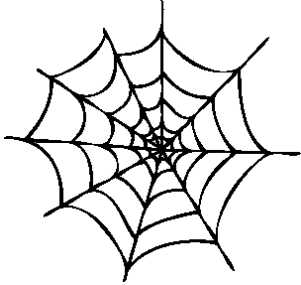
Universitat Autònoma de Barcelona

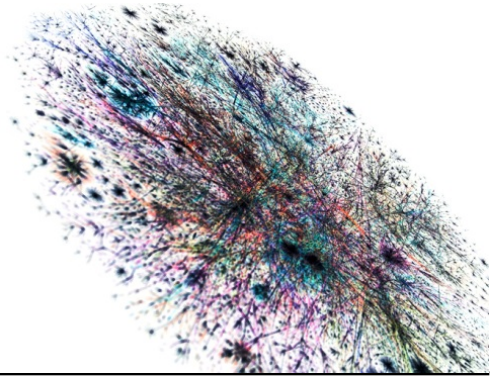


TECNOLOGIES DE DESENVOLUPAMENT PER A INTERNET I WEB

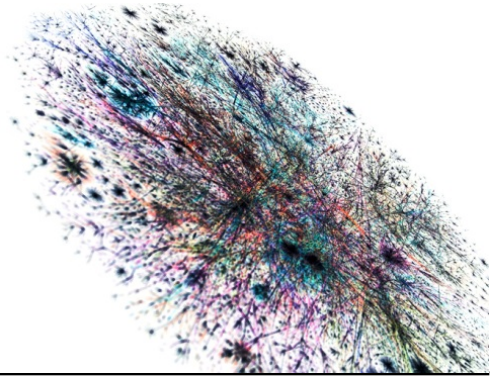
GRAU D'ENGINYERIA INFORMÀTICA - CURS 2013/2014

TEMA 4 – *DISSENY D'APLICACIONS WEB*

1. ARQUITECTURA DE LA WEB
2. SERVIDORS D'APLICACIONS
3. LLENGUATGES DE PROGRAMACIÓ WEB
4. **DISSENY D'APLICACIONS WEB**



1. ARQUITECTURA DE LA WEB
2. SERVIDORS D'APLICACIONS
3. LLENGUATGES DE PROGRAMACIÓ WEB
4. **DISSENY D'APLICACIONS WEB**
   - **4.1 Paradigmes de programació.**
   - **4.2 El model Vista controlador.**
   - **4.3 Virtual Hosting.**
   - **4.4 Aplicacions Web per a dispositius mòbils.**



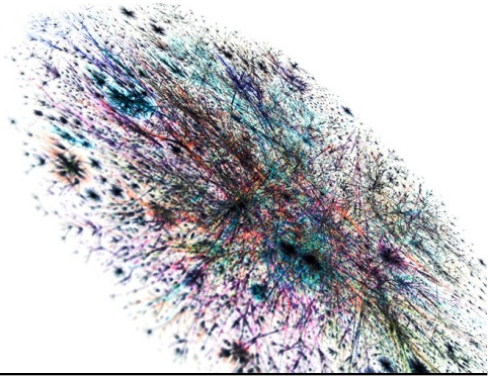
4. DISSENY D'APLICACIONS WEB

4.1 Paradigmes de programació.

4.2 El model vista controlador.

4.3 Virtual Hosting.

4.4 Aplicacions Web per a dispositius mòbils.



Arquitectura Client-Servidor

Els seus orígens daten dels anys 80. Amb l'aparició dels PC la necessitat de tenir un gran ordenador central en empreses va en decrement, essent possible que cada treballador disposi del seu propi PC amb les seves eines de treball. Però l'ús de PC en grans corporacions té inconvenients.

1) Les BDs eren massa grans per a ser gestionades en un PC, els discs durs no tenien suficient capacitat.

2) Si algun usuari actualitzava la BDs tots els altres usuaris havien de veure reflectits els canvis.

