E-COMMERCE and SECURITY - 1DL018

Spring 2009

An introductury course on e-commerce systems

http://www.it.uu.se/edu/course/homepage/ehandel/vt09/

Kjell Orsborn
Uppsala Database Laboratory
Department of Information Technology, Uppsala University,
Uppsala, Sweden

UPPSALA UNIVERSITET

Web Servers

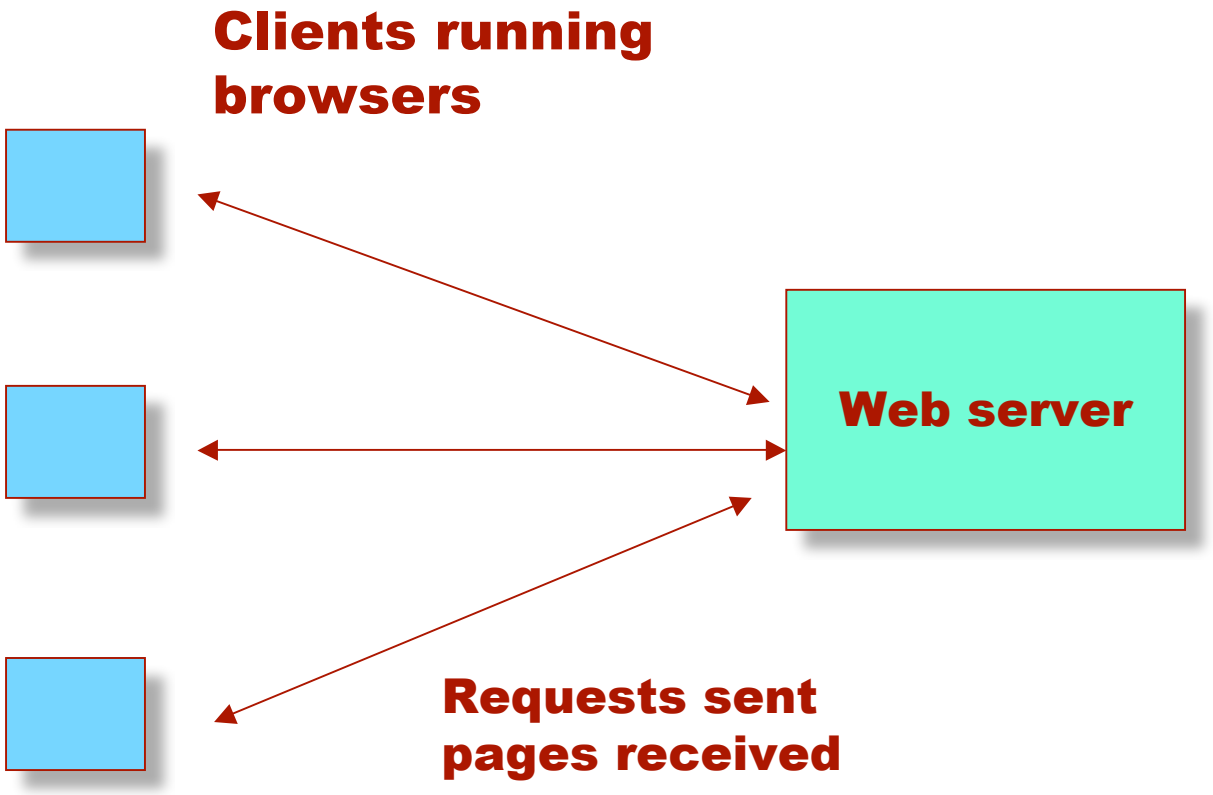
ch 6, 7

Kjell Orsborn

Department of Information Technology
Uppsala University, Uppsala, Sweden

UPPSALA
UNIVERSITET

Web clients and Web servers



Web server

- The term web server can mean one of two things:
  - A computer program that is responsible for accepting HTTP requests from clients, which are known as web browsers, and serving them HTTP responses along with optional data contents, which usually are web pages such as HTML documents and linked objects (images, etc.).
  - A computer that runs a computer program as described above.

Common features of Web servers

- Web server programs might differ in detail, but they all share some **basic common features**:
  - **HTTP**: every web server program operates by accepting HTTP requests from the client, and providing an HTTP response to the client.
    - The HTTP response usually consists of an HTML document, but can also be a raw file, an image, or some other type of document (defined by MIME-types).
  - **Logging**: usually web servers have also the capability of logging some detailed information, about client requests and server responses, to log files;
    - this allows the webmaster to collect statistics by running log analyzers on log files.
- In practice many web servers also implement other **features** including:
  - **Authentication**, optional authorization request (request of user name and password) before allowing access to some or all kind of resources.
  - Handling of **static content** (file content recorded in server's filesystem(s)) and **dynamic content** by supporting interfaces such as SSI, CGI, SCGI, FastCGI, JSP, PHP, ASP, ASP .NET, Server API (NSAPI, ISAPI, Apache API) etc.
  - **HTTPS** support (by SSL or TLS) to allow secure (encrypted) connections to the server on the standard port 443 instead of usual port 80.
  - **Content compression** (i.e. by gzip encoding) to reduce the size of the responses.
  - **Virtual hosting** to serve many web sites using one IP address.

Origin of returned content

- The **origin** of the **content** sent by server is called:
  - **static** if it comes from an existing file lying on a filesystem;
  - **dynamic** if it is dynamically generated by some other program or script or Application Programming Interface called by the web server.
- Serving **static** content is usually much faster (from 2 to 100 times) than serving **dynamic** content, especially if the latter involves data pulled from a database.

