Lecture One Introduction

DESIGN ARCHITECTURE IS VITAL TO ALL DEVELOPMENT!

There are two forms of architecture:
Information architecture:

- A User-focused form that plans out a user's experience having explored their needs/personas etc.
- Uses: user-needs matrix, user personas, wireframes/walkthroughs, site design and url mapping

System architecture:

- A system focused form that designs from a high level point of view based on system requirements.
- uses: ER diagrams, sequence diagrams, specifications and requirements.

System architecture must be appropriate for both users and clients:
User:

- Varying skill/experience. A user will interact with or initiate the client and will have varying requirements.

Client:

- Accepts response messages that it will act upon by either communicating with the user or altering the environment
- Sends Request messages:
    - Response Messages: Returns information or affects environmental change. HTTPResponse with XML content
    - Request Messages: Ask for information or send information (from input or a sensor) for storage. HTTP request embedding.

Middleware/ An application server is a central component that organises all others by accepting request messages and returning response messages.

Backend/Database is typically on a separate node. It stores data and provide it when needed. Could be a database, an index or a flat file.

Web development is complex due to:

- collision of languages (programming/markup/querying)
- shifting standards
- browser compatibility
- HTTP is a stateless protocol (that is, there is no information retained from previous requests. All information to be processed must come with the interaction exclusively.

Web development is adopting many ‘classical’ design principles: APIs, libraries, frameworks, tools and standards.

By nature web development is very disjoint as it involves several disjoint languages each with their own complex tools. A developer must become familiar with all of these tools - there is no “web development IDE”. That said, there are several tools for checking that your code complies with industry ‘standards’.

Lecture Two Django, The Web Application Framework

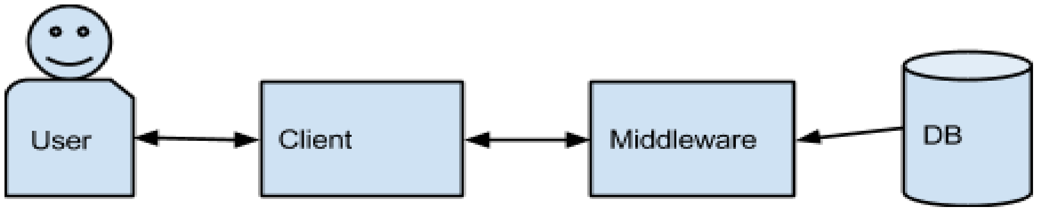
First of all, distinguishing between a top-down/bottom-up architecture:
Top-Down:

- Separate low-level from high-level abstractions which leads to an emphasis on planning and on a modular design. Coding doesn't start until very late.

Bottom-Up:

- Coding begins very early, requires good intuition. Often used to add on to existing modules. Risky.

High-level system architecture:



First decide what needs to be built, then what goes in each box i.e what *technologies* we will use. Middleware is going to be Django, this is the WAF (web application framework) we will be using.

Django can be thought of as a MVC (model-view-controller) architecture where:

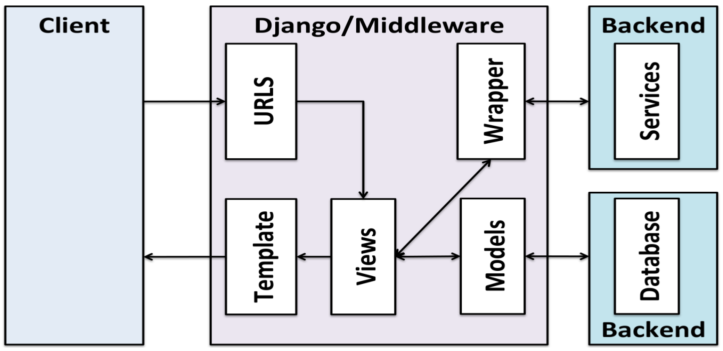
- models maps to models
- controller maps to views *(be careful here!)*
- views maps to templates *(be careful here too!)*

As such it is often referred to as an MTV (model-template-view) system.

There are slight differences, the django-framework acts as a controller, the MTV view provides logic and supplies data (think of it as describing data that the user sees, but not how that data is displayed eg context_dict).

Controlling flow (urls.py) Match view functions with urls
Defining views (views.py) Responsible for handling/processing the specific request, collating backend data and selecting the template.
Providing Templates decouples response format (html) from data
Building Data Models (usually in models.py) Constructs the database and specifies the ER mapping.

simplified display of the internal flow of a django application



Lecture Three System Architectures

The development process:

- Developers: Often a multidisciplinary team of various skill/ability/experience. The team may well be geographically distributed
- Development Environment: A variety of technologies, both legacy and new, all programmed differently.
- Development Process: A flexible process where parallel development is often necessary. There is no single accepted methodology. Agile processes are beneficial (adaptive/evolutionary development).

Usage:

- Users expect **immediate** availability as well as **permanent** availability on a wide range of devices.
- They may be many and have a wide range of cultural/linguistic backgrounds
- Very low threshold for slow or hard to use sites.

Types of Web Applications

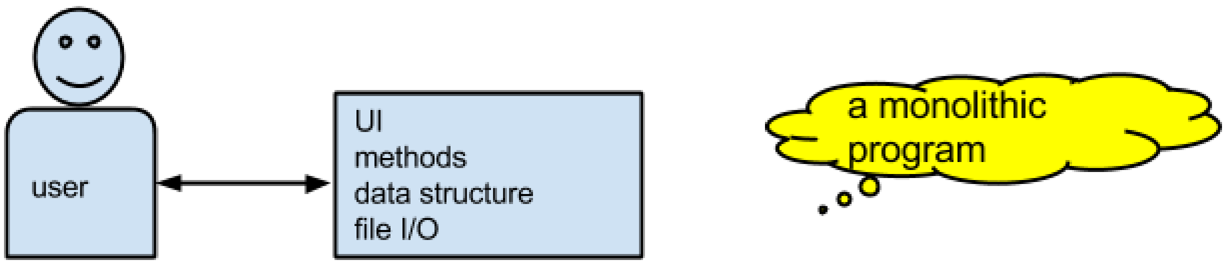
- **static** - basic information displaying HTML
- **interactive** - forms gave possibility for dynamic pages formed by server-side programs
- **transactional** - they also gave the ability to store data in a DB
- **workflow based** - Functionality reflective of business processes (booking travel)
- **Portal oriented** - a Single point of access to many other sites (rango)
- **Collaborative** - Permit multiple users to share information
- **social** - a focal point for communities
- **Mobile/Ubiquitous** - Provide access to small, non-visual devices/sensors.

System architecture shows how every component of a system fits together. Architectural diagrams can be at an algorithmic, component or application level so various diagrams are needed. We will focus on **high-level** system architecture.

Any application must include:

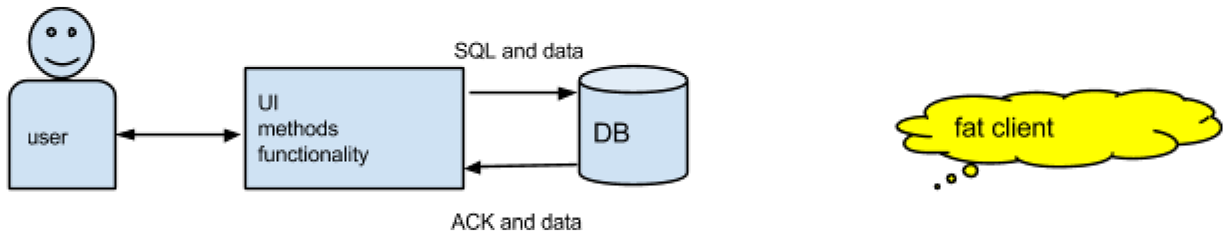
- A user interface
- Logic/algorithms
- Information manipulation
- Data storage.