Aquests dos grans motius van motivar la creació de la programació Client-Servidor.

Arquitectura Client-Servidor

El **servidor** executa aplicacions que no interactuen amb el client.

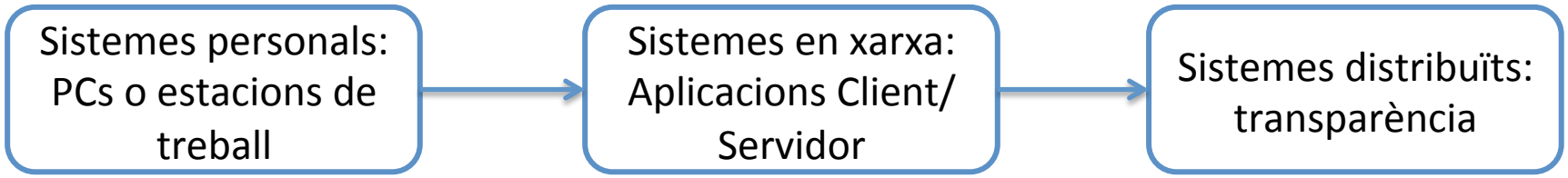
- La seva funció és esperar ordres de l'aplicació del client i interactuar amb aquesta aplicació.
- Habitualment el **servidor** té accés a un gran volum de dades, podent-hi accedir de forma ràpida i eficient.
- Un servidor pot gestionar més d'una petició de forma simultània.

El **client** no té les dades per manipular i/o mostrar al usuari

- L'aplicació client pregunta al usuari quines dades necessita.
- El l'aplicació client connecta al servidor i sol·licita les dades que vol el usuari.
- El client, normalment, obté dades d'un sol servidor i les reporta a un únic client .

Aplicacions Distribuïdes.

L'objectiu principal de les aplicacions distribuïdes és la de **compartir recursos**, ja siguin serveis o dispositius**.**

