ΕΠΛ 602:Foundations of Internet Technologies

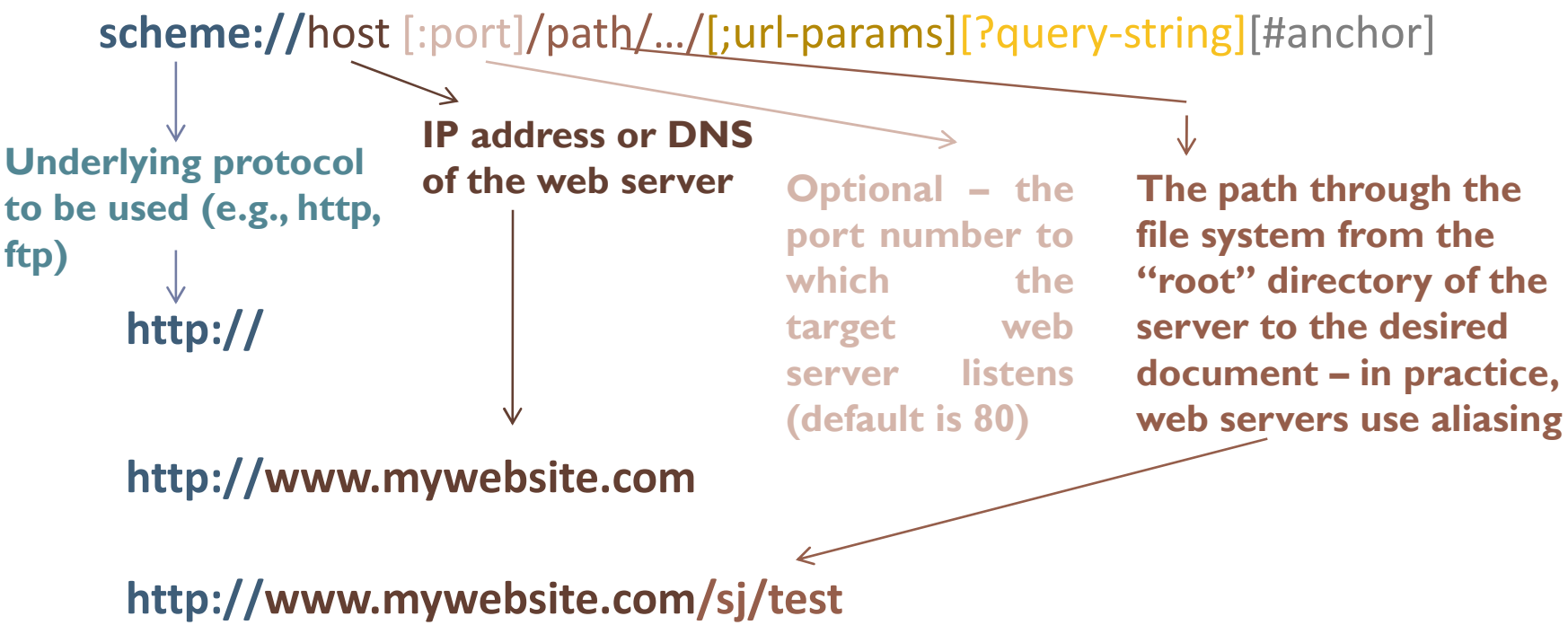
Approaches to Web Application Development

