Lecture 23

Web Services
March 21, 2005

Document Access
- Web servers: documents are HTML pages, images, videos, ...
- An HTML page contains 1+ documents:
  - retrieving a page with n images requires n+1 server requests
  - client (browser) then assembles documents to form a page
  - each document is designated by a URL (Universal Remote Locator)

Web Servers
- Static files are not enough.
- Some documents are dynamic:
  - search engines, page counters, etc.
  - pages tailored to users’ preferences
- Some documents are static but it’s easier to generate them dynamically:
  - web sites with 10000+ pages, high update frequency
  - editing pages is time-consuming:
    - separate content from presentation
    - raw content is stored in databases, program generates pages on the fly

Web Standards
- Q1: which language do I write my documents in? (HTML, XML, …)
- Q2: how does a user indicate which document to load? (URL)
- Q3: how does the browser fetch the document? (HTTP)
- Q4: how does the browser know how to display the document? (MIME)
URL
- Uniform Resource Locator
- Defined by RFC 1738
- Format: <protocol_id>:<address>
  - mailto:cpoellab@cse.nd.edu
  - news:comp.os.research
- HTTP, HTTPS, FTP, ...: address is standardized:
  - <protocol_id>://<machine>[:<port>]/<path>
  - http://www.cse.nd.edu/~cpoellab/

MIME
- Multipurpose Internet Mail Extensions
- Associate a type and an encoding to a document:
  - give a meaning to a set of bytes
  - how should I display this?
- RFC 1521 and 1522
- Originally a standard for email (attached documents must have a type so that the mail program can display them).
- Web uses a subset of MIME (typing and encoding).

MIME
- Format of a MIME type:
  - text/plain, text/html, image/gif, application/pdf
- Format of an encoding:
  - base64, x-gzip
- Each document delivered with MIME type and encoding:
  - if necessary, document is decoded (base64-decode, gunzip)
  - type can be of 3 kinds:
    - browser knows it and has internal display functions (HTML, GIF): document is displayed internally
    - browser knows which application can view it (postscript, Word): document is viewed in external viewer
    - browser does not know it: document is saved in a file

HTTP
- HyperText Transfer Protocol
- HTTP/1.1: complex, described in RFC2617, current.
- HTTP/1.0: simple, described in RFC1945, obsolete.
- RPC-like protocol:
  - connection
  - client sends request to server
  - server answers the request
  - disconnection
  - repeat if more documents to fetch

HTTP
- Connection Phase:
  - TCP connection initiated by client
  - default: port 80 (or otherwise indicated in URL)

HTTP
- Request Phase:
  - <Method> <Path> <HTTP_version> /...
  - <Optional_field> <Value> /...
  - Methods: GET, HEAD, POST, PUT, ...
  - Optional fields: user-agent (id of browser), host: server host name (used for virtual hosts), if-modified-since: transfer document only if it has been modified
  - authorization: username+password
  - empty line indicates end of request
HTTP

- Response Phase:
  - \(<HTTP\_version> <Response\_code> <Text>\)
- Content-Type: \(<MIME\_type>\)
- <Optional\_field>: \(<Value>\)
- <Document>

HTTP

- Response Code:
  - 2xx: Success (200='OK', 201='Created', etc.)
  - 3xx: Redirection (301='Permanent', 302='Temporary', etc.)
  - 4xx: Client Error (400='Bad Request', 401='Unauthorized', 403='Forbidden', 404='Not Found', etc.)
  - 5xx: Server Error (500='Internal Server Error', 501='Not Implemented', etc.)