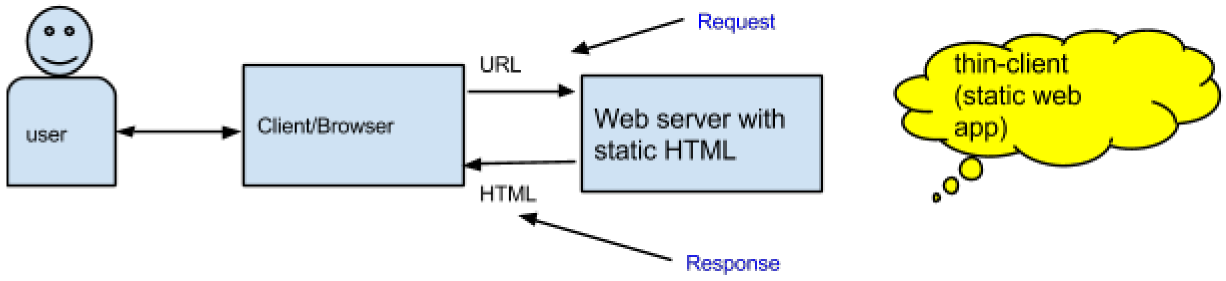
single tier architecture



The structure of applications has changed from monolithic (where everything is coded together) to tiered architectures where every tier can be coded separately and distributed over a network. That said there must be well defined interfaces between the separate tiers.

Two Tier architecture

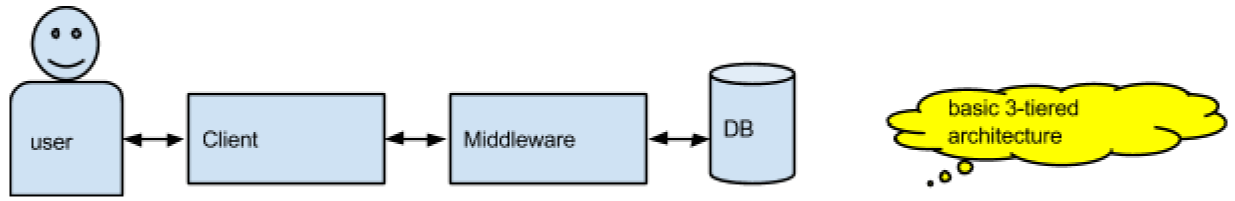




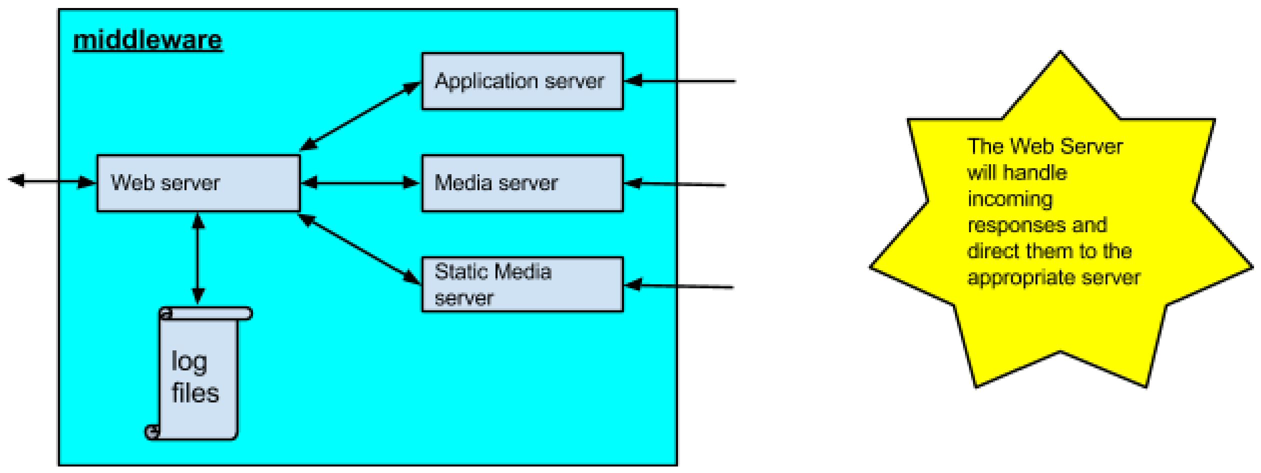
Two tier architecture allows for data and information management which is commonly needed. Often this involves separating a Database Management System (DBMS) from the application to enable secure management of data. Using a DBMS is much easier/quicker than coding access to large amounts of data. Clients act like users, sending queries and acting on the results. Often SQl statements are passed as strings.

The thin-client show above is the standard architecture for serving up static content. Two-tiered structures put a lot of load on the client and ties the client software to the DBMS.

Three or n-tiered architecture: Split application into presentation, application (scope to be >1 tier) and data.



Middleware (zoomed-in)


middleware
Application server
Web server
Media server
Static Media server
log files
The Web Server will handle incoming responses and direct them to the appropriate server

Load Balancing: There are two ways to balance the load, through a **domain name server (DNS)** which rotates resolved URLs through a series of IP addresses that route the message to that machine. Or using a **load balancing server (LBS)** that farms out requests to available machines which in turn inform the LBS of their current load.

Lecture Four System Architecture Diagrams

Using a **high level design** is good because it describes the system making goals, scope and responsibilities clear. It allows for the design to be communicated and permits change as the project develops. It also benefits maintenance, re-usability and accessibility.

Diagrams enable architects to communicate how their application and it's components fit together. Designs often serve as a communication tool with the client. Diagrams ease large and complex applications.

Modified Dataflow Language has the following entities:

- **User** - (or client) instigates or interacts with the application, user's can be end users (who will vary in ability), administrators, developers, agents or even other systems.
- **Client** - (or interface) takes many varying forms, it could be a browser, the API of another system, a device or robot or sensors.
- **Middleware** - Houses an array of components from: DNS, LBS, web servers, application servers, caching servers; though it is often displayed as a single component. The application server is the main interest.
- **Database** - Handles data management (using mySQL, SQLServer or similar). The system needs to be configured by defining tables/relationships, ER diagrams are always useful here.
- **Logs** - (external services) Applications output data to a data sink which stores it as a log. External services represent applications used, they provide an API or interface used to interact with the service.

sometimes they also include:

- **Technology/Devices** - For each box we specify what tech/device is used. ie client: web browser using HTML/CSS/JS, middleware: Apache server using an application built on Django, database: mySQL server.
- **Data Flows** - Arrows denote the flow of information and their direction. Most arrows are two-way, signifying a request followed by a response. The client makes the initial response. This is how entity relations are displayed.

Lecture Six Entities and Relations

Using compressed Chen notation:

- *Rectangles* represent entities
- *Diamonds* represent relationships
- *I, N or M* represents cardinality (where N to M is many-to-many)

Unlike some forms of ER diagram, compressed Chen notation does not label attributes as ovals, rather it lists them as key:value pairs, field:type

Using Django models

In Django every **model** is assigned an **ID**. To Create relationships between models you refer to the model rather than the ID. To denote “many people live in one house” in django:

```
Class Person(models.Model)
        person = models.ForeignKey(House)
```

Think of it as each entity becoming a model. Relationships are specified between entities using:

- models.ForeignKey
- models.OneToOneField
- models.ManyToManyField

(.ForeignKey is used to denote one-to-many)

<u>**Lecture Seven** Web Application Frameworks</u>

Web Applications Frameworks (WAFs) were created to deal with repeated code (boiler plate code) which were common with web development, specifically to access a database and to help with the separation of concerns. Typically they provide

- User authentication, authorisation and security
- DB abstraction
- Template system
- AJAX sub-framework
- Session management
- An architecture usually based on MVC

Like real world frameworks, WAFs provide design and partial implementation of web apps. They provide **default functionality**, while allowing designers to **extend** and **override** to suit specific purposes.

Frameworks have several interpretations:

- A set of classes embodying abstract solutions to a set of problems.
- A reusable design of a system expressed as a set of abstract classes
- A set of prefabricated software building blocks that programmers can use, extend or customized for specific computing solutions
- Large, abstract application in a domain that can be changed to suit individual applications
- Reusable software architecture comprising design and code.