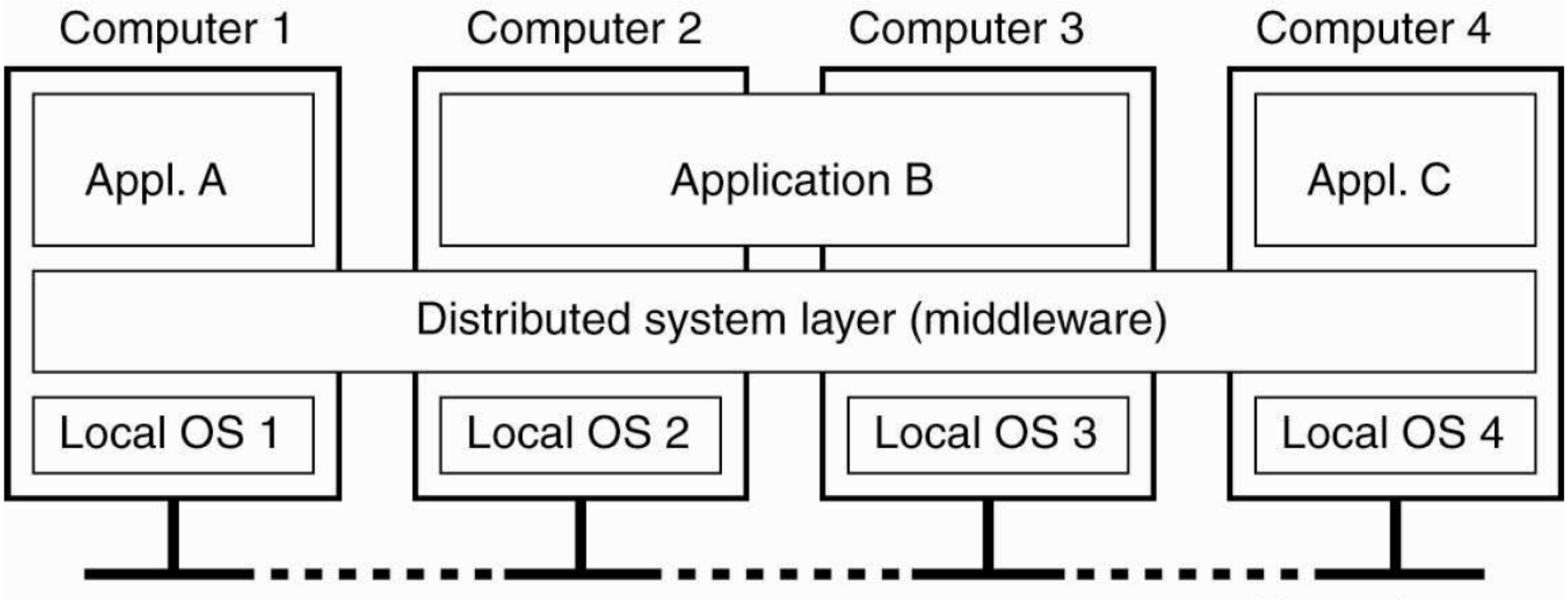


Un sistema distribuït és un conjunt d'ordinadors independents què pels usuaris són vistos com un únic sistema.

1. Un conjunt d'ordinadors.
2. Estan connectats, com si fos un sistema de xarxa.
3. Comparteixen un únic estat.
4. Ofereixen una visió de sistema únic.

Aplicacions Distribuïdes.

Remote Procedure Call (RPC) és un sistema de comunicació què permet a una aplicació executar una subrutina o procediment situada en una altre màquina.

