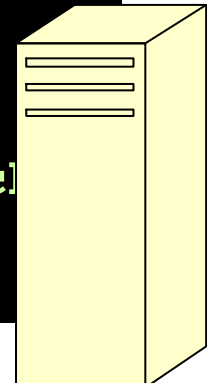


CLIENT TO SERVER

```
GET
  www.fws.nl/index.html
HTTP/1.1
User-agent: Mozilla/4.0
Accept: text/html,
  image/gif, image/jpeg
```



```
HTTP/1.1 200 OK
Date: Mon, 04 Mar 2002
  12:00:00 GMT
Server: Apache/1.3.0
  (Linux)
Last-Modified: Mon, 01
  Mar 2002 09:23:24 GMT
Content-Length: 1024
Content-Type: text/html
<HTML> <HEAD></HEAD>
<BODY>
<h1>Fun with
  Statistics</h1>
Our recent statistics:
<h3>Science</h3>
<b>Distribution of Duc
  in the Netherlands
  </b> ...
```



HTTP Protocol Structure



HTTP Requests

```
GET
  www.fws.nl/index.html
HTTP/1.1
User-agent: Mozilla/4.0
Accept: text/html,
       image/gif, image/jpeg
```

- Request line: **GET**
~/index.html HTTP/1.1
 - **GET**: http method field (possible values are GET and POST (used for filling templates), more later)
 - **~/index.html**: URI field
 - **HTTP/1.1**: HTTP version field
- Type of client: **User-agent: Mozilla/4.0**
- What types of files will the client accept: **Accept: text/html, image/gif, image/jpeg**

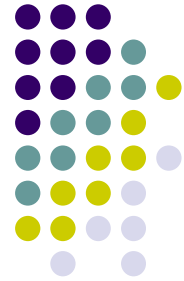
HTTP Protocol (Contd.)

HTTP Responses

- Status line: `HTTP/1.1 200 OK`
 - HTTP version: `HTTP/1.1`
 - Status code: `200`
 - Server message: `OK`
 - Common status code/server message
 - `200 OK`: Request succeeded
 - `400 Bad Request`: Request could not be fulfilled by the server
 - `404 Not Found`: Requested object does not exist on the server
 - `505 HTTP Version not Supported`
- Date when the object was created:
`Last-Modified: Mon, 01 Mar 2002 09:23:24 GMT`
- Number of bytes being sent: `Content-Length: 1024`
- What type is the object being sent: `Content-Type: text/html`
- Other information such as the server type, server time, etc.

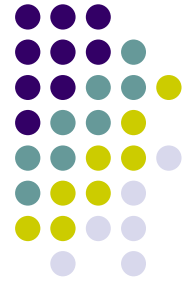
```
HTTP/1.1 200 OK
Date: Mon, 04 Mar 2002
    12:00:00 GMT
Server: Apache/1.3.0 (Linux)
Last-Modified: Mon, 01 Mar
    2002 09:23:24 GMT
Content-Length: 1024
Content-Type: text/html
<HTML> <HEAD></HEAD>
<BODY>
<h1>Fun with Statistics</h1>
Our recent statistics:
<h3>Science</h3>
<b>Distribution of Ducks in
    the Netherlands </b> ...
```

Some Remarks About HTTP



- HTTP is *stateless*
 - No “sessions”
 - Every message is completely self-contained
 - No previous interaction is “remembered” by the protocol
- Implications for applications:
 - Any state information (shopping carts, user login-information) need to be encoded in every HTTP request and response!
 - Popular methods on how to maintain state:
 - **Cookies** (later this lecture)
 - Dynamically generate unique URL’s at the server level (later this lecture)

Web Data Formats



- HTML (HyperText Markup Language)
 - The presentation language for the Internet
- XML (eXtensible Markup Language)
 - A self-describing, hierarchal data model
- DTD (Data Type Definition)
 - Standardizing schemas for XML
- XSLT (not covered in the book) – makes HTML from XML