WAF characteristics:

- Inversion of control - application control flow
- Default behaviour - ‘useful’ functionality related to the application domain
- Extensibility - Allow application to customise for a particular purpose.
- Non-modifiable framework code - key components cannot be changed (broken)

WAF advantages:

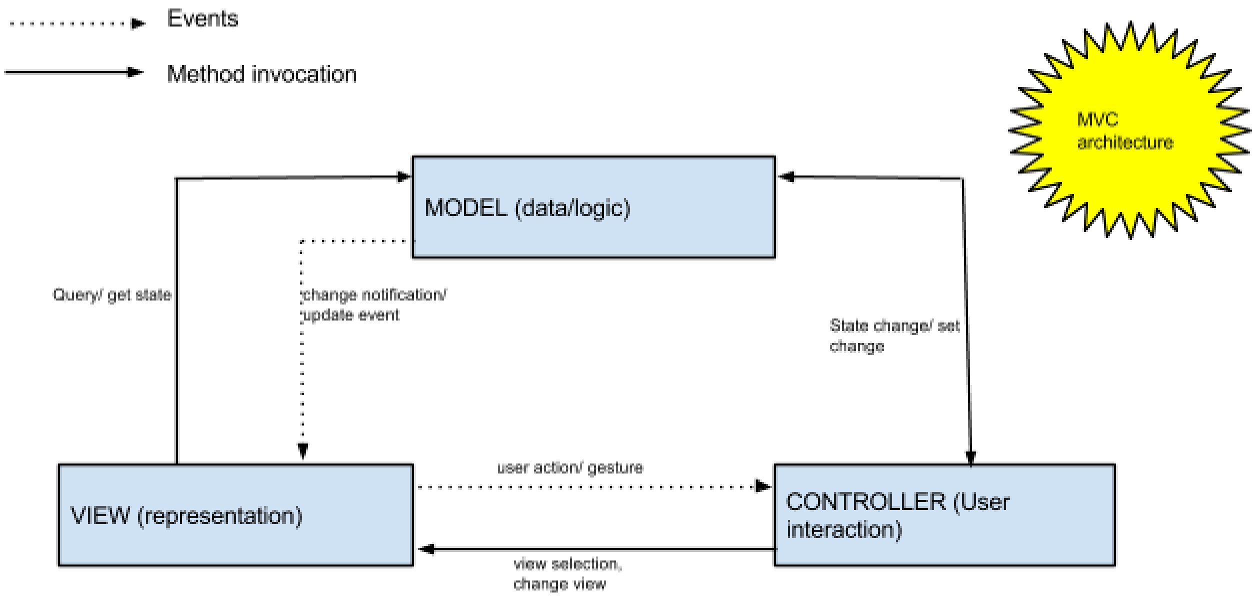
- Enable rapid development
- Concentrate on unique application logic
- Reduce ‘boilerplate’ code (repetitive/complex code)

WAD disadvantages:

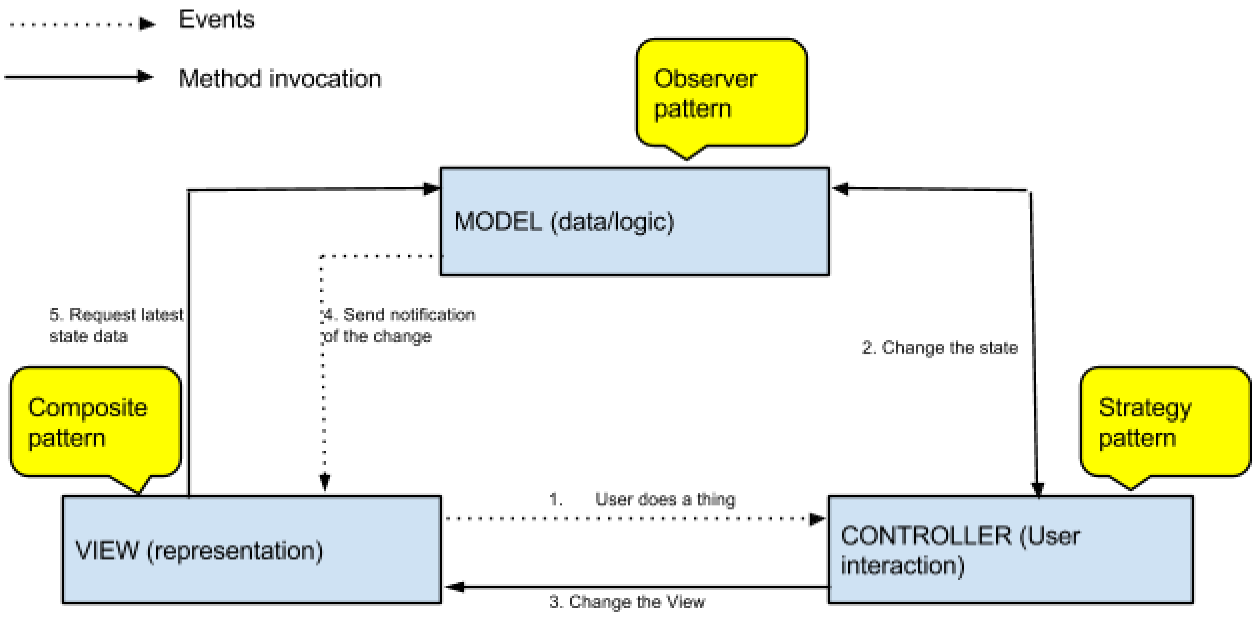
- Imposes a certain model of development
- Can bloat code
- Abstraction can introduce penalties to performance
- Steep learning curve

- (traditionally) poorly documented.

<u>Separation of concerns with GUIs:</u> Presentation logic, application logic and data must be kept separate. To deal with this the MVC was created to map traditional input, processing, output roles to the graphical user interface realm.



Model - Represents application data and domain logic. Notifies Views when it changes, allowing it to query the model. Allows controller to access application functionality encapsulated by the model.
View - Visual representation of the Model, acts as a presentation filter. It is attached to a model from which it gets data necessary for presentation. The view is responsible for forwarding user requests/gestures to the controllers.
Controller - Defines application behaviour and serves as a link between user and system. Interprets user requests/gestures and maps them to actions.



MVC Advantages:

- Easy to Maintain
- Enables independent development and testing
- Provides reusable models & view
- Synchronised views and multiple simultaneous views
- Enforces logical separation of concerns

MVC Disadvantages:

- Initial concerns with splitting of concern
- Increased developmental overheads (3 classes instead of 1) especially with smaller applications
- Debugging is not always straight forward
- Potential for excessive updates
- Requires deep understanding of patterns

A browser MVC Interpretation:

Model -> HTML content
View -> CSS markup adds visual style
Controller -> Combines and renders HTML and CSS

Common WAF Functionality:

- **Web Template System** - provide pages with dynamic content
- **Caching** - Reduces perceived lag
- **Security** - Authentication and authorisation functionality
- **Database Access and Mapping** - Speed up DB access, avoids SQL
- **URL mapping** - Enables URL handling
- **AJAX Handlers** - Creates dynamic, responsive web pages
- **Automatic configuration** - Decrease setup hassle
- **Form Management** - Speed up creation and handling of forms.

Django

- A Python-based WAF
- Primary goal is creation of complex, database-driven sites
- Emphasis on *reusability, pluggability and rapid development*
- Provides optional administrative interface
- Uses MVT over MVC

Why use a WAF?