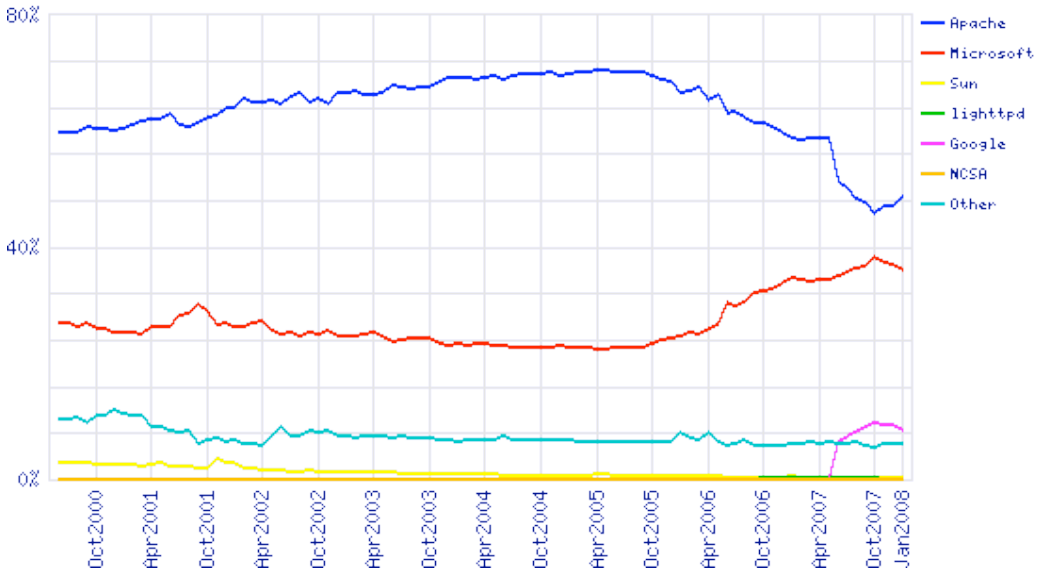
Path translation

- Web servers are able to map the path component of a Uniform Resource Locator (**URL**) into:
  - a local file system resource (for static requests);
  - an internal or external program name (for dynamic requests).
- For a *static requests*, the URL path specified by the client is relative to the Web server's root directory.
  - Consider the following URL as it would be requested by a client:
  - `http://www.example.com/path/file.html`
  - The client's web browser will translate it into a connection to `www.example.com` with the following HTTP 1.1 request:
  - ```
    GET /path/file.html HTTP/1.1
    Host: www.example.com
    ```
  - The web server on `www.example.com` will append the given path to the path of its root directory. On Unix machines, this is commonly `/var/www/htdocs`.
  - The result is the local file system resource:
  - `/var/www/htdocs/path/file.html`
- The web server will then read the file and send a response to the client's web browser.
  - The response will describe the content of the file and contain the file itself.

UPPSALA UNIVERSITET
    ```

Frequent web server (partial list)

- **Netcraft Web Server Survey**
  (http://survey.netcraft.com/Reports/200801/)



Vendor	Product	Web Sites
Apache	Apache	78,374,121
Microsoft	IIS	55,704,513
Google	GFE	8,287,294
Unknown	Unknown	3,326,340
Oversee	Oversee	1,603,182
lighttpd	lighttpd	1,536,981
Other	Other	1,033,668
nginx	nginx	842,206
Zeus	Zeus	504,680
IdeaWebServer	IdeaWebServer	454,427
LiteSpeed	LiteSpeed	432,442
Apache	Coyote	361,198
Sun	Sun-ONE-Web-Server	341,308
Resin	Resin	298,569
Jetty	Jetty	228,789
RapidSite	RapidSite	220,070
Sun	Enterprise	200,169

Web markup languages

- **HTML**: *HyperText Markup Language*, is the predominant markup language for web pages.
  - It provides a means to describe the structure of text-based information in a document by denoting certain text as links, headings, paragraphs, lists, etc.
  - Furthermore, text can be and supplemented with *interactive forms*, embedded *images*, and other objects.
  - HTML can to some degree describe the appearance and semantics of a document, and can include embedded scripting language code affecting the behavior of web browsers.
  - HTML is also used to refer to content of the MIME type text/html or even more broadly as a generic term for HTML whether in its XML-descended form (such as XHTML 1.0 and later) or its form descended directly from SGML (such as HTML 4.01 and earlier).
- **XML:** The *Extensible Markup Language* is a general-purpose markup language.
  - It is classified as an extensible language because it allows its users to define their own elements. Its primary purpose is to facilitate the sharing of structured data across different information systems, particularly via the Internet.
- **XHTML:** The *Extensible HyperText Markup Language*
  - It is a markup language with the same depth of expression as HTML but conforming to XML syntax.
- **DHTML**: *Dynamic HTML* is a collection of technologies used together to create interactive and animated web sites.
  - This include a static markup language (such as HTML), a client-side scripting language (such as JavaScript), a presentation definition language (Cascading Style Sheets, CSS), and the Document Object Model (DOM).

URI:s, URL:s & URN:s

- In computing, a **Uniform Resource Identifier (URI)**, is a compact string of characters used to identify or name a resource. The main purpose of this identification is to enable interaction with representations of the resource over a network, typically the World Wide Web, using specific protocols.
- A **URI** *may* be classified as a locator (URL) or a name (URN) or both.
- URI:s can refer to web pages, mail, newsgroups, FTP etc.
- An example of a URL:

http://www.it.uu.se/edu/course/homepage/ehandel/vt08/

Specifies Web page

Specifies computer

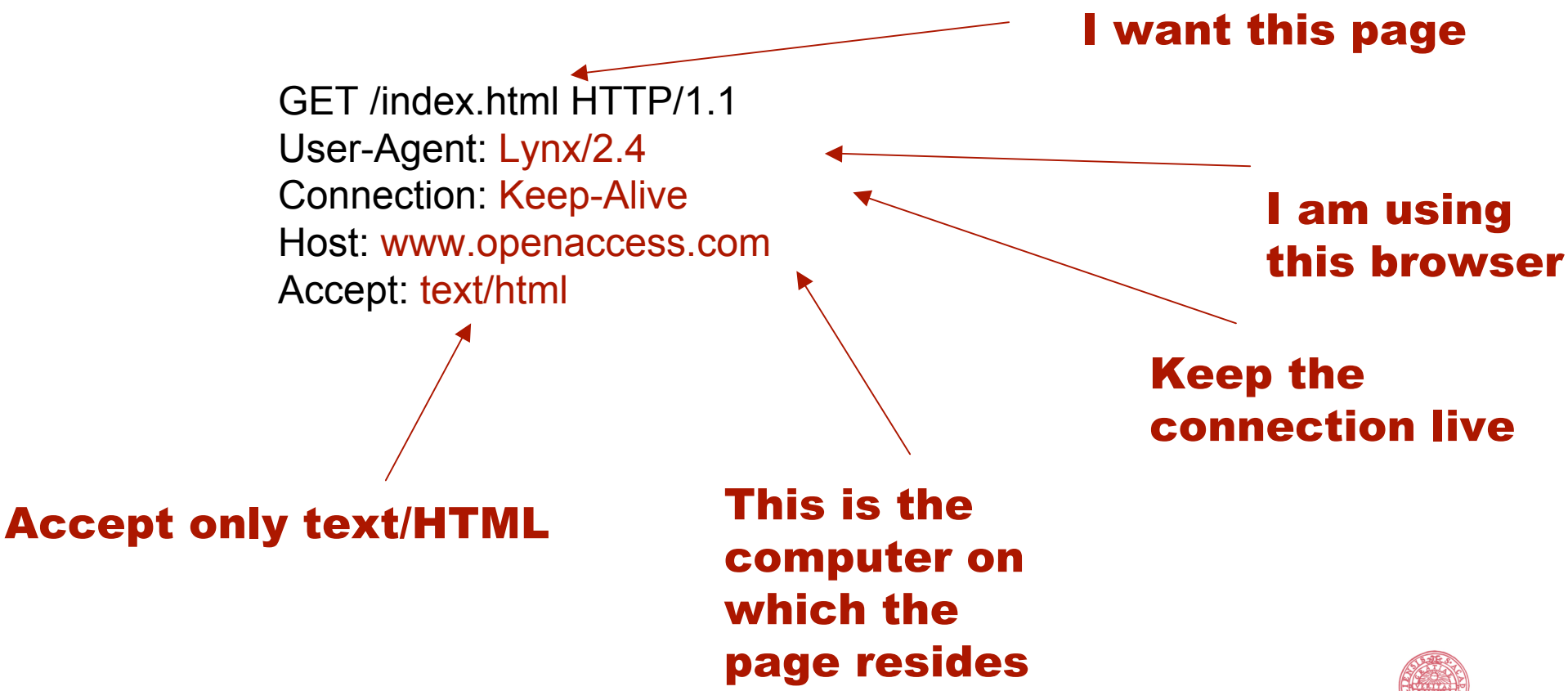
Specifies location of Web page

UPPSALA UNIVERSITET

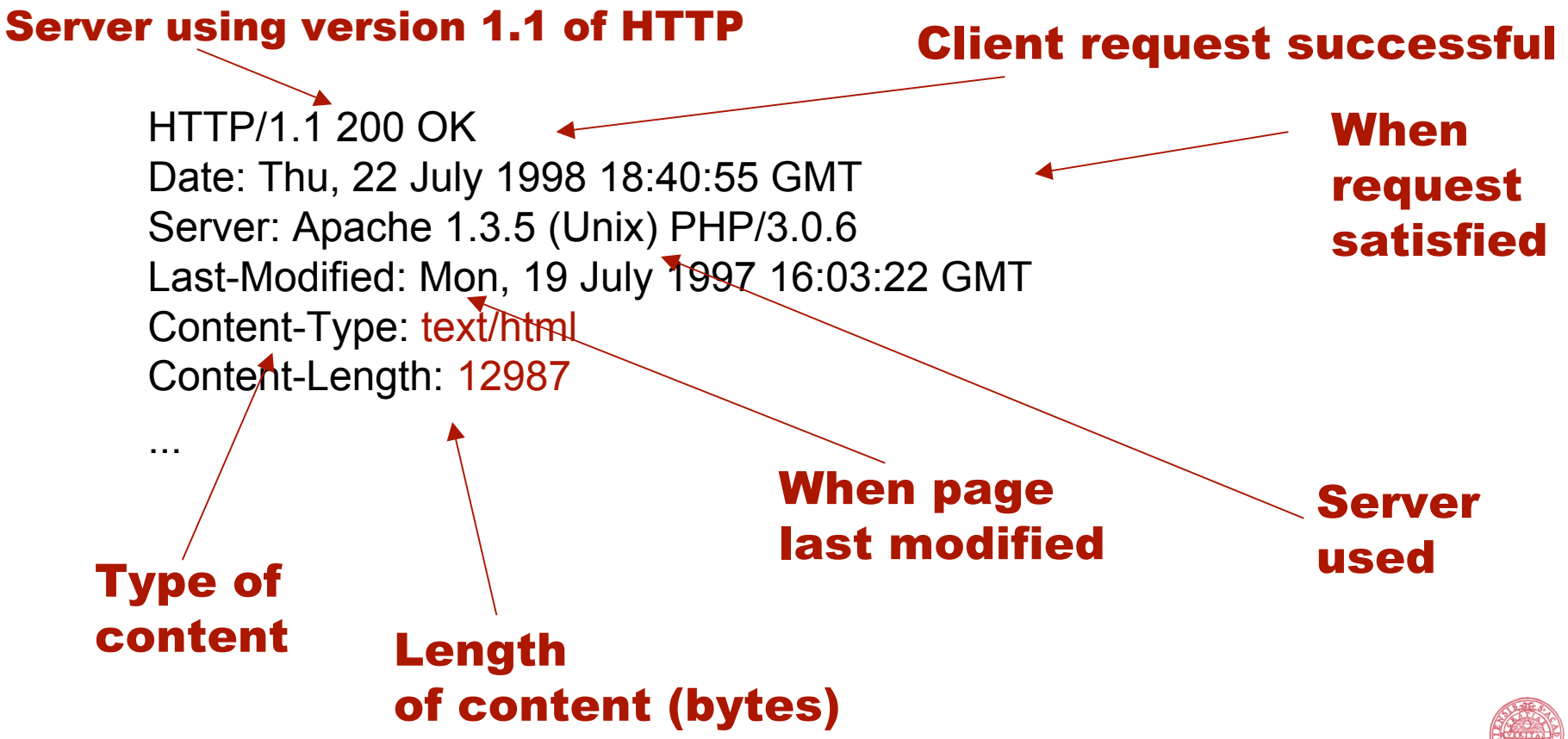
HTTP - Hypertext Transfer Protocol

- **Hypertext Transfer Protocol** (**HTTP**) is a communications protocol for the transfer of information on intranets and the World Wide Web. Its original purpose was to provide a way to publish and retrieve hypertext pages over the Internet between web server and web client.