HTML: An Example



```
<HTML>
  <HEAD></HEAD>
  <BODY>
    <h1>Barns and Nobble Internet
    Bookstore</h1>
    Our inventory:
    <h3>Science</h3>
    <b>The Character of Physical
    Law</b>
    <UL>
      <LI>Author: Richard
      Feynman</LI>
      <LI>Published 1980</LI>
      <LI>Hardcover</LI>
    </UL>
```

```
<h3>Fiction</h3>
<b>Waiting for the
Mahatma</b>
<UL>
  <LI>Author: R.K. Narayan</LI>
  <LI>Published 1981</LI>
</UL>
</BODY>
</HTML>
```

HTML: A Short Introduction



- HTML is a markup language
- Commands are tags:
 - Start tag and end tag
 - Examples:
 - `<HTML> ... </HTML>`
 - ` ... `
- Many editors automatically generate HTML directly from your document (e.g., Microsoft Word has an “Save as html” facility)

HTML: Sample Commands



- `<HTML>`:
- ``: unordered list
- ``: list entry
- `<h1>`: largest heading
- `<h2>`: second-level heading, `<h3>`, `<h4>` analogous
- `Title`: Bold



HTML Hyperlinks

```
<a href="My favorite course">  
www.liacs.nl/databases </a>
```

```
<a href="My favorite song"> ~/alidb.mp3 </a>
```

XML – What's The Point?



- XML document stores data and a description of what the data represents
- Example: Chemical Markup Language

```
<molecule>  
    <weight>234.5</weight>  
    <Spectra>...</Spectra>  
    <Figures>...</Figures>  
</molecule>
```
- XML design goals:
 - XML should be compatible with SGML and HTML
 - It should be easy to write XML processors
 - The design should be formal and precise
- Constraints on Syntax can be formulated as DTD
DTD = 'Schema for a XML Data Repository'
(example: For each molecule we have to specify a weight and figure but spectra are optional)



XML – What's the point

- XML/DTD is a metalanguage
- A metalanguage is a language that is used to define other languages; You can use XML for instance to create FunML
- XML is a easier version of SGML



XML: What is it good for?

With XML/DTD you can:

- Define data structures and store/exchange data
- Make structures platform independent and easily exchangeable/editable (ASCII format)
- Process data automatically using XML parser tools such as XPath

XML cannot

- Define how the data is graphically represented; additional presentation specifications and tools needed; this is a major difference to HTML



Showing the results

- Often it is not needed to display the data; the XML file may be stored in a database (e.g. a molecule description in a chemical database)
- If you need to show data you can use either:
 - XSL (eXtensible Stylesheet Language) – generates HTML from XML, HTML can be displayed by Webbrowser
 - CSS (Cascading Stylesheets)

XML: An Example



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BOOKLIST>
  <BOOK genre="Science" format="Hardcover">
    <AUTHOR>