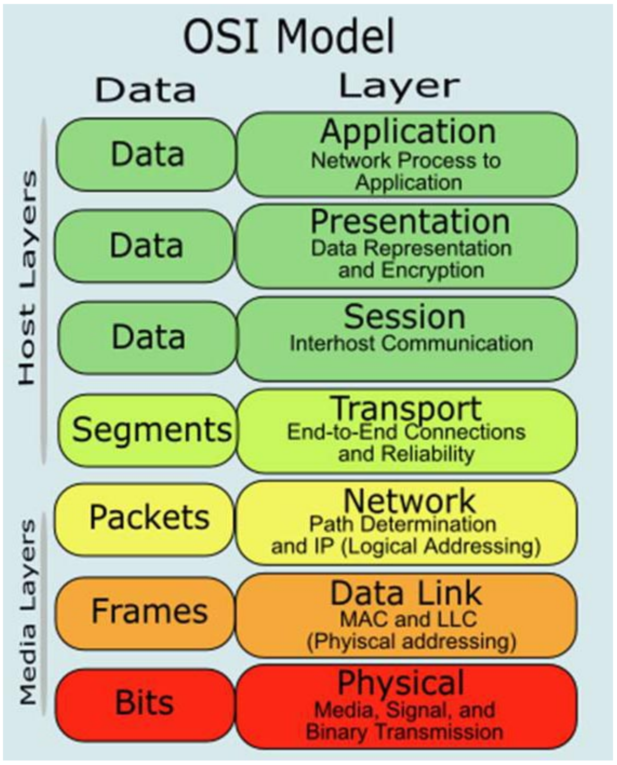
- Enable rapid development inline with rapid development cycle
- Reduce development effort of programming in several different languages
- Include library support for user authentication, session management and creating a web service to help manage complexity
- Reduce boilerplate code

- Design based on increasingly useful MVC (or other/similar) pattern

However:

- They require investment in terms of learning
- Sacrifices flexibility
- Knowledge not necessarily transferable between WAFs
- It is still the early stages of web framework systems - they may not be fully evolved.

Lecture Eight Messaging and Protocols


OSI Model
Data
Layer
Host Layers
Media Layers
Data
Application
Network Process to Application
Data
Presentation
Data Representation and Encryption
Data
Session
Interhost Communication
Segments
Transport
End-to-End Connections and Reliability
Packets
Network
Path Determination and IP (Logical Addressing)
Frames
Data Link
MAC and LLC (Phyiscal addressing)
Bits
Physical
Media, Signal, and Binary Transmission

Seven layers OSI model
(left) describes a view of a networked message. This lecture looks at the **network layer** (using IP) the **transport layer** (using TCP), but mainly the **application layer**

Layers Design Pattern
An architectural design pattern, group subtasks into particular levels of abstraction.

Hypertext Transfer Protocol (HTTP)
Low level application protocol used to send messages, follows a specific request-response pattern with two ways to make a response (GET, POST)

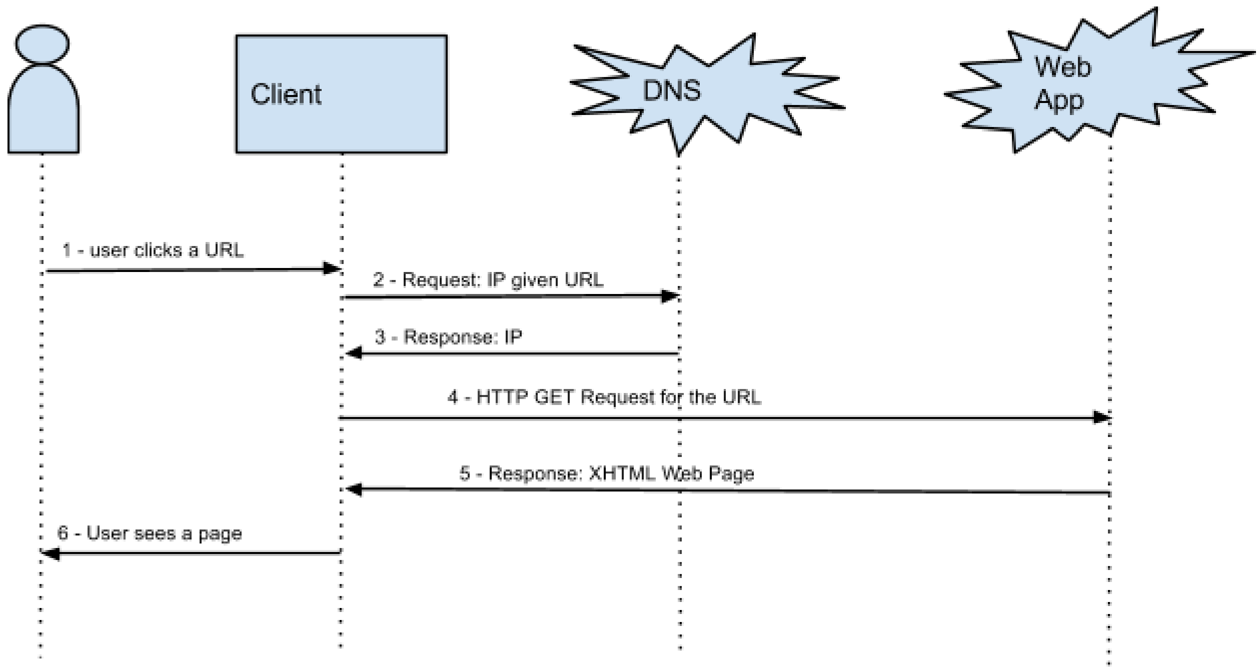
User Agent Specific Protocols (UASP)
Package information to be sent with a response (XML, JSON, XHTML)

The Request-Response Pattern
A requester sends a request message and the receiver of that messages provides a response. Typically this happens synchronously, similar to HTTP, though it can be asynchronous.

When a *user agent* (web browser/client) is asked to send a message (a link is clicked):

- first the URL is turned into an IP address: Request - as DNS for IP, Response - return IP for the URL (ie www.awebsite.com maps to 130.209.34.12).
- Second, a TCP connection is opened on a particular port on the node at that IP address (typically 80 for HTTP or 443 for HTTPS which is encrypted).
- Then a request is made using a specific URL and sent via that TCP connection (ie Request: get the home page of a specified URL), Response: Returns the XHTML for the home page

Sequence Diagrams


Client
DNS
Web App
1 - user clicks a URL
2 - Request: IP given URL
3 - Response: IP
4 - HTTP GET Request for the URL
5 - Response: XHTML Web Page
6 - User sees a page

A system architecture diagram aggregates over all messages, hiding a lot of detail. Sequence diagrams (eg above) display the flow of messages much better as all messages are labelled.

(above examples could include http get requests to the web app from the client for CSS style sheets and javascript files and a http get request to an external library for external javascript)

In Django web apps typically make requests to a database through the Object Relational Mapping (ORM), whilst these may infact be HTTP or some other protocol, we can specify them as ORM Request or ORM Response.

Protocols
A request can be made using various protocols:

- **http** - most common, indicates a file that can be displayed (html, image, sound)
- **https** - Automatically performs SSL (secure socket level negotiation) to send data in an encrypted form
- **file** - a file not recognised in web format to be displayed as text

(some other protocols include:)

- **ftp** - File transfer protocol, refers to sites where files can be extracted and downloaded to client machines
- **mailto** - generates a form for an email message sent to a dedicated user
- **news** - news group or article
- **telnet** - generates a telnet session to this server (command line)

HTTP again
Used to deliver most files and other data through 8 bit characters. Usually HTTP occurs through TCP/IP sockets.

It can also be used to transmit resources, not just files (a resource being a chunk of information identifiable by a URL). HTTP functions as a request-response pattern in a client-server computing model

Request and Response Messages
Under HTTP communication is:

- An HTTP client opens a connection and sends a request to an HTTP server (either a GET or POST typically)
- The server returns a response, usually with the requested resource
- After delivering the response the server closes the connection, making HTTP a *stateless protocol*

HTTP GET
Appends the data to a url as key-value pairs, ie URL ? key= value & key2= value2 . Special characters within the values are replaced (eg spaces become %20) in a process known as url-encoding. The user can see, copy and bookmark them, thus it is easy to 'resubmit' a page. Get should be used for pages which do not affect change on the server, ie information requests.

HTTP Post
Sends data packaged as part of the message, integral to form-data (such as file uploading). POST should be used for affecting server-side change (eg database update, sending an email) or if the dataset is large (GET struggles >1kb)

Other HTTP methods

- **HEAD** - similar to GET except it asks for the return of response headers only
- **PUT** - Used for storing data on a server
- **DELETE** - remove resources from server
- **OPTIONS** - find what a server can do
- **TRACE** - debugging connections
- **CONNECT** - establish a link through a proxy

Stateless
As no information about the client/user is retained HTTP is referred to as *stateless*, this can be a problem if we want to maintain a session. The solution to this is HTTP cookies (client side) or hidden variables (server side) updated through POSTs. (Another solution would be to encode a unique id within the URL).