HTTP

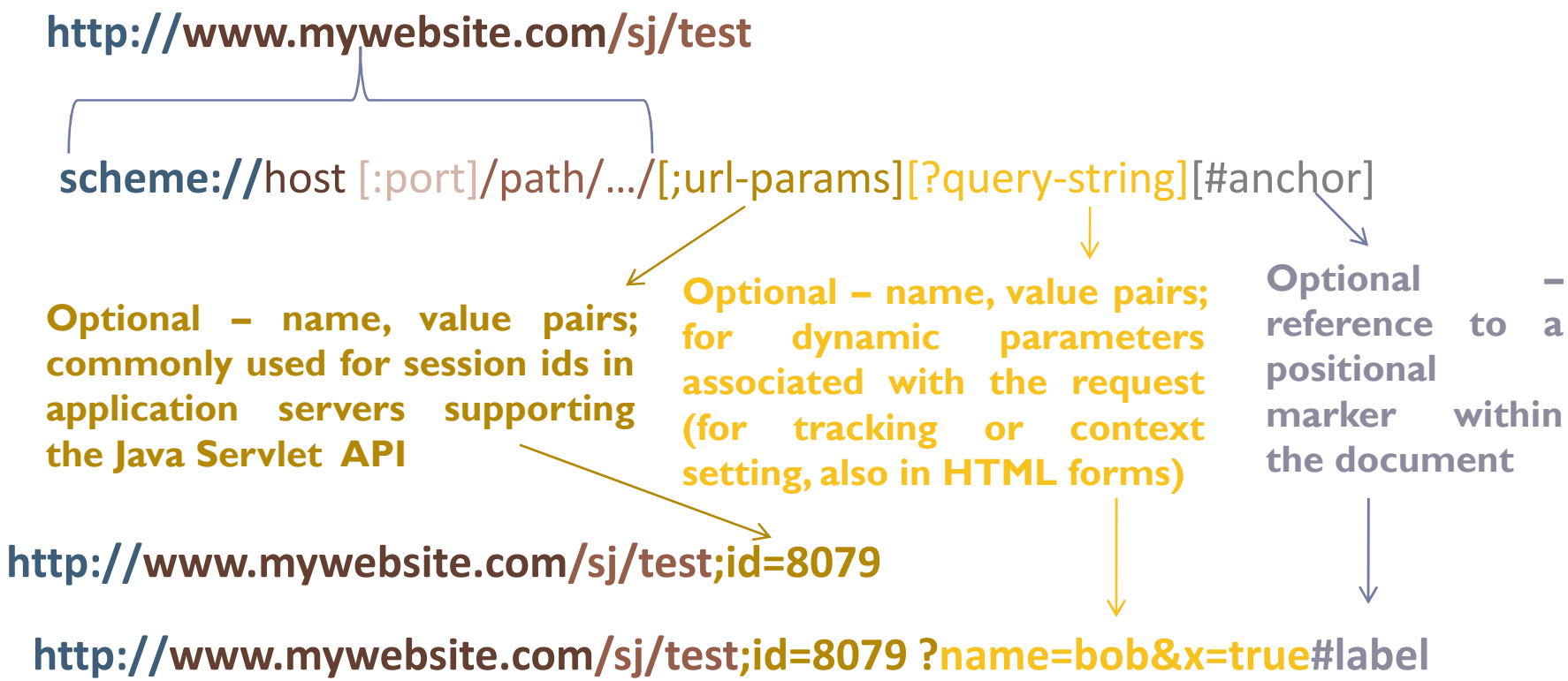
"Web Application Architecture" book, chapter 3

http://www.webappbuilders.com/

Uniform Resource Locator (URL)



Uniform Resource Locator (URL)



application-protocol://IP-address[:port]path-from-the-root[;par][?dyn-par][#anchor]

HTTP

- Application-level protocol in the TCP/IP protocol suite
- Uses TCP
- Client-Server model
- Follows the request-response communication paradigm
- Stateless (HTTP transaction: single request, followed by a single reply)
  - vs stateful: sequences of related commands are treated as a single interaction, often called a *session*
  - session are within a persistent connection

  (more later)
- Through Proxies
  - Firewalls
  - Support for caching
  - Filtering
- Connection defined as a virtual circuit (browser, server, proxies)

HTTP message

Simple example request

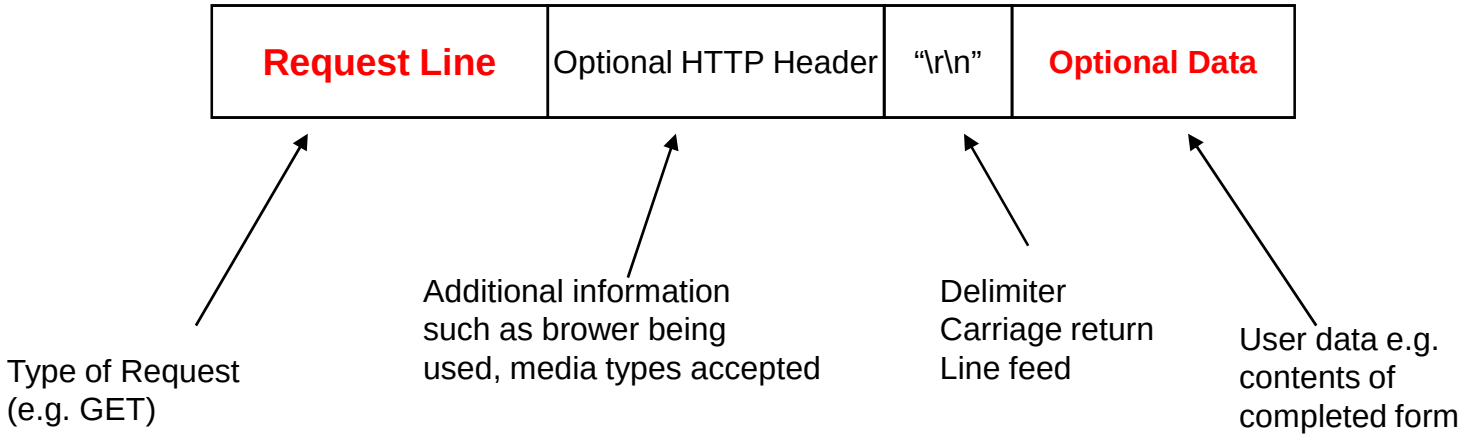
```
Method /path-to-resource  HTTP/version-number
Header-Name-1: value
Header-Name-2: value

[optional request body]
```

```
GET /sj/index.html  HTTP/1.1
Host: www.mywebsite.com
```