    <FIRSTNAME>Richard</FIRSTNAME><LASTNAME>Feynman</LASTNAME>
    </AUTHOR>
    <TITLE>The Character of Physical Law</TITLE>
    <PUBLISHED>1980</PUBLISHED>
  </BOOK>
  <BOOK genre="Fiction">
    <AUTHOR>
    <FIRSTNAME>R.K.</FIRSTNAME><LASTNAME>Narayan</LASTNAME>
    </AUTHOR>
    <TITLE>Waiting for the Mahatma</TITLE>
    <PUBLISHED>1981</PUBLISHED>
  </BOOK>
</BOOKLIST>
```

XML – The Extensible Markup Language



- Language
 - A way of communicating information
- Markup
 - Notes or meta-data that describe your data or language
- Extensible
 - Limitless ability to define new languages or data sets

XML – Structure



- XML looks like HTML
- XML is a hierarchy of user-defined tags called elements with attributes and data
- Data is described by elements;
- Elements can have attributes with values

`<BOOK genre="Science" format="Hardcover">...</BOOK>`

↑
open tag
element name

↑
attribute

↑
attribute value

↑
data

↑
closing tag

XML – Elements



<BOOK genre="Science" format="Hardcover">...</BOOK>



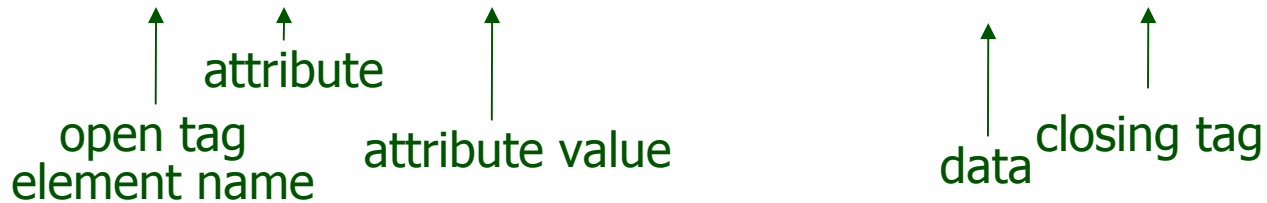
- Xml is case and space sensitive
- Element opening and closing tag names must be identical
- Opening tags: “<” + element name + “>”
- Closing tags: “</” + element name + “>”
- Empty Elements have no data and no closing tag:
 - They begin with a “<” and end with a “/>”

<BOOK/>

XML – Attributes



`<BOOK genre="Science" format="Hardcover">...</BOOK>`



- Attributes provide additional information for element tags.
- There can be zero or more attributes in every element; each one has the the form:
attribute_name='attribute_value'
 - There is no space between the name and the “=”
 - Attribute values must be surrounded by “ or ‘ characters
- Multiple attributes are separated by white space (one or more spaces or tabs).

XML – Data and Comments



<BOOK genre="Science" format="Hardcover">...</BOOK>



- Xml data is any information between an opening and closing tag
- Xml data must not contain the '<' or '>' characters
- Comments (like in HTML):
<!-- comment -->

XML – Nesting & Hierarchy



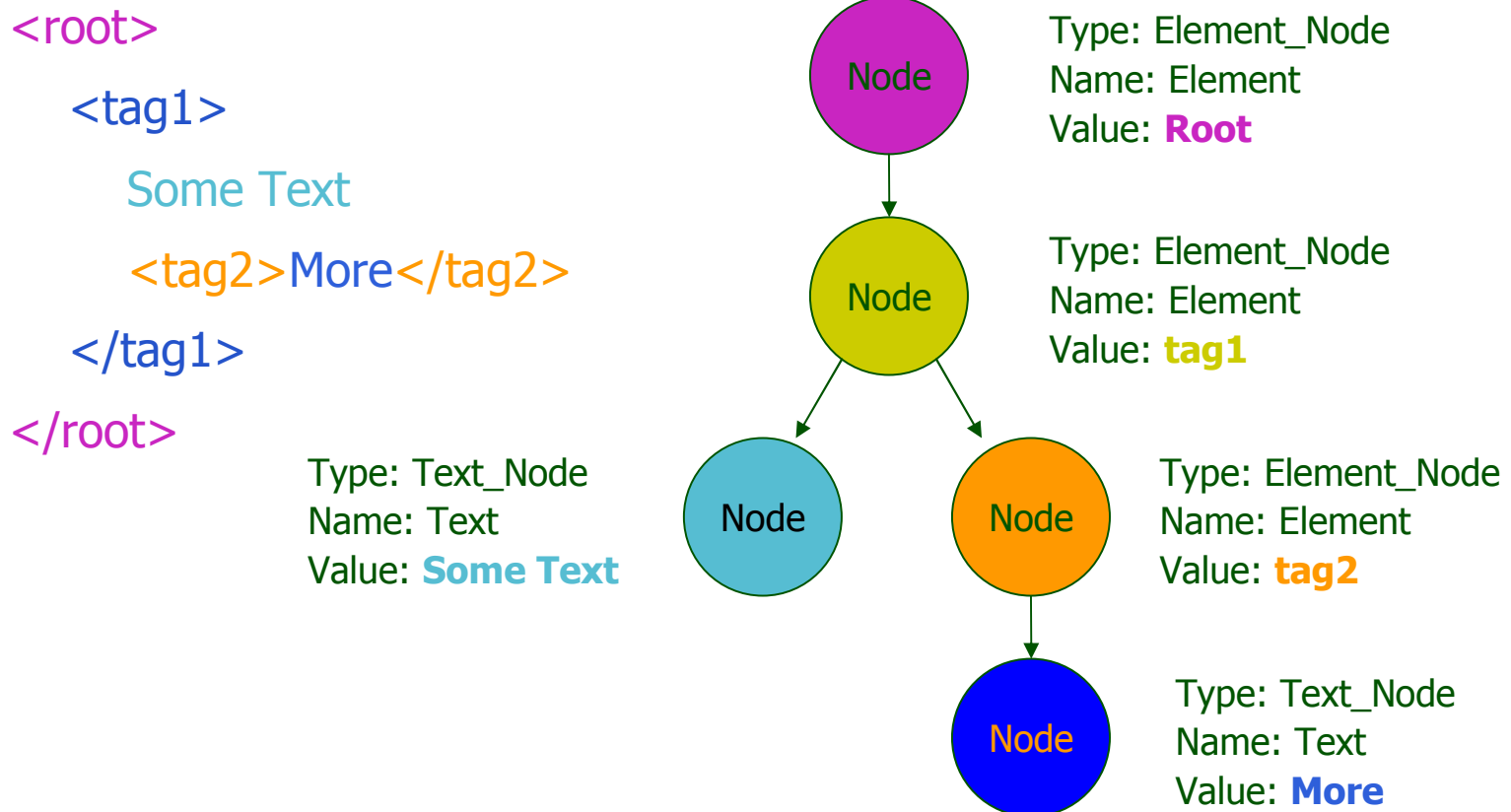
- Xml tags can be nested in a tree hierarchy
- Xml documents can have only one root tag
- Between an opening and closing tag you can insert:
 1. Data
 2. More Elements
 3. A combination of data and elements

