Lecture 10: Internet Applications II

CS178: Programming Parallel and Distributed Systems

February 28, 2001
Steven P. Reiss

I. Overview

A. Last time we talked about architectures for web applications

1. Client was the web browser

a) HTML with forms

b) HTML with Javascript (and forms)

c) Applet

d) Plugins

2. Server was a server as we had done before

a) Controls one or more resources; allows clients access

b) Socket connections from the local network

c) Multiple threads, multiple processes, etc.

3. Between these we had the web server

a) Talks to client via internet and HTTP

b) Talks to server via sockets

c) Code for the application inserted into the server

(1) Using cgi-bin applications

(2) Using servlets

(3) Using jsp

(4) Also was available -- server side javascript

B. This time I want to continue these discussions

1. With a look at data formats

2. With a look at where things might be headed

3. With a look at web components

4. With an example application and its tradeoffs and options

II. Data Formats -- an alphabet soup

A. We’ve talked a lot about data being passed back and forth between applications

1. Because we don’t control these applications we need to understand the data formats

2. Need to have some agreement on data formats

a) Agreement that works across many platforms, system versions, etc.

b) Agreement that works for arbitrary web browsers and servers

B. HTTP

1. This is a protocol for asking for and receiving web documents

a) Consists of header followed by body

b) There are commands and replies

2. Header consists of text lines argument: value

a) Contents/length of body defined by the header

3. Commands

a) GET -- get a file from the server

b) HEAD -- get a file (without any content) from the server

c) POST -- invokes a program on a form

d) PUT -- store some resource

e) OPTIONS -- find out about communications options

4. Command example:

```
GET /people/spr/head.html HTTP/1.1
User-Agent: Netscape/4.76
Connection: Keep-Alive
Host: www.cs.brown.edu
Accept: text/html
```

a) Any data (e.g. on a POST) would follow header

5. Reply example

```
HTTP/1.1 200 OK
Date: Thu, 22 July 2002 18:43:54 GMT
Server: Apache 1.3.5 (Unix) PHP/3.0.6
Last-Modified: Mon, 19 July 2001 16:05;33 GMT
Content-Type: text/html
Content-Length: 12987
```

. . .

a) Status codes -- 3 digit (200 ok; 300 iffy; 400, 500 -- fail)

b) Content type -- any mime type is possible

c) Blank line at end of header

C. HTML

1. This is the stuff of which web pages are made

2. Essentially marked up text

a) <xxx> .... </xxx> indicate sections

b) Not everything needs sections

3. Using tags to indicate sections

4. Capabilities

a) All types of formatting

b) Tables -- useful for any kind of layout

c) Forms

d) Layers and frames

5. CSS -- style sheets

a) Allow easier specification of formats

b) Allow more detailed specification of formats

D. XML

1. There has to be a standard way of shipping data between applications

a) Arbitrarily complex data

b) Data should be self-describing

c) There shouldn’t be an plethora of different languages

2. XML was defined as a standard for data representation

3. Basic idea is an hierarchical tree of elements

a) Each element has a name

b) Each element has a set of attributes (NAME=’VALUE’)

c) Each element has a set of nodes

d) Nodes can be either other elements or text

e) Typically one or the other, but both is allowed

4. Example -- describe a calculation

a) Note that everything must be ended explicitly

b) Note short form for simple elements

c) Note possibility of namespaces

5. XML is designed to be self-describing

a) Early versions: DDL to describe the structure either in the document or ref’d by URL

b) Now they use XSD which is an XML-based representation to define what goes in the document

```
<xsd:schema>
   <xsd:complexType name=”person”>
      <xsd:sequence>
         <xwd:element name=”name” type=”xsd:string”/>
         <xwd:element name=”age” type=”xwd:double” />
      </xsd:sequence>
   </xsd:complexType>
   <xsd:element name=”student” type=”person” />
</xsd:schema>
```

c) These describe

(1) What subcomponents a node can have

(2) The order and arity of these subcomponents

(3) What attributes an element can have

(4) The data type of these attributes

(5) Whether attributes are required or optional

(6) Default values for attributes

6. XML is very useful in its own right

a) With or without data schemas

b) Because there are parsers available -- save writing your own language

c) Because the result is readable (for debugging)

d) Because the result is portable

e) Because the result is easy to generat

7. A whole range of things has grown up around XML

a) XPointer -- the ability to have pointers within an XML document and to other documents

b) XQuery -- a query language for treating XML documents as databases

c) XSL -- a programming language for transforming XML documents either into other XML documents or into formatted text

8. A whole range of XML languages has also grown up

a) XUI -- user interface definition language

b) XHTML -- XML version of html

c) Result of MS Office; Frame MIF files; ...