HTTP

- Response text: same information as Response_code, in plain text.
- Optional fields:
  - Date: date of request
  - Last-Modified: date of last modification of document
  - Server: identification of the server
  - Content-Length: size of document (bytes)

HTTP/1.1

- If a page has one HTML document and 10 images:
  - 11 connections/disconnections required
  - Inefficient: 3WHS, slow start
  - HTTP/1.1: reuse connection
  - Virtual hosts are non-standard extensions of HTTP/1.0: standardized in HTTP/1.1.
- Content negotiation:
  - Document can exist in different versions (GIF vs JPG vs PNG; English vs French vs German; compressed vs uncompressed)
  - HTTP/1.1 allows client to specify preferences (PNG over GIF, French over German): server selects 'best' version

HTML

- SGML: Standard Generalized Markup Language:
  - Meta-standard for defining standard document formats:
    - Each instance of SGML is a DTD (Document Type Definition) or formal grammar specification
    - Each version of HTML is defined as one DTD
    - Advantage: a non-ambiguous specification, one can use syntax checking tools, etc.
    - But everybody writes wrong HTML pages; browsers have to be tolerant
    - One online syntax checker: http://validator.w3.org
HTML

```html
<!DOCTYPE ...>
<html>
<head>
<title>...</title>
...</head>
<body>
...</body>
</html>
```

HTML

- **DOCTYPE**: defines which DTD to use:
  ```html
  <!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
  </DOCTYPE>
  <DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
  ```

- **Headers**: `<h1>...</h1>`, ..., `<h6>...</h6>`
- **Paragraph**: `<p>`
- **Horizontal line**: `<hr>`
- **Embedded image**: `<img src="http://... alt="..."/>
- **Hypertext link**: `<a href="http://...">...</a>`
- **List**: `<ul><li>...</li></ul>`
- Comprehensive reference: http://www.htmlhelp.com/

XML

- eXtended Markup Language:
  - describe any kind of structured documents
  - Anybody can define their own XML document structure by writing a DTD file:
    ```xml
    <!ELEMENT Shopping-List (Item)+>
    <!ELEMENT Item (#PCDATA)>
    ```
  - now you can write a shopping list document with your newly-defined tags:
    ```xml
    <?xml version="1.0"?>
    <!DOCTYPE Shopping-List SYSTEM "shoplist.dtd">
    <Shopping-List>
    <Item>Chocolate</Item>
    <Item>Milk</Item>
    </Shopping-List>
    ```

Web Services

- Web applications: generate dynamic web pages, etc.
- Web services: meant mainly for application to application communication (as opposed to users directly)
  - enables Business-to-Business transactions
  - Example: one web server contacts another one via a URL using the SOAP protocol over HTTP.

Examples

- A stock quote service.
  - an application requires the current value of a stock, the web service returns it.
- A route finder for delivery of goods.
  - given an initial and a final location, find the most cost-effective delivery route.
- A weather service, a map service, a web search service...
  - any composition of Web services.
Web Services

Web service is an RPC-like interface to a Web application:
- examples: Google, Amazon, ...
- you can write programs which automatically order books on Amazon
- Reminder: to build an RPC system you need:
  - communication protocol
  - standard message format
  - implementations of clients and servers
- Web services are built using Web technologies:
  - communication protocol: HTTP
  - message format: XML, etc.

eXtensible Markup Language

- Markup Language.
  - HTML is also a markup language (but it's not extensible!).
  - XML allows you to make up your own tags.
- Document Type Definition
- XML Schema

Example

```xml
<?xml version="1.0"?>
<PUBLICATION>
  <TITLE>Why I am Overworked</TITLE>
  <AUTHOR role="author">
    <FIRSTNAME>Fred</FIRSTNAME>
    <LASTNAME>Smith</LASTNAME>
    <COMPANY>Jones and Associates</COMPANY>
  </AUTHOR>
  <ABSTRACT>This is the abstract</ABSTRACT>
</PUBLICATION>
```

XML DTD Example

```xml
<?xml version="1.0"?>
<!DOCTYPE PUBLICATION [
<!ELEMENT PUBLICATION (TITLE, AUTHOR+, ABSTRACT*)>
<!ELEMENT AUTHOR (FIRSTNAME, LASTNAME, (UNIVERSITY | COMPANY)?)>
<!ATTLIST AUTHOR role (author|techwriter) "author">
<!ELEMENT FIRSTNAME (#PCDATA)>
<!ELEMENT LASTNAME (#PCDATA)>
<!ELEMENT UNIVERSITY (#PCDATA)>
<!ELEMENT COMPANY (#PCDATA)>
<!ELEMENT ABSTRACT (#PCDATA)>
]>
```