HTTP message

HTTP Request messages are sent from client to server.



- There are a number of valid HTTP Request messages
  - **Get** – Used to request a web page from a web server
  - **Post** – Used to send data (e.g. results of registration form) to a web server
  - **Head** – Return the header of a web page, used by search engines to test the validity of hyperlinks
  - **Put / Delete** – Not typically implemented by browsers.

HTTP message

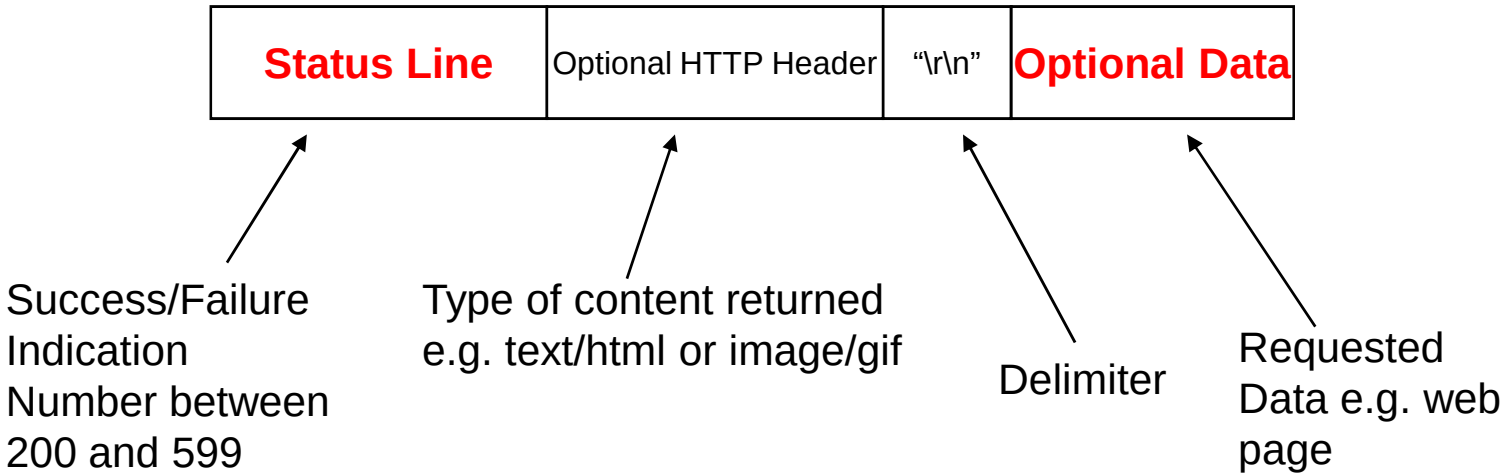
Simple example response

```
HTTP/version-number status-code explanation
Header-Name-1: value
Header-Name-2: value

[response body]
```

HTTP message

HTTP Request messages are sent from server to client.



- The Status Line gives information about the success of the previous HTTP Request
  - 200 – 299 Success
  - 300 – 399 Redirection – Document has been moved
  - 400 – 499 Client Error – Bad Request, Unauthorised, Not found
  - 500 – 599 Server Error – Internal Error, Service Overloaded

Request Methods

GET HEAD POST
PUT DELETE TRACE OPTIONS CONNECT

```
Method /path-to-resource  HTTP/version-number
Header-Name-1: value
Header-Name-2: value

[optional request body]
```

GET

Most common (type a URL, click on a link, etc), if the URL: refers to data, the web server replies by returning the data, if it refers to a program, then the web server runs the program and returns its output

POST

POST has a body where the URL parameters are placed, GET appends them to the path

Web application dependent: e.g., display a form when GET request and process it when POST

Request Methods

```
Method /path-to-resource  HTTP/version-number
Header-Name-1: value
Header-Name-2: value

[optional request body]
```

```
GET /q?s=YHOO HTTP/1.1
Host: filename.yahoo.com
User-Agent: Mozilla/5.0 (Windows; U; Windows XP; en-US; rv:1.8.0.1)
```

```
POST /q HTTP/1.1
Host: filename.yahoo.com
User-Agent: Mozilla/5.0 [en] (WinNT; U)
Content-Type: application/x-www-form-urlencoded
Content-Length: 6

s = YHOO
```

Request Methods

POST VS GET

- Requests using GET should only retrieve data and should have **no other effect**.
  POST may result in the creation of a new resource, the updates of existing resources or both

- POST submits data to be processed. The data is included in the body of the request.