Cookies are tokens stored on the client that can be included in Requests. It is best practice to store session-id rather than actual data about the user so that the server can retrieve information about the user without infringing on privacy.

Inter-Process Communication

HTTP and other URL schemes are based on user-agent/server model of data. If the code has been distributed about the internet in a different way (such as via a web service) then one needs Inter Process Communication Protocols or IPC. IPC techniques are divided into methods for:

- **message passing**
- **synchronisation**
- **shared memory,** and
- **remote procedure calls (RPC)**

Some common techniques include:

- **Remote procedure calls** - a method running on one machine calls a method running on another
- **XML-RPC** - XML protocol for describing method calls
- **SOAP** - The standard method for web services
- **REST** - send XML in a format specific to an application

XML-RPC

Works by sending an HTTP request to a server implementing the protocol. It is designed to call methods. It has enhanced security through encryption and authentication over HTTP however it does not add much over XML as it still requires an application level data model

SOAP

‘Simple Object Access Protocol’ for exchanging structured information in the implementation of web services. It relies on HTTP, XML and RPC, it is the successor of XML-RPC. Soap has several layers:

- Message format
- Message exchange parameters
- Message processing models
- Transport protocol bindings, and
- Protocol extensibility

It suffers from being complicated, verbose, slow and adding lots of overheads.

consists of a soap-envelope which in turn consists of both a soap-header and a soap-body.

REST - REpresentational State Transfer

A different way of using Web Services avoiding specific protocols rather expecting a service to send a resource and for the client to sort it itself (using SAX, DOM, XSLT etc). Rest builds directly on HTTP sending simple GET/POST requests and expected JSON/XML rather than XHTML - similar to AJAX. It is much simpler than both SOAP and XML-RPC.

Lecture Nine Information Architecture

Information architecture is one of two types of architecture (along with system architecture). It focuses on a structural design of shared information environments with a combination of **organization, labeling, search** and **navigation systems** within websites. It also has strong emphasis on shaping information products to support **usability** and **findability**. It is an emerging discipline concentrated on principles of design and architecture in the digital landscape

The key issue of information architecture is that users have goals/aims etc and the organisation may not have solutions/options for these. In addition to this, information architecture (IA) is important as it keeps costs down, finding/ not finding/ construction/ maintenance/ training and brand all have associated costs, by targeting certain areas IA helps achieve optimum efficiency and ultimately keep costs down.

By analysis of **context, content** and **users** IA allows for the development of solutions which focus on**:**

- **Findability** - of the information and the organisation - good navigational structures? usable content searches?
- **Usability** - of the interactions and design - is the site accessable? does it conform to W3C standards? does it use best practice?
- **Understandability** - of the design and the information - is information clearly labelled? is vocabulary controlled and correctly used?

IA systems include Information **retrieval systems**, **navigational systems** and **semantic word networks**.

Top Down / Bottom-Up Design

From an IA perspective a top down design involves designing the user experience from their arrival at the main page of the site.

Bottom-Up design entails catering for the user landing somewhere on the site (typically via a search engine).

Lecture Ten System architecture / Information architecture

An **information architect** identifies user-based requirements and is responsible for how users interpret and interact with information. They investigate customers and their needs whilst factoring in business strategy and technology resources. This occurs long before programming begins and helps minimise waste of resources. ***Conceptual.***

An IA seeks to identify the user and decipher their needs as well as detailing their experience. They do this using **user personas, wireframes, walkthroughs** and **user-needs matricies**.

System Architects establish the structure of the system including it's core design features and provide a framework for implementing them. They provide the users' vision for what the system needs to be. ***Logical.***

Personas
User archetypes used to guide design in terms of features, navigation, interaction and visual design. Includes information on demographics and environmental descriptors.

User needs matrix
Design to capture and prioritise all the needs of the user which in turn helps prioritise the design of page elements and creates a snapshot of the user ecosystem.

Wireframes
depict an individual page (or a template) from an architectural perspective. Intersection of IA and visual design. They save time and allow presentation without coding as well as helping to validate page elements and layout.

Drawing Wireframes
Wireframes are sketches, they should be simple and abstract, but representative. Components should be labeled and functionality should be explained. Features can be mapped to specific specifications. Wireframes should be documents to allow for A/B testing and showing the evolution of design. Wireframes are a good article for feedback and allow for iterative design through prototyping.

Site maps and URL design
Blueprint for navigation of the site showing how it will be/is organised. They help to structure deep hierarchy, are useful for abstraction and provide a high-level views of the relationships between pages.

It is important to translate the site map into logical url design. Urls should (where possible) user keywords, be concise, be in lower-case, avoiding special characters and give users an idea of where it will take them.

Where possible static URLs should be used as this allows users to revisit information and means crawlers can index the content.

Usability Testing

Takes the form of a questionnaire with a mixture of long and short-form responses asked for which short-form responses often asking to rate features on a scale, with results aggregated. Helps in identifying successful and unsuccessful features as well as convincing clients.

A/B testing is a form of usability testing where participants are asked to compare two (or more) prototypes side by side (or not, in blind test) to identify which features are superior to which.

<u>**Lecture Eleven** XML and XHTML</u>

XML stands for "*eXtensible Markup Language*", it is designed to transport and store data. HTML was seen as too limited and mixing format with structure and SGML (standard generalised mark-up language) was seen as too complex, so XML was developed.

XML is intended to be **simpler**, to **separate form from structure**, to be **extensible** and to **transport and store data.** XML's role is to structure **semi-structured** documents and is described as a general purpose language for data description and interchanged.

It has recently emerged as the dominant standard and has seen large growth (in the form of vocabularies and external tools) in recent years. XML can be extended to describe data in specific domains:

- **XHTML** - web pages
- **WML** - Wireless Markup Languages, a specialisation of XML for mobile data
- **MATHML** - For mathematics and calculations
- **Chemical markup language** - "HTML with Molecules"
- **SOAP** - For describing distributed method parameters

Whereas HTML was designed to display data and elements mix format and structure with content and presentation. In XML tags define structure and formatting is handled separately. XML is **tightly controlled** unlike HTML, it is **case sensitive**, all elements need **start and end tags**, and a **hierarchical structure is enforced**. However XML is flexible, unlike HTML **tags can be created** and user-defined.

An XML documents consists of three parts, an optional prolog containing the declaration, an optional epilog containing comments or processing instructions and a body containing all elements and data.

<u>Elements</u> make up the building blocks of XML, they are case sensitive and they must be closed (with an end tag). They can be nested and they can be empty.

<u>Attributes</u> are the characteristics of elements, they are also case sensitive and they have values - all attributes are text strings so they must be in quotes.

To share XML a pre-defined structure can be used. These describe the tags using DTD (Document type definitions) or XML Schemas and XML Namespaces. XML can be checked against the definitions and validated and are reference either at the top of the file or provided separately. If an XML document is **Well-formed** and conforms to the DTD/Schema then it is said to be **valid**.

XHTML 1.0 Strict is encapsulated by all other forms of XHTML 1.0. it separates visual rendering from layout and is highly usable (many different environments can display it). It is highly configurable by the user and highly maintainable for the developer.