```
<root>  
  <tag1>  
    Some Text  
    <tag2>More</tag2>  
  </tag1>  
</root>
```

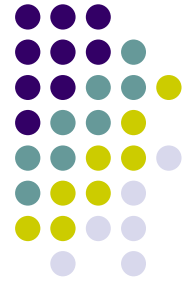
Xml – Storage



- Storage is done just like an n-ary tree



DTD – Document Type Definition



- A DTD is a schema for Xml data
- Xml protocols and languages can be standardized with DTD files
- A DTD says what elements and attributes are required or optional
 - Defines the formal structure of the language

DTD - !ELEMENT (Contd.)



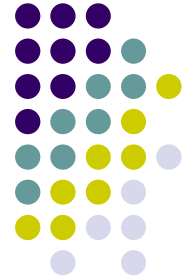
A **regular expression** is a string, e.g.

“Cherry+, (Apple | Orange)*”

with the following structure:

- $exp_1, exp_2, exp_3, \dots, exp_k$: A list of regular expressions
- exp^* : An optional expression with zero or more occurrences
- exp^+ : An optional expression with one or more occurrences
- $exp_1 | exp_2 | \dots | exp_k$: A disjunction (choice) of expressions
- Example: One or more cherries and optionally some apples and oranges, but not apples and oranges at the same time.

DTD – An Example



```
<?xml version='1.0'?>
<!ELEMENT Basket (Cherry+, (Apple | Orange)*) >
  <!ELEMENT Cherry EMPTY>
    <!ATTLIST Cherry flavor CDATA #REQUIRED>
  <!ELEMENT Apple EMPTY>
    <!ATTLIST Apple color CDATA #REQUIRED>
  <!ELEMENT Orange EMPTY>
    <!ATTLIST Orange location 'Florida'>
```

What does this say?



```
<Basket>
  <Cherry flavor='good'/>
  <Apple color='red'/>
  <Apple color='green'/>
</Basket>
```



```
<Basket>
  <Apple/>
  <Cherry flavor='good'/>
  <Orange/>
</Basket>
```

DTD - !ELEMENT



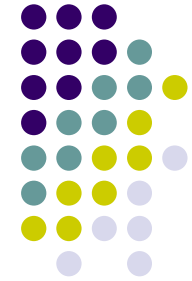
<!ELEMENT Basket (Cherry+, (Apple | Orange)*) >

Name

Children

- **!ELEMENT** declares an element name, and what children elements it should have
- Content types:
 - Other elements
 - **#PCDATA** (parsed character data)
 - **EMPTY** (no content)
 - **ANY** (no checking inside this structure)
 - A regular expression

DTD - !ATTLIST



`<!ATTLIST Cherry flavor CDATA #REQUIRED>`

Element Attribute Type Flag

`<!ATTLIST Orange location CDATA #REQUIRED
color 'orange'>`

- **!ATTLIST** defines a list of attributes for an element
- Attributes can be of different types, can be required or not required, and they can have default values.

DTD – Well-Formed and Valid