XML DTD

```xml
<!DOCTYPE root-element [doctype-declaration...]>  
  name of the root element, document type declarations  
<!ELEMENT element-name content-model>  
  associate content-model to given name:  
  EMPTY: no content allowed
  ANY: any content allowed
  #PCDATA: character data
  (........): sequence
  ...? optional
  ...* zero or more
  ...+ one or more
```
### XML-DTD

- `<ATTLIST element-name attr-name attr-type attr-default ...>`
  - allowed or required attributes
  - types:
    - `CDATA`: any value (default)
    - `(value...)`: enumeration of allowed values
  - attribute defaults:
    - `#REQUIRED`: must be provided
    - `#IMPLIED`: optional, no default
    - `"value"`: default
    - `#FIXED`: like "value" but only this value allowed

### XML Schema

```xml
<card xmlns="http://businesscard.org">
  <name>John Doe</name>
  <title>CEO, Widget Inc.</title>
  <email>john.doe@widget.com</email>
  <phone>(202) 456-1313</phone>
  <logo url="widget.gif"/>
</card>
```

### XML

- DTD, XML-Data, DCD (Document Content Description), DDML (Document Definition Markup Language), SOX (Schema for Object-Oriented XML), XML-Schema, Assertion Grammars, Schematron, TREX, RELAX, DSD (Document Structure Description), ...

### What Makes XML Portable?

- A schema (DTD) is associated with a document which allows to perform validation on the document.
- Human-readable/writable.
- Independent of presentation (formatting).

### Syntactic vs Semantic Interoperability

- While XML is portable, communicating parties still need to agree on:
  - Document type definitions
  - Meaning of tags
  - “Operations” on data (interfaces).
  - Meaning of those operations.
- Semantic interoperability is still a problem!
Why XML?

- XML generation is simple.
- XML parsing is also pretty simple.
  - there are lots of parsers available!
- Exact description but users can read it too.
- Validation (values are ok, defaults are inserted, etc.)
- Browsers understand XML (somewhat).
  - CSS style sheets
  - XSL: eXtensible Stylesheet Language
- XML can be used for document storage and transfer.

XML-RPC

- Use XML to encode requests.
  - procedure name
  - parameter values
- Response is also an XML document.
  - return value(s)
  - errors (faults)
- Both are well defined document types.
  - tag names are defined in the XML-RPC specification document.

HTTP POST

- Use existing protocol (and software!).
- Avoid firewall issues (everyone allows HTTP traffic).
- XML-RPC Request is the body of an HTTP POST.
- XML-RPC Response is the body (content) of the HTTP response.

Example Request

```
POST /RPC2 HTTP/1.0
Host: betty.userland.com
User-Agent: Frontier/5.1.2 (WinNT)
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param><value><i4>41</i4></value></param>
  </params>
</methodCall>
```

Example Response

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param><value><string>South Dakota</string></value></param>
  </params>
</methodResponse>
```

Data Types

- <i> or <i4>
- <boolean>
- <string>
- <double>
- <dateTime.iso8601>
- <base64>
- <struct>
- <array>
XML-RPC Struct

```xml
<struct>
  <member>
    <name>Hostname</name>
    <value>
      <string>monte.cs.rpi.edu</string>
    </value>
  </member>
  <member>
    <name>IPAddress</name>
    <value>
      <string>128.213.7.32</string>
    </value>
  </member>
</struct>
```

XML-RPC Array

```xml
<array>
  <data>
    <value><i4>12</i4></value>
    <value><string>Egypt</string></value>
    <value><boolean>0</boolean></value>
    <value><i4>-31</i4></value>
  </data>
</array>
```

XML-RPC Programming

- Need to be able to generate HTTP requests (client) and responses (server).
- Need to generate XML documents.
- Need to parse XML documents and extract specific items.
- Need to handle faults (errors).

Faults (Exceptions)

```xml
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><int>23</int></value>
        </member>
        <member>
          <name>faultString</name>
          <value><string>Unknown stock symbol XYXY</string></value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```