The main XHTML elements are:

- <html> - root element of the document
- <head> - additional content information
- <title> - document title
- <style> - style sheet reference
- <body> - holds the content of the document
- <script> - references client-side scripts (eg JS)
- <meta> - holds additional document information

the block elements in XHTML:

- <h1> … <h6> - Different levels of heading/subheading
- <div> - generic content grouping
- <span> - generic form to add structure
    - <div> and <span> plus the use of style sheets is best practice when using XHTML strict
- <p> - paragraph
- <ol>/<ul> - Ordered and unordered lists

the inline elements in XHTML:

- <a href="link">
- <img src="link" alt="desc">

There are three main classes of attribute in XHTML:

- **Core** - available for all elements, core attributed describe an element's name or id , their class or style and informational attributes (title/lang etc)
- **Event** - calls scripts (such as when submitting a form)
- **Tag specific** - eg src/alt for <img> and href for <a>

Differences with HTML:

- XHTML starts with **<XML>** prologue then <html>
- tags are **case sensitive**
- all **tags must be closed**, including empty tags
- **Hierarchy** is strictly enforced
- Attributes **must** be **in quotes**
- No attribute minimisation (ie <table border = “true”> XHTML, <table border> HTML)
- **Restricted placements** of elements (eg no <p> inside <h1>)
- Style tags deprecated

Semantically:

- XML is designed to **transport and store** data, it centers on **carrying information.**
- HTML was designed to **display** data and centers on **displaying information**

Using XML in a program
either DOM (the document object model) which builds a **hierarchical model in memory** of XML elements or SAX (The simple API for XML) which provides an **event driven parser** for XML. DOM is more appropriate for **entire documents**, SAX is more appropriate for **parts of data** or if there are memory constraints

DOM parsing
DOM is the W3C standard for accessing documents, it is separated into three parts; Core DOM, HTML DOM and XML dom.

XML DOM is the standard object model for XML, it defines objects and properties of all XML elements along with methods for accessing them. It **defines everything in an XML documents as a node**. With the XML documents being a document node (or root node) and every element being an element node. This allows a document to be viewed as a Node tree.

Working with DOM
Regardless of language/environment there is a set structure for working with DOM:

1. load the XMl document object
2. locate the document element or some element of interest
3. extract the attributes, their values and the element data and modify/add/remove elements and attributes.
4. Repeat from step 2 until done

SAX parsing
SAX is a sequential access parser API for XML, it is not an alternative for DOM but another mechanising for reading XML. It is oriented towards **state independent processing** (although a variant StAX is). It is a **stream parser** which is **event-driven**, parsing is unidirectional (cannot turn back) and callback methods are triggered by events when parsing.

Events are available for text nodes, element nodes, processing instructions and comments given they're in XML. Events are triggered when open or close elements are encountered, data sections are encountered or processing instructions are encountered.

Working with SAX

1. Create a custom object model
2. Create a SAX parser
3. Create a DocumentHandler to turn the XML into instances of your custom object model
    - ContentHandler - implements the main SAX interface
    - DTDHandler - for DTD events
    - EntityResolver - for external entities
    - ErrorHandler - for reporting errors and warnings
    - DefaultHandler - for everything else

SAX Handler methods:

- startDocument
- endDocument
- startElement(name,attributes)
- endElement(name)
- characters(ch)

<u>DOM vs SAX</u>

- SAX users less memory
- SAX tends to be faster
- SAX can process files larger than main memory, DOM cannot
- DOM is easier to program
- DOM can handle parsing which require access to the entire document (given it fits in memory)
- SAX cannot handle parsing tasks directly (ie if all XML is required for validation)

<u>JSON</u>

The only contender to XML is JSON. JSON is:

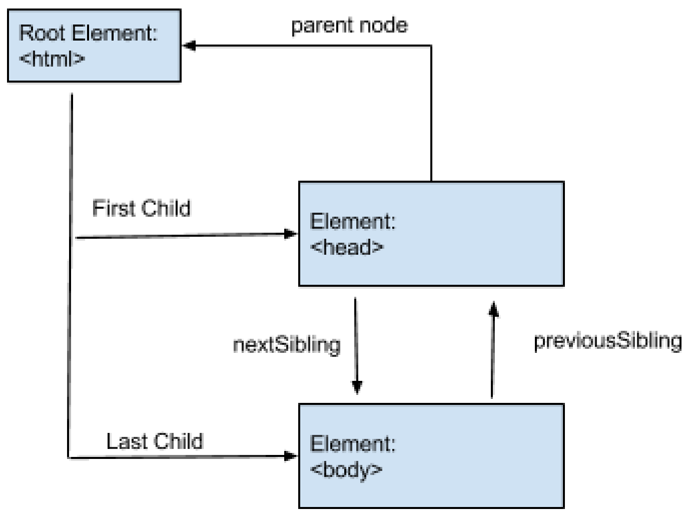
- **lightweight** and "easy" to read/write/be parsed by machines
- built on two universal data structures, a collection of name/value pairs and an ordered list of values
- **language independent**

JSON vs XML:

- JSON can typically encode **more data using less space**
- JSON is **simpler** than XML, it has less grammer and maps directly to a data structure
- XML is **more extensible**, although JSON does not need to be as it's **not a markup language**
- JSON and XML are both open and have potential for **interoperability**.

LECTURE TWELVE Client-Side Environment

The Document Object Model has a hierarchical structure (like a tree) with parents, children and siblings.



Element properties:
- someElement.**innerHTML** - the text value
- someElement.**nodeName**
- someElement.**nodeValue**
- someElement.**parentNode**
- someElement.**childNodes**
- someElement.**attributes**

Methods:
- someElement.**getElementsById(id)** - get element with specified id
- someElement.**getElementsByTagName**
- someElement.**appendChild(node)** - inserts a child
- someElement.**removeChild(node)**

(nb, these are all Javascript methods)

Advantages of the Document Object Model (DOM)
- XML/Tree structure makes DOM easy to traverse and elements can be accessed more than one time
- The tree structure is modifiable, things can be added or modified
- It is the W3C standard

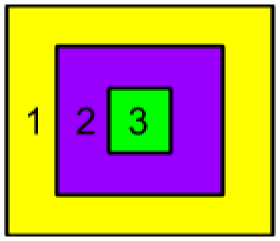
However:
- It can be resource intensive, as it needs to be fully loaded in main memory.
- Depending on the size/complexity of the tree it can be slow
- Not suited to all devices - a graphics intensive application is not likely to be well suited to DOM
  - A better alternative might be to use the Canvas directly through OpenGL or similar.

Event Handling

There are a number of events that can be handled on a webpage:
"onSelect, onKeypress, onMouseOver, onClick, ondblClick, onBlue, onMouseDown, onFocus, onMouseUp, Scroll"
Each part has an associated object as part of the DOM model, the Event object will provide information about: The target element in which the event occurred, the state of the keyboard keys, the location of the mouse and the state of the mouse buttons.

Just like any interactive application, events can be caught and can trigger/execute functions. Each event object has an ‘Event Target’ (the node in the DOM tree from which an event originated) and there are two types of ‘Event Flow’ event capture (global handling) and event bubbling (local handling). Event flow follows a ‘Round trip’ pattern.

Event Bubbling is where events happen from innermost to outermost element and Event Capture is the reverse ie if you clicked on 3 on the image below, Event Bubbling would execute 3 then 2 then 1, Event Capture would execute 1 then 2 then 3.



DOM Development Advantages include:

- control of client software version
- difficult for users to tamper with the software
- releases are relatively easy
- software releases occur often
- user tracking, logging and analysis
- low entry barrier to installation
- familiar user interface conventions

however:

- it’s an http-based communication model
- session management
- application container model
- access to local file system
- is has serious performance issues
- the user interface is not very expansive
- it has high security and privacy expectations
- there is a need to handle a variety of devices

Lecture Fifteen (12-14 irrelevant) Client Side Scripting

The context is as follows: A device runs a browser which allows for the display of content in the form of HTML, stylesheets in the form of CSS and script(s) in the form of JS. The browser will send an HTTP request to the web and receive a response which will update the HTML/CSS and JS.

Javascript is becoming more and more popular. Importantly it is **web only** and has nothing to do with Java (it was originally called "LiveScript"). It has become the standard in all browsers and has been proven useful for a wide range of programming tasks. Whilst it looks like a procedural language, it is more similar to a function language with functions 1st class and support for anonymous functions (*heavily* used by jQuery).

It is Object-Oriented but in a different form to how most languages are in that there are no classes or inheritance. Syntactically similar to Java/C (if/else, while, for) with familiar primitive data types. It is an interpreted language (no compiling) that support dynamic typing in which functions can be anonymous and nested (1st class)

procedural programming emphasises changes in state whilst **functional** programming focuses on treating computation as the evaluation of mathematical functions, avoiding state and mutable data (data that can change)

*Leif's words not mine, but yeah, fuck'em

Most javascript is written by amateurs with a lack of training/discipline(/common sense*) as such it is an extremely expressive language which is severely underutilised.