Request Methods

HEAD

Requests that use the HEAD method are processed similarly to requests that use the GET method but the server sends back *only headers* (not the body) in the response

Used to support caching

Still useful for implementing change-tracking systems, testing and debugging new applications and discovering server capabilities

Request Methods

Safe methods

(for example, HEAD, GET, OPTIONS and TRACE) intended only for information retrieval and should not change the state of the server.

No side effects, except e.g., as logging, caching, the serving of banner advertisements or incrementing a web counter.

Idempotent operations

multiple identical requests should have the same effect as a single request.

POST not necessarily idempotent: sending an identical POST request multiple times may further affect state or cause further side effects

Some cases e.g., a user does not realize that their action will result in sending another request, or they did not receive adequate feedback that their first request was successful.

Web browsers **alert dialog boxes** to warn users when reloading a page may re-submit a POST request, but up to the web application to handle cases where a POST request should not be submitted more than once.

whether a method is idempotent is not enforced by the protocol or web server.

Request Methods

Multiple References Generated by One Page

HTML pages may contain references to other accessible resources

- Graphical images
- Java applets

Web browsers must parse the retrieved HTML page to see what additional resources are needed

Browser must send HTTP requests to retrieve additional resources

Status Codes

1 Informational

100 (notify clients that they may continue) *in reply to Expect:100-continue header*

2 Successful responses

200 201 (message was satisfied and a new resource was created)

3 Tell the client to perform additional actions (redirection)

4 Client requests errors or special conditions

400 Bad Request 401 Not Authorized 403 Forbidden 404 Not found

5 Server errors

500 Internal Server Error 501 Not Implemented

Status Codes: Redirection (3xx)

❖ Redirection: the browser is instructed to resubmit the request to the URL specified in the Location header

- 301 moved permanently
- 302 temporarily

❖ Browsers respond "silently" to redirection status codes

❖ (not supported or disabled) Web servers include a message body that explicitly references a link to the new location -> follow the link manually

❖ Web servers treat *a URL ending in a slash* as a request for a directory (depending on server configuration return either a file with a default name (e.g., index.html) or the contents of the directory)

❖ If the user forgets the trailing "/", the server a redirection response

❖ Proxies react to 301 status by updating internal relocation tables (*cache 301 redirections*) e.g., redirecting users to the login page when trying to access a protected URL

Headers

General Headers

Apply to both request and response messages

Do not describe the body of the message

Example:

Date (time and date of the message creation),

Connection (indicates whether the client or server intends to keep the connection alive - keep-alive default setting for HTTP/1.1)

Request Headers

Allows clients to pass additional information

Example

User-Agent (type of software)

Host (virtual hosting)

Referee (context information about the request, e.g., if because of a click on a link in a page, the header is set to the URL of that page)

Authorization Browsers include this header in all follow-up requests [after being notified of an authorization challenge (401) and prompting the user for credentials, once credential accepted included (expiration is browser-specific)

Headers

Response Headers

Help the server to pass additional information about the response that cannot be inferred from the status code

Examples

Location for redirecting (used with 301, 302)

WWW-Authenticate (used with 401) Basic realm = "KremlinFiles", if browser, users are prompted for credentials

Realm: which resources require what type of authorization – web masters can administrate web servers to define realms, associate them with files and directories and establish userid and passwords that limit access to these resources

Server server software

Entity Headers

Either message bodies or (in the case of no body) target resources

Examples

Content-Type the MIME type of the message body

Content-Length to help the browser in rendering

Last-modified critical for caching

Support for content types

HTTP borrows its content typing system from Multipurpose Internet Mail Extensions (MIME)

A two layer ordered encoding model
Content-Encoding (gzip, compress, deflate)

Content type

type */* subtype [";" parameter-string]

Examples

Content type: text/html
Content type: textplain; charset='us-ascii'
Content type: application/pdf
Content type: video/quicktime

Browsers use types and sub-types either to select a proper content-rendering module or to invoke a third-party tool
Server-side applications use type information to process requests