- HTTP is a request/response protocol standard between a client and a server. A client is the end-user, the server is the web site. The client makes an HTTP request using a web browser, spider, or other end-user tool.
- Typically, an HTTP client initiates a request. It establishes a Transmission Control Protocol (TCP) connection to a particular port on a host (port 80 by default). An HTTP server listening on that port waits for the client to send a request message. Upon receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and a message of its own, the body of which is perhaps the requested file, an error message, or some other information.
- Resources to be accessed by HTTP are identified using Uniform Resource Identifiers (URIs) (or, more specifically, Uniform Resource Locators (URLs)) using the http: or https: URI schemes.

An example of a request



An example of a reply



HTTP commands & status codes

- Some HTTP commands:
  - GET Get a file from a Web server
  - HEAD Like GET no content sent just header info
  - POST Invoke a program on a form
  - PUT Store some resource
  - DELETE Deletes the specified resource
  - OPTIONS Find out about communication options
  - CONNECT Convert request connection to transparent TCP/IP tunnel
- Status codes:
  - Indicates the result of a request
  - Starting with 2 mean correct response
  - Starting with 3 means browser has to carry out some other action
  - Starting with 4 means there is a problem, normally the page cannot be found (Error 404)
  - Starting with a 5 indicates a serious server problem

Web Forms

- A **webform** on a web page allows a user to enter data that is, typically, sent to a server for processing and to mimic the usage of paper forms.
  - Forms can be used to submit data to save on a server (e.g. ordering a product) or can be used to retrieve data (e.g. searching on a search engine).
- XHTML/HTML forms
  - A form in XHTM or HTML is by far the most common way to use a form online.
- The following elements can make up the user-input portion of a form:
  - input field:
    - text (a simple text box that allows input of a *single* line of text)
    - checkbox - a check box
    - Radio - a radio button
    - file - a file select control for uploading a file
    - reset - a reset button that, when activated, tells the browser to restore the values to their initial values.
    - submit - a button that tells the browser to take action on the form (typically to send it to a server)
  - textarea - much like the text input field except a textarea allows for multiple rows of data to be shown and entered
  - select - a drop-down list that displays a list of items a user can select from
  - Many special elements are also available through JavaScript libraries.

Web forms combined with programs

- Forms can be combined with various scripting languages to allow developers to create dynamic web sites.
- **Client-side scripting:**
  - The de facto standard client-side scripting language for web sites is JavaScript. While client-side languages used in conjunction with forms are limited, they often can serve to do pre-validation of the form data and/or to prepare the form data to send to a server-side program.
- **Server-side scripting:**
  - Server-side programs can do a vast assortment of tasks to create dynamic web sites from authenticating a login to retrieving and storing data in a database. Most server-side program requests must pass through the web server's Common Gateway Interface to execute the program to actually perform the tasks. The advantage of server-side over client-side is the concentration of functionality onto one computer (the server) instead of relying on each web browser implementing all of the various functions the same.
- Scripting languages are the most common server-side programs used for web sites, but it is also possible to run compiled programs.
  - Some of the scripting languages commonly used: PHP, Perl, ASP, Adobe ColdFusion, and JSP.
  - Some of the compiling languages commonly used: C, C++, and Java.

An example of an HTML form