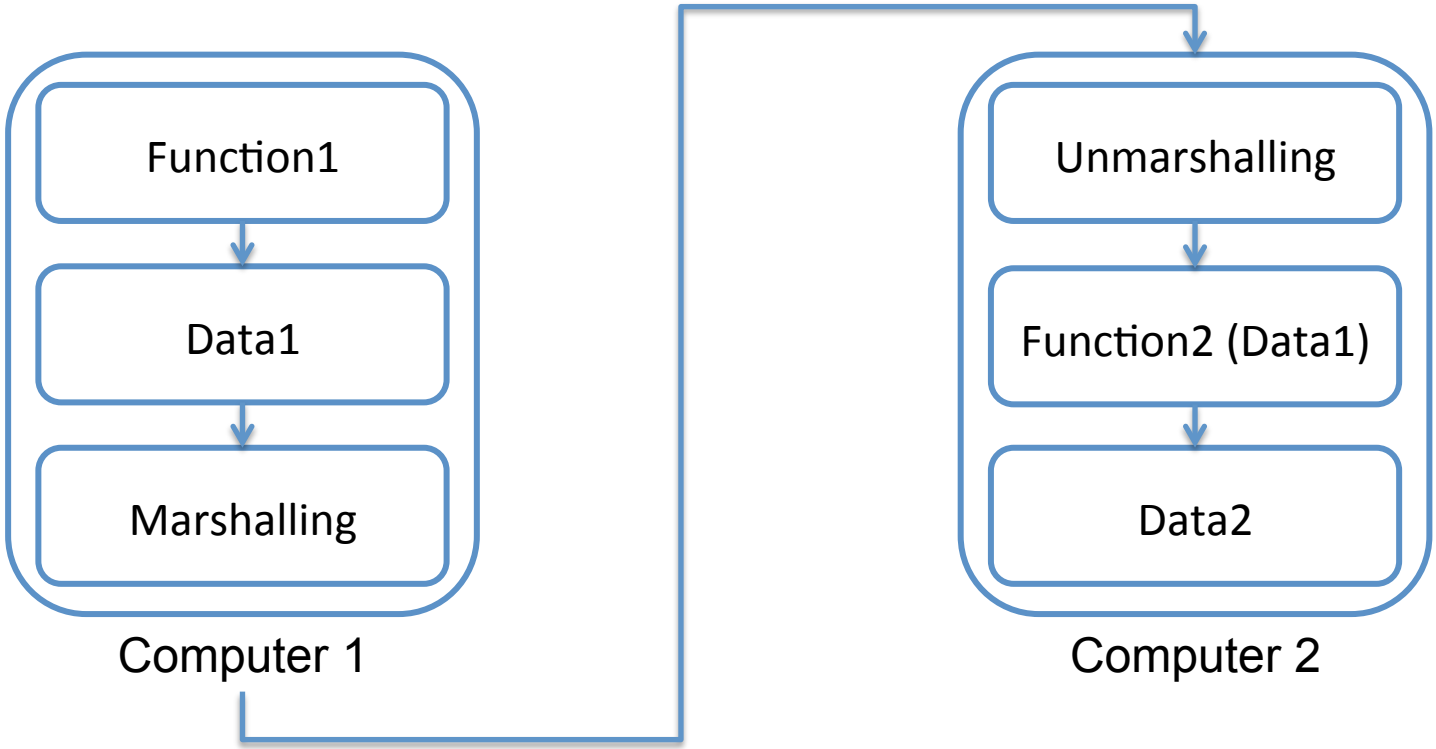


El middleware és la capa de software que permet realitzar crides a mètodes que es troben en màquines remotes.

Aplicacions Distribuïdes.

L'objectiu del middleware és la de gestionar la transferència de l'estructura de dades, fent-la compatible entre els dos objectes que s'estan comunicant.

Aquesta tasca es realitza per diferents estàndards en funció de les necessitats, convertint l'estructura de dades en un flux de bytes que podrà ser transmès per la xarxa, conservant l'ordre dels bytes entre les diferents arquitectures. Aquest procés és anomenat marshalling (semblant a la serialització), i el procés invers unmarshalling.



Aplicacions Distribuïdes. Avantatges i inconvenients.

Avantatges respecte un sistema centralitzat:

- Baix cost: pot estar format per PCs senzills.
- Escalabilitat: és una conseqüència de ser modular.
- Flexibilitat: permet la reutilització de màquines antigues.
- Disponibilitat: la replicació de recursos.
- Paral·lelisme: podem executar tasques amb paral·lel.
- Ús més eficient dels recursos gràcies a la migració.
- Accés transparent als recursos.

Inconvenients respecte un sistema centralitzat:

- Un sistema centralitzat del mateix cost és més eficient que cada un dels components del sistema distribuït.
- Si la distribució de recursos és inadequada, podem saturar recursos mentre altres estiguin lliures.
- Mantenir la consistència potser molt costos i problemàtic.
- La xarxa d'interconnexió sol ser una font de problemes.

Aplicacions Distribuïdes. Propietats.

Aplicacions Distribuïdes

- Transparents
- Escalables
- Fiables i tolerants a errors
- Consistents

Transparents

- Accés: diferències en la representació de dades i com és accedit el recurs.
- Localització: on es troba localitzat el recurs.
- Migració: la mobilitat de recursos.
- Reubicació: la mobilitat de recursos en temps d'execució.
- Replicació: qualsevol recurs pot ser replicat (costos i complex).
- Concurrència: un recurs pot ser utilitzat per més d'un usuari.
- Errors: ocultar la falla i restauració d'un recurs.
- Persistència: utilització de memòria o disc dur.

Aplicacions Distribuïdes. Propietats.

Aplicacions Distribuïdes

- Transparents
- Escalables
- Fiables i tolerants a errors
- Consistents

Escalables

- Créixer: basat en la modularitat.
- Mantenir el rendiment per replicació.

Fiables i tolerants a errors

- Disponibles: els recursos han d'estar disponibles el màxim de temps possible.
- Tolerància a errors: capacitat per seguir operant de forma correcte davant l'error d'alguns dels components.