Support for content types

Multipart message

A MIME multipart message contains a boundary in the "Content-Type: " header; this boundary, is placed between the parts, and at the beginning and end of the body of the message

```
Content-Type: multipart/mixed; boundary="frontier"

This is a message with multiple parts in MIME format.
--frontier
Content-Type: text/plain

This is the body of the message.
--frontier
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64

PGh0bWw+CiAgPGhlYWQ+CiAgPC9oZWFkPgogIDxib2R5PgogICAgPHA+VGhpcyBpcyB0aGUg
Ym9keSBvZiB0aGUgbWVzc2FnZS48L3A+CiAgPC9ib2R5Pgo8L2h0bWw+Cg==
--frontier–
```

o Each part consists of its own content header (zero or more *Content-* header fields) and a body.
oThe sending client must *choose a boundary string* that doesn't clash with the body text. Typically this is done by inserting a long random string.

Support for content types

Multipart message

The content type multipart/x-mixed-replace developed as part of a technology to emulate server push and streaming over HTTP.
All parts of a mixed-replace message have the same semantic meaning. However, each part invalidates - "replaces" - the previous parts as soon as it is received completely.
Clients should process the individual parts as soon as they arrive and should not wait for the whole message to finish.

```
Content-Type: multipart/x-mixed-replaced; boundary=ThisRandomString
Connection: close

--ThisRandomString
Content-Type: image/gif

...

--ThisRandomString
Content-Type: image/gif
...
```

Caching

A set of mechanisms allowing responses to HTTP requests to be held in temporary storage as a means of improving server performance
Future requests are satisfied from the temporary store, eliminating the overhead of asking the server for a fresh copy

- Browser-side
- Proxy-side
- *Server-side*

Caching decisions guided by information provided by the server

HTTP1.1 provides a mechanism for enforcing caching rules based on the Cache-Content header
- **public** setting authorizes both shared and user-localized caching
- **private** setting indicates that the response is directed to a single user and should not be stored in a shared cache (e.g., a secure request about their private accounts)
- **no-cache** setting indicates that neither browser nor proxies are allowed to cache, but there are options (cache but exclude specific headers)

HTTP1.0 browsers and proxies are not guarantee to obey such instructions

Caching

When to refresh?

HTTP/1.0
HEAD and then GET

HTTP/1.1
New Headers: If-Modified-Since: (If-Unmodied-Since)
304 Not Modified or the body

Security

- authentication (verify user identity) vs
- authorization (check whether access to a specific resource)

Built-in support for basic authentication:
where user credentials (userid and password) are transmitted via the Authorization header as a single encoded (not encrypted) string
Safe only if performed over a secure connection (e.g., https)

Built-in (basic):
Server replies with 401 (+realm)
pop-up menu
Client resubmits with the Authorization header
If ok, server sends content, browser uses then in future requests
If not, after attempts, sends 403 Forbidden

Many web applications implement their own authentication and authorization schemes
Use body of POST/ don't use 401 but may use 403

Session support

Session ids
Server Set-Cookie
Client Cookie

Server applications can use the Set-Cookie header

```
Set-Cookie:  <name>=<value>
[; Comment=<value>]  [; Max-Age=<value>]
[; Expires=<date>] [; Path =<path>]
[; Domain=<domain name>] [; Secure]
[; Version=<version>]
```

An attribute-value pair <name> = <value> is sent back by the browser in qualifying subsequent requests

Max-Age the lifetime of the cookie in secs (Expires)

Session support

Set-Cookie: <name>=<value>
[; Comment=<value>] [; Max-Age=<value>]
[; Expires=<date>] [; Path =<path>]
[; Domain=<domain name>] [; Secure]
[; Version=<version>]

The Path and Domain attributes delimit which request qualify, by specifying the server domains and URL paths to which this cookie applies

- Domains: suffixes of the originating server's host name
- Path attribute default to the path of the URL associated with the server application

For subsequent requests directed at URLs where the domain and path match, the browser must include a Cookie header with the appropriate attribute-value pair

Secure tells the browser to submit corresponding Cookie headers over secure connections -- Version

Session support

Set-Cookie2: <name>=<value>
…
~~[; Expires=<date>]~~ [; Path =<path>]
[; Domain=<domain name>] [; Port=<portlist>]
…

Cookie-Jars in browsers (in memory (current browser session) or persistent (for cookies with defined lifetimes)

Persistent connection

For performance *allow* connections to persist across multiple requests, but we should *not depend on* persistent connections for application logic

HTTP/1.0 Connection: keep alive

HTTP/1.1 connections are persistent, except when explicitly closed by a participating program via the Connection: close header