JS can be written in-line although this violates separation of concerns. It is much easier to manage the code over time if Scripts are kept in external files and linked to from the <head>. Scripts can also be added to event handlers although this is fragile to maintain.

Javascript and the DOM: JS was created with the intention of manually scripting/manipulating documents. HTML documents are modeled using DOM and this means they have methods and properties which can be accessed and altered using JS. Elements in the DOM can be accessed using the methods outlined in lecture 12 (getElementsByTagName() / getElementsByiD() etc)

Despite JS and DOm being functionally useful, coding on the client-side is not particularly easy. One should consider using java as it has a substantially larger library of useful functions. DOM scripting entails a lot of repetitive, domain specific, boiler plate code (ie code is repeated a lot).

To solve this either JS needs it’s own standard library with a focus on domain-specific programming tasks like user interaction or animation for example, or other tools are needed such as jQuery or Prototype

Javascript syntax is simple and lexical (relating to language/words) with datatypes/values/expressions/variables/statements/functions etc. It is case-sensitive and ignore whitespace. Semicolons are optional although they are considered good practice, JS interpreters will automatically add them which is a very bad thing so it is better to be explicit. for example:

```
return
true;
```

would compile to:

```
return;
true;
```

which would return ‘undefined’

Similar to other C-style languages comments can be single // or multiline /* -- */. JS has a very large reserved word list which cannot be used as identifiers for functions or variables.

Functions being 1st class means they can be passed as data types and do not require a return type to be declared in the signature.

Objects are collections of named values known as the object’s *properties*. They can be created using a constructor or the object literal shorthand (ie `var point = {x:2.5, y:5.4}` constructs a new Object and assigns to point variables x and y the values 2.5 and 5.4 respectively). Arrays are also supported and unlike java arrays do not have to be of the same type.

Lecture Sixteen Cascading Style Sheets (CSS)

To achieve style on the web one must use more than just HTML for the display of content. There are a number of ways style (position/colour/font/size etc) can be *achieved*: One can describe a page in XML then use XSL to generate formatted XHTML or one can use CSS with XML or, the best option is to use CSS in combination with XHTML (ie everything).

Stylesheets describe the rendering of html elements stylistically. They can affect individual elements or all elements of a particular kind. CSS follows a basic set of formatting rules:

```
ie.   selector {                          eg.   h3{
              property1: value1;                      color: yellow;
              property2: value2;                      size: 18px;
              …                                       ...
      }                                         }
```

The **selector** indicates the element (or set of elements), examples include:

● p { … }	all <p> elements
● h1, h2, h3 { … }	all <h1>, <h2> and <h3> elements (subheaders)
● *{ … }	all elements
● #menu a { … }	all <a> elements with the id="menu"

The **property** refers to a stylistic aspect to be affected.
The **value** is the specific configuration of said aspect. Values can be one of many types or *units*:

- Numbers - any real number
- Percentages - any real number followed by %, relative often to parent div
- Color - either named, functional rgb or hexadecimal rgb
- Length - paired with numbers to display exact values, mm/cm/pt etc
- Relative Length - em (relative to font-size) / px (relative to pixel size of monitor)

CSS can also be used inline or imported from a document. To use CSS inline it needs to be passed as an attribute, in speech marks, following "style=" for example:

`<h2 style="color: blue; font-size: 18pt">` make <h2> with blue text of size 18.

Inline CSS effects only a single element and not all elements of the same type, it can be useful for overriding existing style but obviously is breaks separation of content and presentation.

External CSS needs to be references in the file head with file extension type="text/css" specified. This is generally the best method in terms of separation of concerns/maintenance and performance.

Class selectors allow for the same with a group of elements. to define a class selector one uses (.) notation for example if we had style: `*.warning{font-weight:bold;}` then any element with the class “warning” would be displayed in bold text, eg <p class=”warning”> bold text.</p>

ID selectors provide a way to stylize unique elements through **id attributes**. IDs are unique and should only be used once however some browsers are less fussy about this than others. The # symbol denotes a unique ID. Eg style of #first-para{ font-weight:bold;} would effect <p id=”first-para”>bold paragraph</p>

CSS also has **inheritance**, styles are applied to an element and all of it’s descendants. CSS also has a **weighting scheme** for it’s style, that is, some elements are more important than others: ID selectors have top priority, then Class attributes and finally element/pseudo-element selectors. Styles are ‘weighted’ in this order as a 3 value comma-separated list (an ID selector would add 1,0,0 to the weight of style, class attributes would add 0,1,0 - etc). In cases where two or more rules have the same weight there is a conflict. CSS is based on the **cascading of styles** made possible by inheritance, specificity and order. The purpose of cascading is to find one winning rule amongst a set of rules and apply that to a given element.

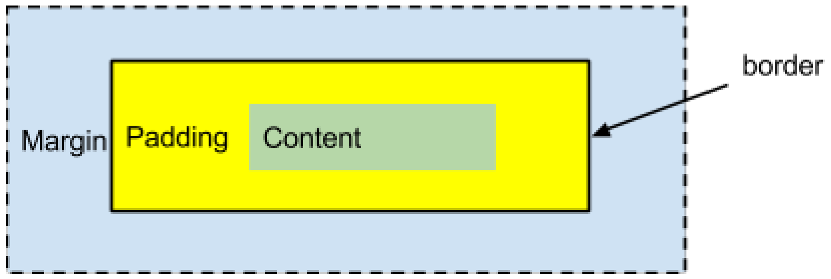
Rules marked important (with the (!) signifier) are given a higher weight in the case of a conflict. Then items are sorted by specificity, finally all declarations applying to an element are sorted, the later a declaration appears in a style sheet, the more weight it is given.

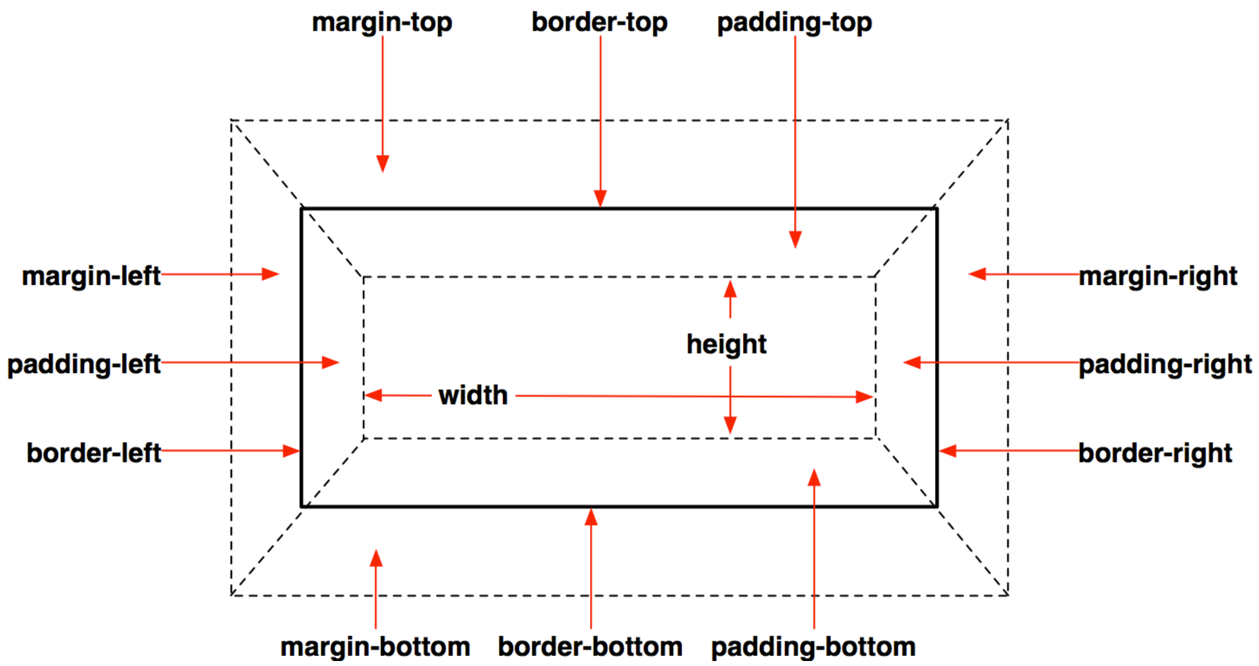
<div> or dividers are the preferred solution to the old way of formatting a page, the extensive use of tables. Divs allow for pages to be separated up into <div> elements and then style can be applied to those elements (in-line or through class/id selectors).