```
<?xml version='1.0'?>  
<!ELEMENT Basket (Cherry+)>  
  <!ELEMENT Cherry EMPTY>  
  <!ATTLIST Cherry flavor CDATA #REQUIRED>
```

Not Well-Formed

```
<basket>  
  <Cherry flavor=good>  
</Basket>
```

Well-Formed but Invalid

```
<Job>  
  <Location>Home</Location>  
</Job>
```

Well-Formed and Valid

```
<Basket>  
  <Cherry flavor='good'/>  
</Basket>
```



DTD External

- A DTD can be an external document that is referred to:

The following line should be typed after the line

```
<?xml version="1.0"?>
```

```
<!DOCTYPE name of root-element SYSTEM  
"address">
```

or, if you would like to include the DTD in the document

```
<!DOCTYPE name of root-element []>
```

XML and DTDs



- More and more standardized DTDs will be developed
 - MathML
 - Chemical Markup Language
- Allows light-weight exchange of data with the same semantics
- Sophisticated query languages for XML are available:
 - Xquery
 - XPath

XML-Schema Definitions (XSD)



- Alternative to XML-DTD
- Recently (2009) declared as a new standard
- Stronger typing and conformity with programming languages (Java, Python)
- Became a w3c standard in 2009



XML Schema Datatypes

- xsd:string
- xsd:decimal
- xsd:integer
- xsd:float
- xsd:boolean
- xsd:date
- xsd:time

XML Schema



- QName: Qualified Name, globally unique identifier.
- Consists of NCNames (Non-Colonized Names),
- Every name, except the last one, defines a "Namespace".
- The last NCName is a local name in the namespace
- A '.' is used to separate the namespace identifiers and the local name

XML Schema



- anyURI: Uniform Resource Identifier ([URI](#))
- language: language identification de-DE, en-US, fr
- ID: Identification label
- IDREF: Reference to a ID-Label

Simple Types



```
<xsd:simpleType name="monthInt">
  <xsd:restriction
    base="xsd:integer">
    <xsd:minInclusive value="1"/>
    <xsd:maxInclusive
value="12"/>
    </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="months">
  <xsd:list itemType="monthInt"/>
</xsd:simpleType>
```

```
<months> 1 2 3 4 5 6 7 8 9 10 11 12 </months>
```

- Simple types are not allowed to include more than one nested structures
- A list is used to define a list of multiple instances of a simple type

Unions in XML Schema



A list of month can now have strings (such as “Mar”) but also integers (such as 3).

```
<xsd:simpleType name="monthname">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Jan"/>
    <xsd:enumeration value="Feb"/>
    <xsd:enumeration value="Mar"/>
    <!-- and so on ... -->
  </xsd:restriction> </xsd:simpleType>
<xsd:simpleType name="month">
<xsd:union memberTypes="monthname monthInt"/>
</xsd:simpleType>
```

Complex Data Types XSD



```
<xsd:complexType name="pc-Type">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="produced" type="xsd:string"/>
    <xsd:element name="processor" type="xsd:string"/>
    <xsd:element name="mhz" type="xsd:integer"
      minOccurs="0"/>
    <xsd:element name="comment" type="xsd:string"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:integer"/>
</xsd:complexType>
```

Example of a complex data type in XSD, see W3C documentation for further details



Conclusions

- HTTP provides a simple, stateless, protocol for the communication of a Web Client with a Web Server
- Resources on the server are specified via Unified Resource Identifier
- HTML is *the* hypertext language used for coding presentable documents in the Web
- As HTML, XML is using tags and attributes
- XML is not in the first place used for presentation, but for a light-weight exchange of structured data
- DTDs is classically used to specify XML formats
- XML-Schema (XSD) recently becomes a new standard. It offers additional concepts and types, but at the same time complexity of documents is increased

Overview and Outlook



- Internet Concepts ✓
- Web data formats
 - HTML, XML, DTDs ✓
- Introduction to three-tier architectures
- The presentation layer
 - HTML forms; HTTP Get and POST, URL encoding; Javascript; Stylesheets. XSLT
- The middle tier
 - CGI, application servers, Servlets, JavaServerPages, passing arguments, maintaining state (cookies)