Pipelining: browsers can queue requests messages without waiting for responses
Servers are responsible for submitting responses to browsers in the order of their arrival

Virtual Hosting

Virtual hosting: map multiple host names to a single IP address

HTTP1.1 uses Host header

HTTP

- using TCP increase reliability and also cost
- HTTP uses TCP
  - one connection per request-reply
  - HTTP 1.1 uses "persistent connection"
    - multiple request-reply
    - closed by the server or client at any time
    - closed by the server after timeout on idle time
  - Marshal messages into ASCII text strings
  - resources are tagged with MIME (Multipurpose Internet Mail Extensions) types: test/plain, image/gif...
  - content-encoding specifies compression alg

HTTP methods

- GET: return the file, results of a cgi program, …
- HEAD: same as GET, but no data returned, modification time, size are returned
- POST: transmit data from client to the program at url
- PUT: store (replace) data at url
- DELETE: delete resource at url
- OPTIONS: server provides a list of valid methods
- TRACE: server sends back the request

Web Application Development

"Web Application Architecture" book, chapter 9

Lecture Outline

- Taxonomy of web applications and frameworks
- Comparative survey of approaches and frameworks

From Web pages to Web applications

Dynamic web
From building a web site -> design a web application

Web application: a client/server application that uses a web browser as its client program

Delivers interactive services through web servers distributed over the Internet or an intranet

A web application can present dynamically tailored content based on request parameters, traced user behaviors and security consideration

Example: online shopping cart

Introduction

Motivation

Neither practical nor desirable to develop every new web application from scratch

Common Application Functionality

- Accept user requests
- Interpret user requests
- Authenticate requestors
- Authorize requestors
- Access data
- Transform data
- Construct responses
- Transmit responses

Web servers provide clear endpoints:
- Acceptance of requests
- Transmission of responses

Approaches vs Frameworks

Approach

- Library of functional components
- Can be re-used across applications
- Usually, around a programming language + web-specific APIs and packages

Framework

- Consistent infrastructure with rich services
- Usually, integrated support for database access, authentication and state or session management

Separate Content from Presentation

Developers (business logic and access to content)
Designers (page format)

Taxonomy

Categories

Categories of Web Application Approaches

- Scripting or programmatic approaches
- Template approaches
- Hybrid approaches
- Frameworks

Categories: Programming

Scripting or programmatic approaches

Code-centric

The source associate with a page object consists basically from code in a scripting (e.g., Perl, Pyhton, Tcl) or programming language (e.g., Java)

Embedded formatting instructions -> commonly produced using output statements written in the associated language

Examples: CGI, Java Servlets

Programmer needs to translate designer intention into code and integrate it into the script or program

Categories: Template

Template approaches

uses a source object (the template) with formatting structures and limited embedded constructs for programming

Focus on formatting not programming logic

Page-centric
Around the page structure and formatting tags

Source objects: page templates (mostly HTML + embedded constructs for conditional processing, iterative result presentation and parameter substitution)

Examples: SSI (Server Side Includes), Adobe's Cold Fusion, Apache's WebMacro/Velocity)

Categories: Hybrid

Hybrid approaches

combine scripting elements with template structures

Allow embedded blocks containing "scripts"
Most translate hybrid source objects into code (+some form of pre-compilation)

Examples: PHP, Active Server Pages (ASP) => .NET, Java Server Pages (JSP)

Frameworks

Web Application Frameworks

- Provide a consistent architecture for building and accessing request context elements that can be embedded within the web
- Support state and session management and authentication
- Support for data access and transformation
- Separates content from presentation
- Patterns support frameworks