CSS allows for the **floating** of elements horizontally (right/left NOT up/down). Elements after the floating element will flow around so if screen size changes elements will move down.

Positioning in CSS allows you to position elements relative to borders using top/bottom/left.right ie div.onTheTop { position:fixed; top:0px; left:50px;} would be px right of the left border and would be touching the top border. Position also needs a protocol either *static* (the default), *fixed*, *relative*, or *absolute.* (fixed is considered lazy)

CSS follows a box model, shown below, margin/padding and border can all be changed.





<u>CSS Benefits:</u>

- Allows for separation of style and content
- provide a richer document appearance than (X)HTML alone
- Saves time by allowing for style editing in only one place
- improve load times by compactly storing presentation concerns

NOTE: as style is simply another property of an element in the DOM tree, JS can be used to dynamically alter it, achieving various visual effects.

Lecture Seventeen JQuery

jQuery is one of the most popular JS libraries because it simplifies client-side scripting. It does this by simplifying the process to:

- Selecting the DOM element(s)
- creating the UI animations/effects
- handling events
- developing AJAX (more on this next lecture) applications

jQuery acts as a layer of abstraction over various browsers reducing the need to 'browser sniff' (identify the client's browser and code for all likelihoods, delivering the appropriate segment on a case-by-case basis).

JQuery supports plug-ins too which has proved very popular.

It uses a very basic pattern of selecting and acting on a particular DOM element - it reuses CSS selectors:

```
$('p').css('color', 'blue');
```

The use of anonymous functions and chaining makes jQuery very clean and consistent.

<u>**Lecture Eighteen** Asynchronous Javascript And XML (AJAX)</u>

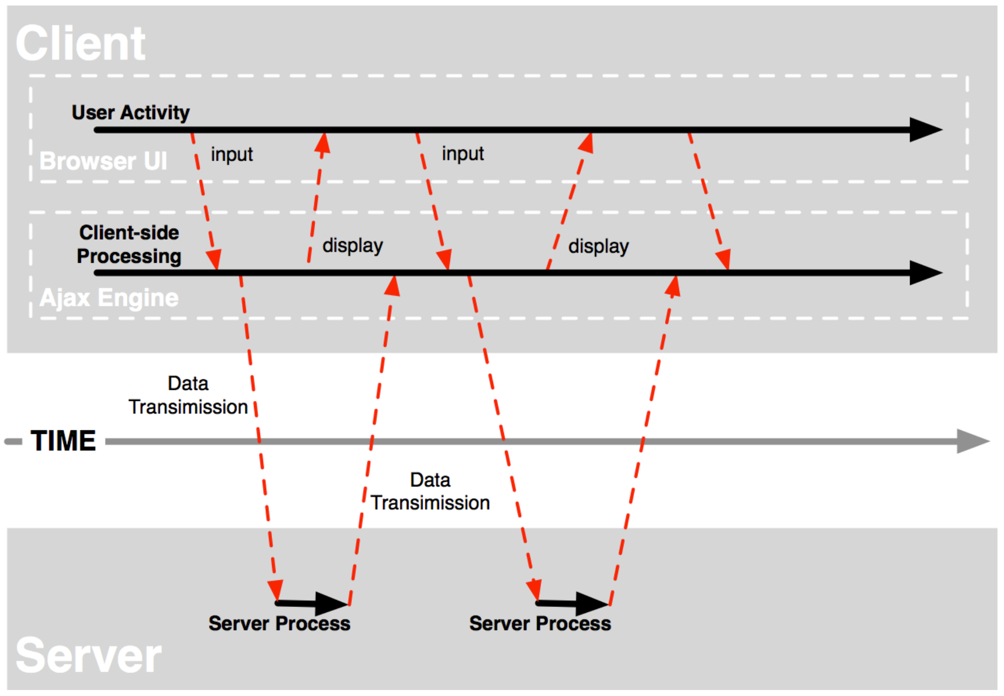
(note, asynchronous comes from greek meaning 'without time')

AJAX is booming in popularity in the last ten years despite all of it's components being around since the late 90s. AJAX is a technology set that underlies web applications. It is made up of XHTML/CSS standards-based presentation, the DOM, XML/XSLT for data interchange and manipulation, asynchronous data retrieval using XMLHttpRequest and Javascript to bind it together.

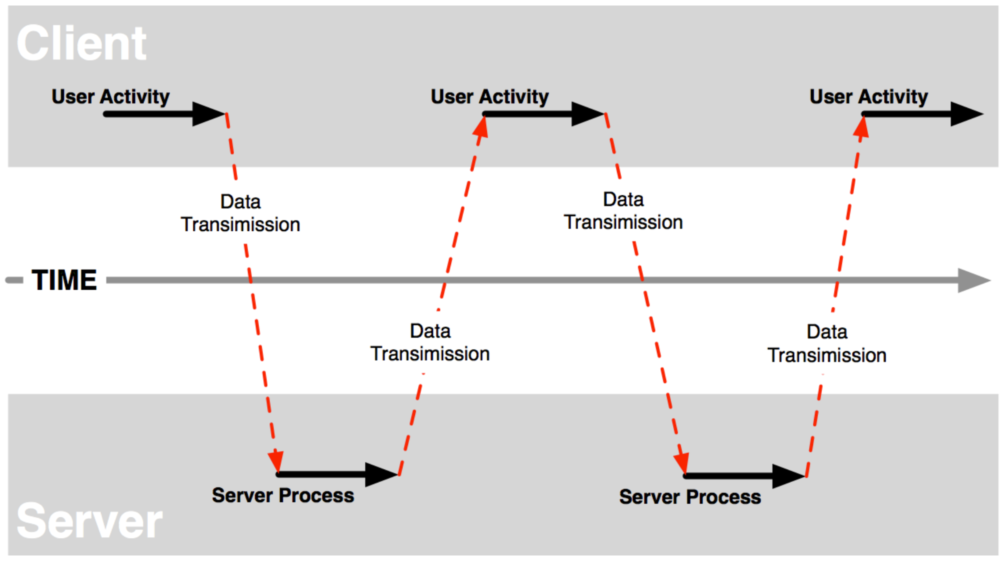
Ajax **eliminates the need to reload** a web page in order to get new content from it and **ends the start-stop interaction** where a user must wait for new pages to load. In general it improves the interactive experience of web apps.

AJAX uses an intermediate layer (the AJAX engine) to communicate between the client and the server. JS allows for the DOM to be create/modify/remove style and content through event handlers attached to user or browser generated events. XML can model that data and through the DOM we can access it.

<u>Achieving asynchronousity:</u> By using the XmlHttpRequest object (XHR) introduced by microsoft in IE5, the client can communicate with the server by sending HTTP requests (much like normal client/server communication)



with AJAX



without AJAX

As with content and style, JS can now programmatically manage HTTP communication as XHR does not block script execution after sending an HTTP request.

<u>XHR Properties</u>

- **readyState property** - cycles through several states whilst it sends an HTTP request, waits for and then receives a response from the server.
- **onreadystatechange property** - Accepts EventListener value, specifying what method must be invoked upon a readyState value change.
- **status property** - The HTTP status code, of type <short> (eg 200 = OK, 404 = not found)
- **responseXML property** - XML response data from the received HTTP response.
- **responseText property** - Text of the HTTP response (XML is not the only method for modelling data in AJAX applications, JSON is a popular alternative)