III. Remote Method Invocation over the web

A. Recall how we got to Java RMI in distrib. apps

1. We started with sockets as a communications medium

2. We went to messages to get discrete units

3. But we wanted something that was more language-oriented which meant procedure calls

4. This led first to remote procedure calls for C

5. Then to remote object invocation for arbitrary languages using an IDL

6. Then to remote object invocation using language facilities in Java RMI

B. One can do the same thing over the web to provide client-server communication

1. Need to be able to send messages

a) This we can already do via HTTP; messages can be forwarded easily by webserver to app server

2. Need to have a way of encoding the calls

3. Need to have a way of encoding types, etc.

4. Need to have a registry

C. Encoding calls is done using SOAP

1. This is another XML-based language

```
<soapv:Envelope>
   <soapv:Body>
      <Order>
         <customer>99211-33b</customer>
         <partno>8881-4ah</partno>
      </Order>
```

2. **Used for wrapping up objects and object invocations**
3. **Essentially an XML-based implementation of serialization along with some messaging**
4. **Typically would be done automatically by stub for remote object**

D. Type encoding is done using WSDL

1. **Web service description language**
2. **Another XML-based language**
3. **Lets user define the services provided by a web object**
    - a) These correspond to methods
    - b) Thus this is an XML definition of a remote interface

E. Registry is done as a known web address that handles known services

F. This framework is beginning to be available

1. **.Net (C#) can generate SOAP stubs/skeletons as well as WSDL interfaces from base code definition**
2. **Java web services is moving to make this functionality available in Java (IBM web sphere already does)**

G. Advantages of this approach

1. **Easier model for programming**
2. **Allows more straightforward coding of clients**
3. **Same model can be used between servers as client-server**
4. **Server can actually call the client -- push type web pages**

H. Disadvantages

1. **Performance expectations for remote method calls**
2. **Handling network failure**
3. **Clients aren’t reliably connected to server**

I. Pronostications

1. **Will this succeed?**

IV. Web Components

A. Using standard components is a plus

1. **Web browser as client standardizes UI**
2. **Web server as middleware standardizes communication, etc.**

B. Web components implies that a lot of other things can be standardized and built up from standard existing pieces

1. **Example: authentication**
   - a) A lot of sites need to allow you to register/login
   - b) Then there is information associated with the particular user that should be known to the server in constructing appropriate responses to the client
   - c) Maintaining users, passwords, associated information, etc. can be messy
     - (1) Especially if security or privacy are issues
   - d) But one can have a standard component that does this
     - (1) Component hooked to web server; user commands get directed there as appropriate
     - (2) Either information from there is then included in messages to server or server can access the component directly
   - e) Can even do network-wide authentication
     - (1) One authentication server for multiple sites
     - (2) Works the same way, just does calls to it across the web
     - (3) This is what MS Passport attempts to be
2. **Lots of other components are possible**
   - a) Low-level -- billing, video streaming, audio streaming, ...
   - b) High-level -- complete e-store services (account management, inventory, shipping, ...)

C. Easier to build then buy

1. **Provided they are designed & chosen appropriately**

V. A Case Study

A. Purpose

1. **I'd like to look at a problem that requires a web implementation**
2. **Show what steps go into the design of such an application**
3. **Show what decisions need to be made in its design**

B. Problem definition

1. **We want to define a front end to Internet search**
   - a) Takes a search query from user
   - b) If its ambiguous, asks user to disambiguate
   - c) Expands query with additional words
   - d) Sends new query to multiple search engines
   - e) Merges the result
2. **The server controls a semantic database**
   - a) Given a word, return set of meanings
   - b) Given a meaning, return a set of associated words
3. **User front end should be easy to use, fast, etc.**
   - a) Should work over the web on most browsers
4. **Back end should scale to thousands of users**
5. **Back end might track info about a user**
   - a) Allow personal definitions of terms
   - b) Allow disambiguation based on past experience

C. Next time we will look into how this can be built