Frameworks

Separate CONTENT from PRESENTATION

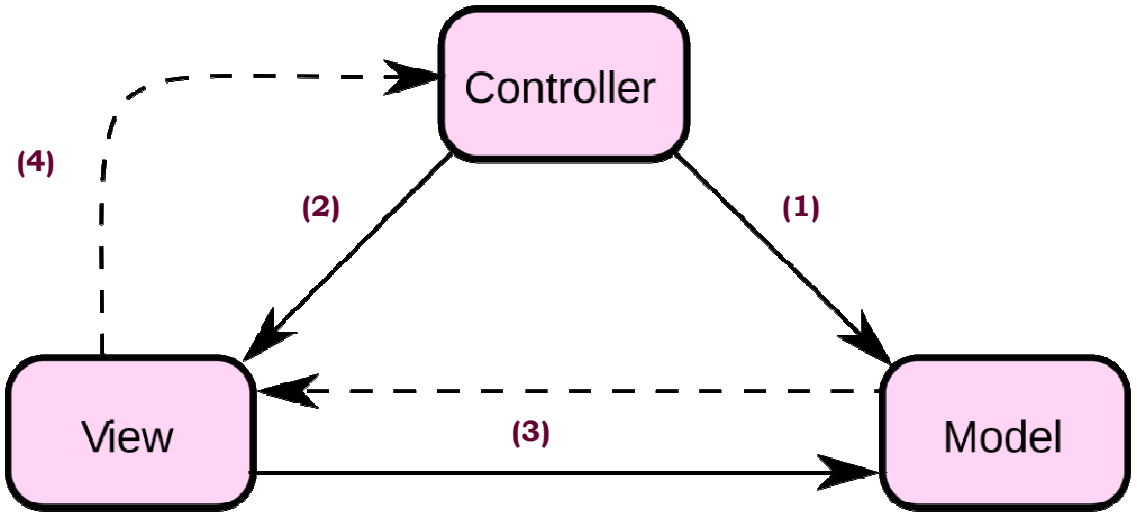
- The Model: modules responsible for producing content
- The View: modules responsible for presenting content in a particular format (organization and layout)

Map/territory analogy

Data model should be usable by a variety of views (presentation formats)

Some controlling mechanism should be the glue that hooks up retrieved content with the presentation format

The Model-View-Controller (MVC) pattern



Controller receives a user request
(1) Controller constructs the Model that fulfills this request
(2) Controller selects a view to present the results
(3) The View communicates with the model to determine its contents
The View presents the contents to the user in the desired format
(4) The View acts as the interface for transmitting further requests from the user to the Controller

The Model-View-Controller (MVC) pattern

- Facilitates separation of content from presentation
- Allows applications to dynamically tailor the view based on user preference, device capabilities and business rules

Developers vs. Designers

- Different skill sets
- Some tools favor the developer; some favor the designer

Designers (presentation experts, CSS, XML, XSLT, Dreamweaver, FrontPage) usability
Developers (content access and manipulation) scalability, maintenance, performance

Hybrid
Who owns and is responsible for a hybrid page object

Controller – developer
View – designer

The Model-View-Controller (MVC) pattern

- Scalability
- Configurability
- Separation of Roles

Too complex

New generation of
Rapid Application Development (RAD)