```
<FORM METHOD=”POST” ACTION=”/cgi-bin/Form1process”>

<P> Please type your name below </P>

<P> <INPUT TYPE=”TEXT” NAME=”nameField” MAXLENGTH=”30”> </P>

<P> Please type your address below </P>

<P> <TEXTAREA NAME=”addField” ROWS=”5” COLS=”40”> </TEXTAREA> </P>

<P> Male<INPUT TYPE=”RADIO”
      NAME=”maleButton”
      VALUE=”mButt”> </P>

<P> Female <INPUT TYPE=”RADIO”
      NAME=”femaleButton”
      VALUE=”fButt”> </P>

<P> <INPUT TYPE=”SUBMIT” VALUE=”Submit”> </P>
</FORM>
```

A form containing two radio buttons, a text field and a text area

The Common Gateway Interface (CGI)

- An interface in the Web server
- Used to communicate with non-Web software such as database server software
- Include a number of environment variables that can be interrogated
- Some examples of CGI variables:

– **SERVER_SOFTWARE**	**Server software**
– **REMOTE_ADDR**	**The address of the client**
– **SERVER_PROTOCOL**	**The protocol used and it version**
– **REQUEST_METHOD**	**The method used by the request for example GET**
– **SERVER_PORT**	**The port number used for the Web server (normally 80)**

Logging by the Web server

- Produces a file of data about accesses to a Web server
- Used for marketing purposes
- Used for optimising the web server
- Also used for security post-mortems

Webmaster duties

- Optimising the performance of a Web server
- Setting up the initial configuration
- Setting up initial directories and files
- Implementing security policies
- Monitoring server logs

Web development

- Typical web development an be divided into:
  - **Client Side Programming**
  - **Server Side Programming**

Client side programming

- Client side programming involves code resident at the client although it may be downloaded from the server
- Examples of Client Side Coding tools:
  - CSS
  - XHTML
    - *according to modern web design standards, XHTML replace the older* HTML
  - Applets
  - Javascript
  - Flash
    - *Adobe* Flash Player *is a ubiquitous client-side platform ready for rich internet applications* (RIAs).
    - Flex 2 *is also deployed to the Flash Player (version 9+)*
  - Microsoft SilverLight

Server side programming

- Server side programming involves code at the server:
  - Server-side includes (SSI), Common gateway interface (CGI), Perl, PHP, JavaServer Pages (JSP), Java Servlets, Python, Ruby, ColdFusion Markup Language (CFML), ASP, ASP.NET, Tcl

- Examples of combinations of Server Side Coding tools:
  - PHP and MySQL
  - ASP and MSSQL
  - ASP.NET and MSSQL
  - CGI and/or Perl
  - Java, e.g. J2EE or WebObjects
  - Python, e.g. Django (web framework)
  - Ruby, e.g. Ruby on Rails
  - Smalltalk, e.g. Seaside
  - ColdFusion
  - Lotus Domino

Applets

- Code downloaded to client
- Code referenced in HTML
- Do not have constructors
- Heavy security restrictions on such downloaded code
- Ex:

```
<OBJECT code = "java:MyAp.class" WIDTH = "200" HEIGHT = "300"> </OBJECT>
```

Differences between applets and applications

- Applets do not use constructors
- Not allowed to use `System.exit` browser terminates
- All output to visual objects
- Extends the `Applet` class

- Applications use constructors
- Uses `System.exit` for termination
- Output to visual objects and `System.out`
- Extends `Frame` normally

Java and JavaScript

- A significant advance in Web technology was the Java platform by Sun Microsystems.
  - Web pages can embed small Java programs (called Java client-side applets) directly into the view. These applets provide a richer user interface than simple web pages.
  - Applets never gained the expected popularity for various reasons, including lack of integration with other content (applets were confined to small boxes within the rendered page).
  - Adobe Flash now performs many of the functions that were originally envisioned for Java applets, including the playing of video content, animation, and some rich UI features.
- JavaScript, in comparison, is a scripting language initially developed for use within web pages.
  - In conjunction with a Web page's Document Object Model, JavaScript has become a much more powerful technology than its creators originally envisioned. The manipulation of a page's Document Object Model after the page is delivered to the client has been called Dynamic HTML (DHTML), to emphasize a shift away from *static* HTML displays.
  - In simple cases, all the optional information and actions available on a JavaScript-enhanced Web page will have been downloaded when the page was first delivered.
- Ajax ("Asynchronous JavaScript And XML")
  - Ajax is a JavaScript-based technology that provides a method whereby parts *within* a web page may be updated, using new information obtained over the network at a later time in response to user actions.
  - Avoid whole-page reloads.
  - Ajax is seen as an important aspect of what is being called Web 2.0.
  - Ajax techniques in use can be seen in Gmail, Google Maps, and other dynamic Web applications.
- Java itself has become more widely used as a language for server-side and other programming.

JavaScript examples

- javascript is another type of client-side programming (other than java)
- javascript programs are declared in the **head** portion of your web page
- javascript code goes between **script** tags

JavaScript - example #1

js1.html

```
<html>
<head>
   <title>
        first javascript program
   </title>
   <script language="javascript">
        document.write( "<h2> hello from javascript!</h2>" )
   </script>
 </head>
 <body>
   <p>
   <font color="red">
        hello from the body of your web page!
   </font>
</body>
</html>
```

JavaScript - example #2

js2.html

```
<html>
<head>
<title>
second javascript program
</title>
<script language="javascript">
var num;
num = window.prompt( "enter a number", "0" )
document.write( "<h2>hello from javascript!</h2>" )
document.write( "num=" + num )
</script>
</head>
<body>
<p>
<font color="red">
hello from the body of your web page!
</font>
</body>
</html>
```

JavaScript - example #3

js3.html

```
<html>
<head>
<title>
third javascript program
</title>
<script language="javascript">
var num;
var num2;
num = window.prompt( "enter a number", "0" )
num2 = 2*parseInt( num )
document.write( "<h2>" )
document.write( "hello from javascript!" )
document.write( "<p>" )
document.write( "your number =" + num )
document.write( "<p>" )
document.write( "2 times your number =" + num2 )
</script>
</head>
<body>
<p>
<font color="red">
hello from the body of your web page!
</font>
</body>
</html>
```

JavaScript - example #4

js4.html

```
<html>
<head>
<title>
fourth javascript program
</title>
</head>
<body>
<p>
<font color="red">
hello from the body of your web page!
<form name="form1">
<input type="text" name="formname">
</form>
<script language="javascript">
var myname;
myname = prompt( "enter your name:", "" )
document.form1.formname.value = "hi, " + myname
</script>
</font>
</body>
</html>
```

JavaScript - example #5

js5.html

```
<html>
<head>
<title>
fifth javascript program
</title>
</head>
<body>
<script language="javascript">
<!--
alert( "hello friend!" )
myname = prompt( "what is your name?", "" )
okay = confirm( "so, your name is " + myname + "?" )
if ( okay ) {
document.write( "hello " + myname + ", and welcome to my chocolate factory!" )
}
else {
document.write( "sorry, wrong number!" )
}
//-->
</script>
</body>
</html>
```

JavaScript - example #6

js6.html

```
<html>
<head>
<title>
sixth javascript program
</title>
</head>
<body>
<form name="form1">
please enter your name:<br>
<input type="text" name="yourname"><br>
click this button
<input type="button" value="greet"
onclick="form1.txtgreet.value='hello, ' + form1.yourname.value">
<br>
<input type="text" name="txtgreet">
</form>
</body>
</html>
```

JavaScript - example #7

js7.html

```
<html>
<head>
<title>
seventh javascript program
</title>
<script language="javascript">
<!--
function greet() {
document.form1.txtgreet.value = "Hello, " + document.form1.yourname.value
}
//-->
</script>
</head>
<body>
<form name="form1">
please enter your name:<br>
<input type="text" name="yourname"><br>
click this button
<input type="button" value="greet" onclick="greet()">
<br>
<input type="text" name="txtgreet">
</form>
</body>
</html>
```

JavaScript - example #8

js8.html

```
<html>
<head>
<title>
eighth javascript program
</title>
<script language="javascript">
<!--
function calcIt() {
var num1, num2
num1 = Number( document.form1.mynum1.value )
num2 = Number( document.form1.mynum2.value )
document.form1.txtsum.value = num1 + num2
}
//-->
</script>
</head>
<body>
<form name="form1">
enter first number:<br>
<input type="text" name="mynum1"><br>
enter second number:<br>
<input type="text" name="mynum2"><br>
click this button to add them up!
<input type="button" value="add'em" onclick="calcIt()">
<br>
<input type="text" name="txtsum">
</form>
</body>
</html>
```

JavaScript - example #9

js9.html

```
<html>
<head>
<title>
ninth javascript program
</title>
<script language="javascript">
<!--
function calcIt( num1, num2 ) {
var sum
sum = num1 + num2
document.form1.txtsum.value = sum
}
//-->
</script>
</head>
<body>
<form name="form1">
enter first number:<br>
<input type="text" name="mynum1"><br>
enter second number:<br>
<input type="text" name="mynum2"><br>
click this button to add them up!
<input type="button" value="add'em"
onclick="calcIt( Number(form1.mynum1.value), Number(form1.mynum2.value) )">
<br>
<input type="text" name="txtsum">
</form>
</body>
</html>
```

Recap: the web server

- Nothing but a very sophisticated file server
- Reacts to statements expressed in HTTP
- Requests range from those asking for a simple page to those asking for some processing, for example forms processing
- A number of commercial servers in existence, however the most popular is Apache a freeware product
- Technologies used for server-side programming
  - Server side includes
  - Scripting languages such as PHP and others
  - Java Server Pages
  - Servlets
  - Active Server Pages (Microsoft)
  - Web services

Server Side Includes

- Server Side Includes (SSI) is a simple server-side scripting language used almost exclusively for the web. As its name implies, its primary use is including the contents of one file into another one dynamically when the latter is served by a web server.
- SSI is primarily used to "paste" the contents of one or more files into another. For example, a file (of any type, .html, .txt, etc.) containing a daily quote could be included into multiple SSI-enabled pages throughout a website by placing the following code into the desired pages:

```
<!-- #include virtual="../quote.txt" -->
```

- With one change of the quote.txt file, pages including the snippet will display the latest daily quote. Server Side Includes are useful for including a common piece of code throughout a site, such as a navigation menu.
- For a web server in a default configuration to recognize an SSI-enabled HTML file and therefore carry out these instructions, the file must end with an .shtml or .shtm extension (a web server can also be configured to process files with extension .html).
- SSI is most suitable for simple automatization tasks; more complex server-side processing is often done with one of the more complex programming languages Perl, PHP, ASP, JSP, CFML, Python and Ruby.

Servlets

- Portable across servers
- Can draw on a huge amount of Java-oriented technology
- Resident in memory—hence fast
- Security features of Java
- Maintain state across requests
- Uses clean object-oriented model

Servlet mechanisms

- Must implement `Servlet` usually by inheriting from `HttpServlet`
- Methods such as `doGet` and `doPost` respond to commands in HTTP
- Needs to create streams for communication with a Web browser

An example

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloThere extends HttpServlet{

public void doGet
        (HttpServletRequest rq, HttpServletResponse rp)
throws ServletException, IOException
{
        rp.setContentType("text/html");
        PrintWriter browserOut = rp.getWriter();
        browserOut.println("<HTML>");
        browserOut.println
                ("<HEAD><TITLE> Hello there </TITLE></HEAD>");
        browserOut.println("<BODY>");
        browserOut.println("<H3> Hello there</H3>");
        browserOut.println("</BODY></HTML>");
}

}
```

The servlet life cycle

- Servlet is initialised
- Loaded into memory
- Waits for requests
- Acts on requests
- Is finally unloaded

Forms processing

- Servlets can access form elements from Web pages
- They use the method `getParameter`
- Can then build a reply page which depends on the parameter.

Using `getParameter`

```
public void doPost(HttpServletRequest rq, HttpServletResponse rp)
throws ServletException
{
..
String fName = rq.getParameter("FirstName"),
       sName = rq.getParameter("SurName");
if(fName = = null || sName  = = null)
{
//At least one of the text boxes is empty
//Send an error page back to the user
}
else
{
..
PrintWriter out = rp.getWriter();
out.println
("Hello there <P>"+ fName+ " " + sName+ " <P>");
..
}
}
```

Java code for processing forms element

```
<FORM METHOD = "POST" ACTION = "/servlets/formprocessor">

<INPUT TYPE = "text" NAME = "FirstName" Size = "40" MAXLENGTH = 50>
<INPUT TYPE = "text" NAME = "SurName" Size = "40" MAXLENGTH = 50>
```

Persistence and the Web server

- HTTP is stateless
- This means that theoretically it cannot track state across requests
- A number of solutions adopted
- One solution involves user signing in, another is URL rewriting

Servlets and persistence

- Session problem solved simply using Servlets
- Involves the use of a `HttpSession` object analogous to a hash table
- The object is persistent and can be accessed across HTTP requests

An example

```
public void doPost(HttpServletRequest rq, HttpServletResponse rp)
{
..
HttpSession sess = rq.getSession(true);
..
sess.setAttribute("Credit Cards", orderVector);
..
Vector shopVect = (Vector) sess.getAttribute("Credit Cards");
```

1. Create a session object

2. Associate a `Vector` with it

3. Finally retrieve the `Vector` after it has been filled

Dynamic page technology

- Increasingly popular way of programming servers
- Involves embedding code within an HTML page
- Two main technologies Java Server Pages and Active Server Pages (Microsoft)

An example of Java Server Pages

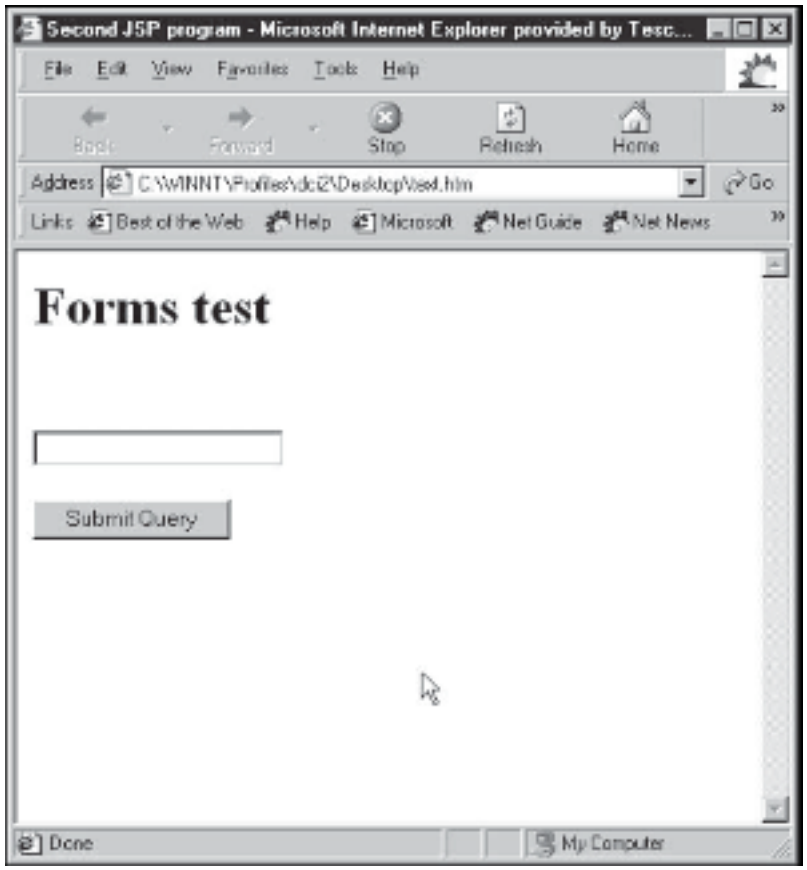
```
<% for (int j = 0; j<100;j++)
{
if (j%2==0) %>
<P>
The value of the even integer is <%= j %>
<% }%>
```

<% and %> used to identify code

A JSP forms example

An HTML page that contains a form to be processed by JSP code is shown below:

```
<HTML>
<HEAD>
<TITLE>
Second JSP program
</TITLE>
</HEAD>
<BODY>
<H1> Forms test </H1>
<FORM ACTION = “calculate.jsp” METHOD=“GET”>
<BR><BR>
<INPUT TYPE = “TEXT” NAME = “Number”>
<BR><BR>
<INPUT TYPE=“SUBMIT” NAME=“Getvalue”>
</FORM>
</BODY>
</HTML>
```



A JSP forms example cont …

When the previous form is submitted, the JSP program calculate.jsp is executed.