Overview

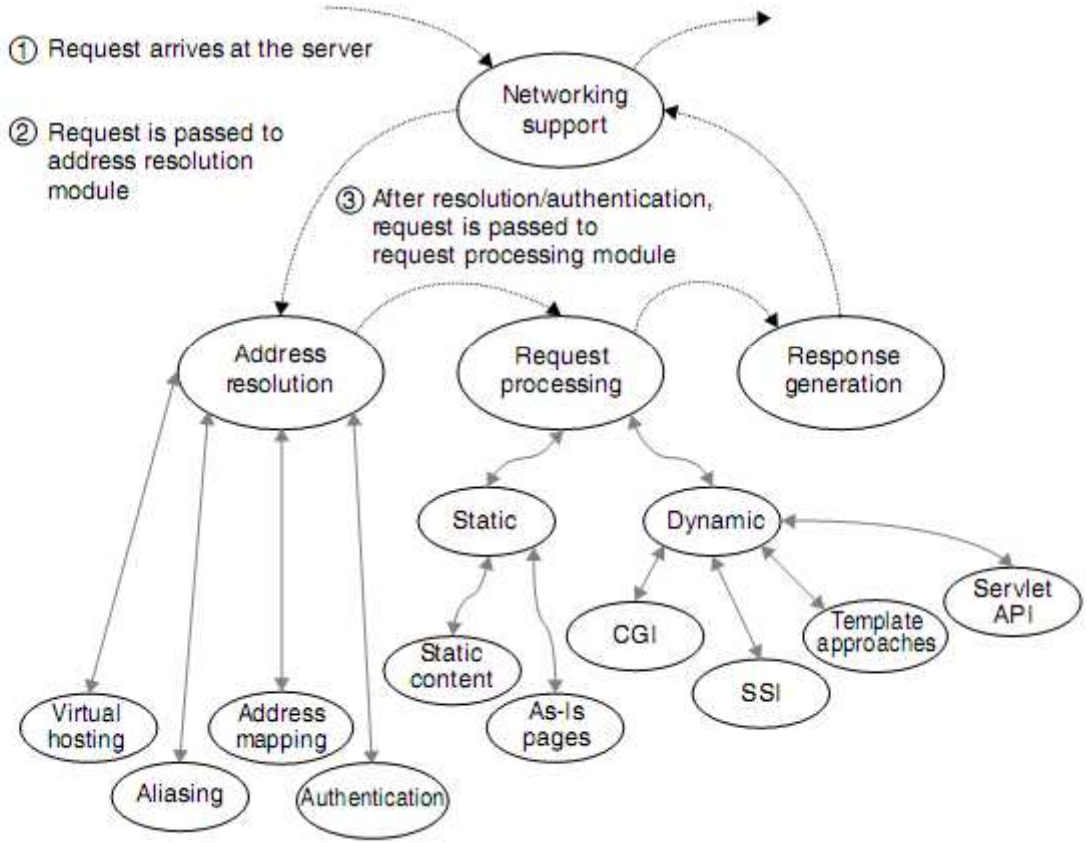
Web development

Server (configuring, implementing the server or components of the server)
Server applications (interacts and passes information to the server)
Server-side languages and frameworks

Web server operation

Address Resolution Preprocessing:

1. Virtual hosting: if the web server is providing service for multiple domains, determine the target domain
2. Address mapping: whether the request is for static or dynamic content and resolve the address to an actual location within the server file system
3. Authentication



Stateless: any information must be contained within the request

Web servers

Persistent connections:

Within a single open connection:

- A series of requests
- FIFO response delivery

Server maintains two queues:

Input:

Output: after processing, marked for release but remain, till all predecessors

Web servers

Static vs Dynamic

To determine how to process filename suffixes (extensions) and URL prefixes

Default: URL static content
Path beginning with /servlet or /cgi-bin/ and target .cgi -> Java servlet, CGI script
Target filename .php or cfm -> template processing

Web servers: static content

Static Content

- Static content page
  - Server maps the URL to a file location relative to the server document root (root_path/path_portion_of_URL)

Server

1. Retrieves the file
2. Constructs the response
3. Transmits it to the browser

Status code
Content-type: determines how the browser should render the body of the response (not the URL)

- As-is-page

Static files containing complete HTTP responses (including headers)
.asis file extension

```
HTTP/1.1 200 OK
Date: Tue, 29 May 2001 23:15:29 GMT
Last-Modified: Mon, 28 May 2001 15:11:01 GMT
Content-type: text/html
Content-length: 193
Server: Apache/1.2.5

<HTML>
<HEAD><TITLE>School Page</TITLE></HEAD>
<BODY>
<H2>My Links</H2>
<ul>
<li><a href ="classes.html">My classes</a></li>
<li><a href ="friends.html">My friends</a></li>
</ul>
</BODY>
</HTML>
```

Web servers: dynamic content

Variety of sources, such as search engines, databases, news feeds, etc
Dynamic content – server must take explicit programmatic action to generate a response

- execution of an application program
- inclusion of information from a secondary file
- interpretation of a template

Methodologies for Accessing Dynamic Data

- Common Gateway Interface (CGI)
- Template or hybrid languages (PHP, Cold Fusion, ASP, JSP)

Web servers: Features

Advanced Server Features

- Virtual hosting – ability to map multiple server and domain names with separate document trees and server-side applications to a single IP address

Physical configuration parameters (physical resources, such at listening ports, number of persistent connections, server processes, etc)
Logical configuration parameters (location and configuration of the document tree and server-side applications, etc)

- Chunked transfers – enables processing of partially transmitted messages

Transfer-Encoding: chunked header (recommended for slow connections)

Web servers: Features

Caching support

Server-side caching -> cache static pages only, caching of dynamic pages responsibility of the server applications

In terms of the protocol:

- Support If-Modified-Since and If-Unmodified-Since
- Include the Last-Modified header whenever possible
- The Data header must be included with every response

Web servers: Server Configuration

Directory structure

server root (HTTP server installation directory) – common subdirectories (document root, log directory, CGI and servlet root directories, configuration directory) but differs based on the situations (different servers sharing the same files)

Threads – some processes are kept running at all times to improve performance

Virtual hosts must be configured separately (not the physical resources though)

Address resolution – translate URL to file system pathname, choose processing module

MIME support – map MIME types and file extensions

Server extensions – add new MIME types and/or processing modules

Web servers: Security

- Minimize remote login to server
- at least: monitor and log all attempts to access the system
- Passwords should be crack-resistant
- Check file permissions on configuration and password files
- Disable SSI pages in user directories
- Separate FTP and HTTP directories
- Use HTTPS/SSL (encrypted messages) for secure messages, including passwords
- Use a firewall to isolate machines on a LAN – run an HTTP proxy on the firewall machine configured to screen HTTP requests

Web Application Development Approaches

Programming Approach

- CGI
- FastCGI
- Servlet API

Template Approach

- SSI, ColdFusion

Hybrid Approach

- PHP, JSP

Questions?