This program is shown below:

```
<HTML>
<HEAD>
<TITLE> Third JSP program </TITLE>
</HEAD>
<BODY>
<H1>
One less than the value that you typed in is
<%=Integer.parseInt(request.getParameter (“Number”)) - 1%>
</H1>
</BODY>
</HTML>
```

(request is an implicit object defined by the HttpServletRequest class)

Rich internet applications

- Rich Internet applications (RIA) are web applications that have the features and functionality of traditional desktop applications.
- RIAs typically transfer the processing necessary for the user interface to the web client but keep the bulk of the data (i.e., maintaining the state of the program, the data etc) back on the application server.
- RIAs typically:
  - run in a web browser, or do not require software installation
  - run locally in a secure environment called a sandbox

History of RIAs

- The term "Rich Internet Application" was introduced in March 2002 by Macromedia, though the concept had existed for a number of years earlier under names such as:
  - Remote Scripting, by Microsoft, circa 1998
  - X Internet, by Forrester Research in October 2000
  - Rich (web) clients
  - Rich web applications
- Traditional web applications centered all activity around a client-server architecture with a thin client.
  - RIAs use client side technology to execute instructions on the client's computer circumventing the slow and synchronous loop for many server-based user interactions.
- There is no strict line between what constitutes an RIA and what does not.
  - However, RIAs share one characteristic: they introduce an intermediate layer of code, often called a client *engine*, *between t*he user and the server.
  - The client engine acts as an extension of the browser, and usually takes over responsibility for rendering the application's user interface and for server communication.
- RIA capabilities are limited by the capabilities of the system used on the client.
  - In general, the client engine perform application functions that will enhance the user interface, or improve its responsiveness, compared to standard Web browsers.
  - In RIAs, the client engine performs additional asynchronous communications with servers.

Benfits of RIAs

- Although developing applications to run in a web browser is a much more limiting, difficult, and intricate process than developing a regular desktop application, the efforts are often justified because:
  - installation is not required -- updating and distributing the application is an instant, automatically handled process
  - updates/upgrades to new versions are automatic
  - users can use the application from any computer with an internet connection, and usually regardless of what operating system that computer is running
  - web-based applications are generally less prone to viral infection than running an actual executable
- Because RIAs employ a client engine to interact with the user, they are:
  - Richer. They *can off*er user-interface behaviors not obtainable using only the HTML widgets available to standard browser-based Web applications. This richer functionality may include anything that can be implemented in the technology being used on the client side, including drag and drop, using a slider to change data, calculations performed only by the client and which do not need to be sent back to the server, for example, a mortgage calculator.
  - More responsive. *The interface b*ehaviors are typically much more responsive than those of a standard Web browser that must always interact with a remote server.

UPPSALA UNIVERSITET

Benefits of RIAs …

- The most sophisticated examples of RIAs exhibit a look and feel approaching that of a desktop environment.
- Using a client engine can also produce other performance benefits:
  - *Client/Server balance*. *The demand for cli*ent and server computing resources is better balanced, so that the Web server need not be the workhorse that it is with a traditional Web application. This frees server resources, allowing the same server hardware to handle more client sessions concurrently.
  - *Asynchronous communication*. *The client engine* can interact with the server without waiting for the user to perform an interface action such as clicking on a button or link. This allows the user to view and interact with the page asynchronously from the *client engine*'s communication with the server. This option allows RIA designers to move data between the client and the server without making the user wait. Perhaps the most common application of this is prefetching, in which a*n applicati*on anticipates a future need for certain data, and downloads it to the client before the user requests it, thereby speeding up a subsequent response. Google Maps uses this technique to move adjacent map segments to the client before the user scrolls their view.
  - *Network efficiency*. The *network traffic may* also be significantly reduced because an application-specific client engine can be more intelligent than a standard Web browser when deciding what data needs to be exchanged with servers. This can speed up individual requests or responses because less data is being transferred for each interaction, and overall network load is reduced. However, use of asynchronous prefetching techniques can *neutralize* or even reverse this potential benefit. Because the code cannot anticipate exactly what every user will do next, it is common for such techniques to download extra data, not all of which is actually needed, to many or all clients. or all clients.

Shortcomings of RIAs

- Shortcomings and restrictions associated with RIAs are:
  - *Sandboxing*. Because RIAs run within a sandbox, they have restricted access to system resources. If assumptions about access to resources are incorrect, RIAs may fail to operate correctly.
  - *Disabled scripting*. *Jav*aScript or another scripting language is often required. If the user has disabled active scripting in their browser, the RIA may not function properly, if at all.
  - Clien*t processing speed*. *To ach*ieve platform independence, some RIAs use client-side scripts written in interpreted languages such as JavaScript, with a consequential loss of performance (a serious issue with mobile devices). This is not an issue with compiled client languages such as Java, where performance is comparable to that of traditional compiled languages, or with Flash movies, in which the bulk of the operations are performed by the native code of the Flash player.
  - Script *download time*. *Although* it does not have to be installed, *the additi*onal client-side intelligence (or client engi*ne) of RIA ap*plications needs to be delivered by the server to the client. While much of this is usually automatically cached it needs to be transferred at least once. Depending on the size and type of delivery, script download time may be unpleasantly long. RIA developers can lessen the impact of this delay by compressing the scripts, and by staging their delivery over multiple pages of an application.

Shortcomings of RIAs …

- *Loss of integrity*. *If the appli*cation-base is X/HTML, conflicts arise between the goal of an application (which naturally wants to be in control of its presentation and behaviour) and the goals of X/HTML (which naturally wants to give away control). The DOM interface for X/HTML makes it possible to create RIAs, but by doing so makes it impossible to guarantee correct function. Because an RIA client can modify the RIA's basic structure and override presentation and behaviour, it can cause failure of the application to work properly on the client side. Eventually, this problem could be solved by new client-side mechanisms that granted an RIA client more limited permission to modify only those resources within the scope of its application. (Standard software running natively does not have this problem because by definition a program automatically possesses all rights to all its allocated resources).
- *Loss of visibility to search engines*. *Search engines* may not be able to index the text content of the application.
- Dependence on an *Internet connection*. *While the ideal ne*twork-enabled replacement for a desktop application would allow users to be "occasionally connected" wandering in and out of hot-spots or from office to office, today (in 2007) the typical RIA requires network connectivity.
- *Accessibility*. There *are a lot of kn*own Web accessibility issues in RIA, most notably the fact that screen readers have a hard time detecting dynamic changes (caused by JavaScript) in HTML content.in HTML content.

UPPSALA UNIVERSITET

Web application framework

- **web application framework:**
  - a software framework that is designed to support the development of dynamic websites, Web applications and Web services.
  - The framework aims to alleviate the overhead associated with common activities used in Web development.
  - For example, many frameworks provide libraries for database access, templating frameworks and session management, and often promote code reuse.

Web application framework - history

- Common Gateway Interface
  - As the design of the World Wide Web was not inherently dynamic, early hypertext consisted of hand-coded HTML that was published on web servers. Any modifications to published pages needed to be performed by the pages' author. To provide a dynamic web page that reflected user inputs, the Common Gateway Interface (CGI) standard was introduced for interfacing external applications with web servers. CGI could adversely affect server load, though, since each request had to start a separate process.
- Tighter integration
  - Programmers wanted tighter integration with the web server to enable high traffic web applications. The Apache HTTP Server, for example, supports modules that can extend the web server with arbitrary code executions (such as mod_python) or forward specific requests to a web server that can handle dynamic content (such as mod_jk). Some web servers (such as Apache Tomcat) were specifically designed to handle dynamic content by executing code written in some languages, such as Java.
- Web languages
  - Around the same time, new languages were being developed specifically for use in the web, such as PHP and Active Server Pages.
- Web libraries
  - While the vast majority of languages available to programmers to use in creating dynamic web pages have libraries to help with common tasks, web applications often require specific libraries that are useful in web applications, such as creating HTML (for example, JavaServer Faces).
- Full Stack
  - Eventually, mature, "full stack" frameworks appeared, that often gathered multiple libraries useful for web development into a single cohesive software stack for web developers to use. Examples of this include JavaEE, OpenACS, and Ruby on Rails.

Web application framework - architectures

- Model view controller
  - Many frameworks follow the Model View Controller (MVC) architectural pattern to separate the data model, business rules and user interface.
- Push-based vs. Pull-based
  - Most MVC frameworks follow a push-based architecture. These frameworks use actions that do the required processing, and then "push" the data to the view layer to render the results. **Struts**, **Django**, **Ruby on Rails** and **Spring MVC** are good examples of this architecture.
  - An alternative to this is pull-based architecture, sometimes also called "component-based". These frameworks start with the view layer, which can then "pull" results from multiple controllers as needed. In this architecture, multiple controllers can be involved with a single view. **Tapestry**, **JBoss Seam** and **Velocity** are examples of pull-based architectures.
- Content Management Systems
  - Some self-described content management systems have begun to expand into higher layer web application frameworks. For instance, **Drupal's** structure provides a minimal core whose function is extended through modules that provide functions generally associated with web application frameworks. **Joomla** and **Plone** have similar functionality. Historically these projects have been termed content management systems. However, it is debatable whether "management of content" is the primary value of such systems. Add-on modules now enable these systems to function as full fledged applications beyond the scope of content management. They may provide functional APIs, functional frameworks, coding standards, and many of the functions traditionally associated with Web application frameworks.

Web application framework - features

- Security
  - Some web application frameworks come with authentication and authorization frameworks, that enable the web server to identify the users of the application, and restrict access to functions based on some defined criteria. Django is one example that provides role-based access to pages, and provides a web-based interface for creating users and assigning them roles.
- Database access and mapping
  - Many web application frameworks create a unified API to a database backend, enabling web applications to work with a variety of databases with no code changes, and allowing programmers to work with higher-level concepts. For higher performance, database connections should be pooled as e.g. AOLserver does. Additionally, some object-oriented frameworks contain mapping tools to provide Object-Relational Mapping, which will map objects to tuples.
  - Other features web application frameworks may provide include transactional support and database migration tools.
- URL mapping
  - By automatically rewriting a url with parameters to a friendly URL, the system becomes easier to use, and as an additional benefit, is better indexed by search engines. An example would be the address ending in ?cat=1&pageid=3 to /category/science/topic/physics or just /science/physics. When the id of the category changes the url can stay the same (hence the advantage for search engines). Rewriting URL's can help make an application better conform to some elements of RESTful design practices.

Web application framework - features …

- Web template system
  - *Dynamic web pages* usually consist of a static part (HTML) and a dynamic part, which is code that generates HTML. The code that generates the HTML can do this based on variables in a template, or on code. The text to be generated can come from a database, thereby making it possible to dramatically reduce the number of pages in a site.
  - Consider the example of a real estate agent with 500 houses for sale. In a static web site, the agent would have to create 500 pages in order to make the information available. In a dynamic website, the agent would simply connect the dynamic page to a database table of 500 records.
  - In a template, variables from the programming language can be inserted without using code, thereby losing the requirement of programming knowledge to make updates to the pages in a web site. A syntax is made available to distinguish between HTML and variables. E.g. in JSP the <c:out> tag is used to output variables, and in Smarty, {$variable} is used.
  - Many template engines do support limited logic tags, like IF and FOREACH. These are to be used only for decisions that need to be made for the presentation layer, in order to keep a clean separation from the business logic layer, or the M(odel) in the MVC pattern.
- Caching
  - *Web caching* is the caching of web documents in order to reduce bandwidth usage, server load, and perceived "lag". A web cache stores copies of documents passing through it; subsequent requests may be satisfied from the cache if certain conditions are met. Some application frameworks provide mechanisms for caching documents and bypassing the web template system.

Web application framework - features …

- Ajax
  - Ajax, shorthand for *"Asynchronous JavaScript and XML"*, is a web development technique for creating interactive web applications. The intent is to make web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire web page does not have to be reloaded each time the user requests a change. This is intended to increase the web page's interactivity, speed, and usability.
  - Due to complexity of Ajax programming, there are numerous Ajax frameworks that exclusively deal with Ajax support. Some Ajax frameworks are even embedded as a part of larger frameworks. For example, the Prototype JavaScript Framework is included in Ruby on Rails.
- Automatic configuration
  - Some frameworks minimize web application configuration through the use of introspection and/or following known conventions. For example, many Java frameworks use Hibernate as a persistence layer, which can generate a database schema at runtime capable of persisting the necessary information. This allows the application designer to design business objects without needing to explicitly define a database schema. Frameworks such as Ruby on Rails can also work in reverse, that is, define properties of model objects at runtime based on a database schema.
- Web services
  - *Some framework*s provide tools for creating and providing web services. These utilities may offer similar tools as the rest of the web application.

Web application framework - technologies …

- Languages
  - Many languages have an associated web application framework. However, certain languages either have a critical mass of developers to give a higher level of support to frameworks, or provide features that prove conducive to the development of web application frameworks
- Java
  - There are numerous Java frameworks either in development or in use. Many of these frameworks are built on top of, or borrow elements from the Java EE platform.
- C# and VB.NET
  - C# and VB.NET are the most popular languages on the language-neutral Microsoft's ASP.NET platform. One of the most popular web application frameworks early on was the DotNetNuke web application framework. Since ASP.NET itself is a technology designed for building web applications, it is often wrongly referred to as only a web application framework. This is a reduced view of ASP.NET; besides all of the essential web application framework features, you can choose your favourite language of .NET Languages. ASP.NET also has an integrated AJAX framework, ASP.NET AJAX.

Web application framework - technologies …

- PHP
  - PHP's original design for dynamic web pages has given support to projects such as CakePHP, PRADO, Qcodo, symfony, Zoop Framework, the eZ publish web publishing framework and the Zend Framework. These frameworks assist application structure and modeling by providing a framework layer on top of the core language. These attack the programming problem from the "bottom-up."
  - In contrast with the mentioned frameworks, software projects like MODx, Drupal, Joomla or Typo3 have begun to morph from web content management systems to a higher layer web application framework. Their structure generally provides a minimal core whose function is extended through modules that provide functions generally associated with web application frameworks. As open source projects, their communities contribute many modules (for example, Drupal has over 1,000 such modules and Typo3 more than 2,500). Use of these CMS's core+modules constitutes a method for assembling a website with a broad range of application features without actually doing any PHP-level coding.
- Perl, Python and Ruby
  - There are numerous dynamic language frameworks.
  - Perl has Maypole, Catalyst and Jifty.
  - Python has for example Django, TurboGears, pylons, Quixote, Karrigell and web2py.
  - Ruby has Nitro, Merb and Ruby on Rails.

Web application framework - technologies …

- TCL
  - OpenACS is an open source web application framework designed for developing high traffic web applications in Tcl.
- Smalltalk
  - Seaside is an open source web application framework for developing web applications in Smalltalk. Although the main development of Seaside happens in Squeak there exist ports for other Smalltalk dialects.
- JavaScript
  - Helma is an open source web application framework / publishing system written in Java which uses Javascript as programming language.
- Operating Systems
  - With very few exceptions, web application frameworks are based upon platform independent languages that run on a variety of platforms. While some frameworks may recommend particular configurations, most can run on Windows, Linux, Mac and other Unix-based platforms. A notable exception is DotNetNuke, written for the .NET Framework, that does not support the Mono runtime.

Web application framework - client side

- ActionScript
  - Cairngorm
- JavaScript
  - *Morfi*k, server side and client side.
  - Backbase
  - Clean AJAX
  - Dojo Toolkit
  - Echo
  - Ext
  - JQuery
  - Microsoft AJAX Library
  - Mochikit
  - MooTools
  - OpenLink AJAX Toolkit
  - Prototype JavaScript Framework
  - qooxdoo
  - Rialto Toolkit
  - Rico
  - Script.aculo.us
  - SmartClient
  - Spry framework
  - Yahoo! UI Library

Web application framework - server side

- ASP
  - CLASP
- ASP.NET
  - ASP.NET MVC Framework
  - Base One Foundation Component Library
  - CSLA
  - DotNetNuke
  - MonoRail
- ColdFusion
  - ColdBox
  - ColdFusion on Wheels
  - ColdSpring
  - Fusebox
  - Mach-II
  - Model-Glue
  - onTap

Web application framework - server side …

- Java
  - Apache Cocoon
  - Apache Struts (model-view-controller pattern) (push-based framework)(
  - AppFuse
  - Aranea framework
  - Backbase Enterprise Ajax for JSF / Struts / Spring
  - Click
  - [fleXive]
  - Google Web Toolkit
  - Grails
  - Hamlets
  - ICEfaces
  - IT Mill Toolkit
  - ItsNat
  - JavaServer Faces
  - JBoss Seam (pull-based framework)
  - Makumba
  - OpenLaszlo
  - OpenXava
  - Oracle ADF
  - Reasonable Server Faces
  - RIFE
  - Shale Framework (software)
  - SmartClient
  - Spring Framework
  - Stripes (framework) (model-view-controller pattern)
  - Tapestry (model-view-controller pattern), (pull-based framework)
  - ThinWire
  - WebObjects (Apple)
  - WebWork
  - Wicket framework
  - ZK Framework
  - ztemplates

UPPSALA UNIVERSITET

Web application framework - server side …

- JavaScript (server-side)
  - Morfik, server side and client side.
  - AppJet
  - Helma Object Publisher
  - Jaxer
- Perl
  - Catalyst
  - Interchange
  - Maypole
  - Mason

Web application framework - server side …

- PHP
  - Akelos PHP Framework
  - Antares Framework
  - CakePHP (model-view-controller pattern)
  - Canvas Framework
  - CodeIgniter
  - DIY Framework
  - Drupal
  - epesi
  - FUSE
  - Horde
  - Joomla!
  - Kohana
  - Kumbia PHP Framework
  - MODx
  - PHP For Applications
  - PHPOpenbiz
  - PostNuke
  - PRADO
  - Qcodo
  - QPHP Framework
  - Radicore Framework
  - Seagull PHP Framework
  - Simplicity PHP framework
  - Solar Framework for PHP5
  - Symfony
  - Tigermouse
  - TYPO3
  - Zend Framework
  - Zoop Framework
  - Lion Framework

Web application framework - server side …

- Python
  - CherryPy
  - Django (push-based framework)
  - Karrigell
  - Nevow
  - Porcupine
  - Pylons
  - Spyce
  - TurboGears
  - TwistedWeb
  - Web2py
  - Webware
  - Zope
- Ruby
  - Camping (microframework)
  - Nitro
  - IOWA
  - Ramaze
  - Cerise
  - Ruby on Rails (model-view-controller pattern) (push-based framework)
  - Merb

Web application framework - server side …

- Other languages/Multiple languages
  - Morfik, use Pascal, Basic, Java or C# to develop full Ajax apps.
  - Alpha Five
  - Fusebox (ColdFusion and PHP)
  - Kepler (Lua)
  - HAppS (Haskell)
  - OpenACS (Tcl)
  - Seaside (Smalltalk)
  - UnCommon Web (Common Lisp)
  - Yaws (Erlang)

Dynamic web pages

- Classical hypertext navigation occurs among "static" documents, and, for *web users*, this experience is reproduced using static web pages. However, web navigation can also provide an *interactive experience* that is termed "*dynamic*". Content (text, images, form fields, etc.) on a web page can change, in response to different contexts or conditions. There are two ways to create this kind of interactivity:
  1. Using client-side scripting to change interface behaviors **within** a specific web page, in response to mouse or keyboard actions or at specified timing events. In this case the dynamic behavior occurs within the presentation.
  2. Using server-side scripting to change the supplied page source **between** pages, adjusting the sequence or reload of the web pages or web content supplied to the browser.
     Server responses may be determined by such conditions as data in a posted HTML form, parameters in the URL, the type of browser being used, the passage of time, or a database or server state.
     The result of either technique is described as a **dynamic web page**, and both may be used simultaneously.
- To adhere to the first definition, web pages must use presentation technology called, in a broader sense, rich interfaced pages.
  - Client-side scripting languages like JavaScript or ActionScript, used for Dynamic HTML (DHTML) and Flash technologies, are frequently used to orchestrate media types (sound, animations, changing text, etc.) of the presentation.
  - The scripting also allows use of remote scripting, a technique by which the DHTML page requests additional information from a server, using a hidden Frame, XMLHttpRequests, or a Web service.
- Web pages that adhere to the second definition are often created with the help of server-side languages
  - … such as PHP, Perl, ASP or ASP.NET, JSP, and other languages.
  - These server-side languages typically use the Common Gateway Interface (CGI) to produce *dynamic web pages*.
  - These kinds of pages can also use, on client-side, the first kind (DHTML, etc.).

Dynamic web page cont…

- The Client-side dynamic content is generated on the client's computer.
  - A web server retrieves the page and sends it as is.
  - The web browser then processes the code embedded in the page (normally JavaScript) and displays the page to the user.
  - The innerHTML property (or write command) can illustrate the "Client-side dynamic page" generation: 2 distinct pages, A and B, can be regenerated (by an "event response dynamic") as `document.innerHTML = A` and `document.innerHTML = B`; or "on load dynamic" by `document.write(A)` and `document.write(B)`.

  The problems with client-side dynamic pages are:
  - Some browsers do not support the language or they do not support all aspects (like write command and innerHTML property) of the language.
  - The information cannot be stored anywhere but the user's computer, so it cannot really be used for statistics gathering.
  - Search engines are not able to run client-side languages and cannot crawl links generated by them.
  - Some users have scripting languages disabled in their browsers due to possible security threats.
- Server-side dynamic content is a little bit more complicated.
  - The browser sends an HTTP request and the server retrieves the requested script or program.
  - The server executes the script or program which typically outputs an HTML web page. The program usually obtains input from the query string or standard input which may have been obtained from a submitted web form.
  - The server sends the HTML output to the client's browser.
  - Server-side has many possibilities for dynamic content, but the use of it can be a strain on low-end, high-traffic machines. Some web sites use the Robots Exclusion Standard to keep web crawlers from accessing dynamic pages for this reason. If not properly secured, server-side scripts could be exploited to gain access to a machine.
- Mixing client and server sides
  - Ajax is a newer web development technique for interchange dynamically contents with the server-side, without reloading the webpage.
  - Google Maps is an example of a web application that uses Ajax techniques.

UPPSALA UNIVERSITET

History of dynamic web pages

- It is difficult to be precise about "dynamic web page beginnings" or chronology, because the precise concept makes sense only after the "widespread development of web pages".
- Context and dates of the "web beginnings":
  - HTTP protocol has been in use by the Web since 1990
  - HTML, as standard, since 1996
  - The web browsers explosion started with 1993's Mosaic.
- For server-side dynamic pages:
  - The dynamic page generation was made possible by the Common Gateway Interface, stable in 1993.
  - Then Server Side Includes pointed a more direct way to deal with server-side scripts, at the web server.
- For client-side:
  - The first "widespread used" version of JavaScript was 1996 (with Netscape 3 and the ECMAscript standard).