Aplicacions Distribuïdes. Propietats.

Aplicacions Distribuïdes

- Transparents
- Escalables
- Fiables i tolerants a errors
- Consistents

Consistents

- Problemes de replicació:
  - La xarxa pot fallar.
  - La seguretat del sistema és més vulnerable.
  - La gestió del estat global és més complexa de gestionar.
- Problemes de consistència:
  - Distribució física: varies còpies cadascuna amb el seu estat.
  - Errors i retards ens la comunicació.
  - Manca d'un rellotge global. Com ordenem els events?

Aplicacions Distribuïdes. Escenaris.

Les aplicacions distribuïdes solen ser de gran utilitat en els següents escenaris.

- Aplicacions d'alt rendiment: clúster computing.

- Aplicacions tolerants a errors: replicació.
  - Sistemes informàtics bancaris.
  - Transport públic, etc.

- Aplicacions d'alta disponibilitat: caching o mirroring.
  - Baix temps de resposta (WWW, sistemes de fitxers).
  - La consistència és important però no crítica.

¡PREGUNTA D'EXAMEN!

Quina funció el marshalling o serialització en un sistema distribuït?

¡PREGUNTA D'EXAMEN!

Quines són les avantatges i inconvenients d'un sistema distribuït respecte un sistema centralitzat?

¡PREGUNTA D'EXAMEN!

Quines propietats a de complir un sistema distribuït?

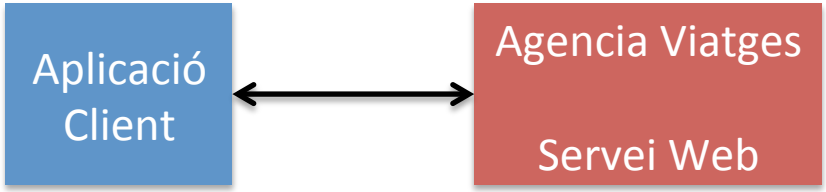
Programació WEB

Els paradigmes de programació són propostes tecnològiques adoptades per la comunitat de desenvolupadors encarades a resoldre un o varis problemes. Principalment existeixen 4 paradigmes de programació: imperatiu, funcional, lògic i orientat a objecte. Cap d'aquest 4 paradigmes encaixa en la programació web, per aquest motiu la comunitat de programadors va definir paradigmes específics per a la programació WEB: *Arquitectura Orientada a Serveis, Arquitectura Orientada a Web.*

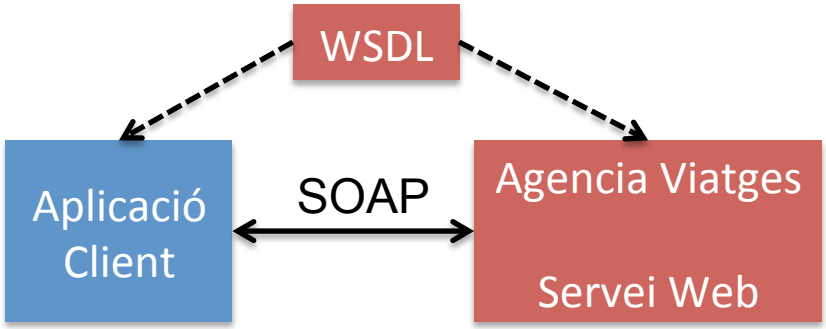
Arquitectura Orientada a Serveis (SOA):

- Un servei (altrament conegut com web services) és una funció, aplicacions o tecnologies que tenen com a propòsit servir als clients.
- Aquestes aplicacions o tecnologies poden intercanviar dades entre elles amb l'objectiu d'oferir un servei
- Els proveïdors ofereixen els seus serveis com a procediments remots, mentre que els usuaris sol·liciten els serveis mitjançant crides remotes als procediments.

Arquitectura Orientada a Serveis:



Arquitectura Orientada a Serveis:

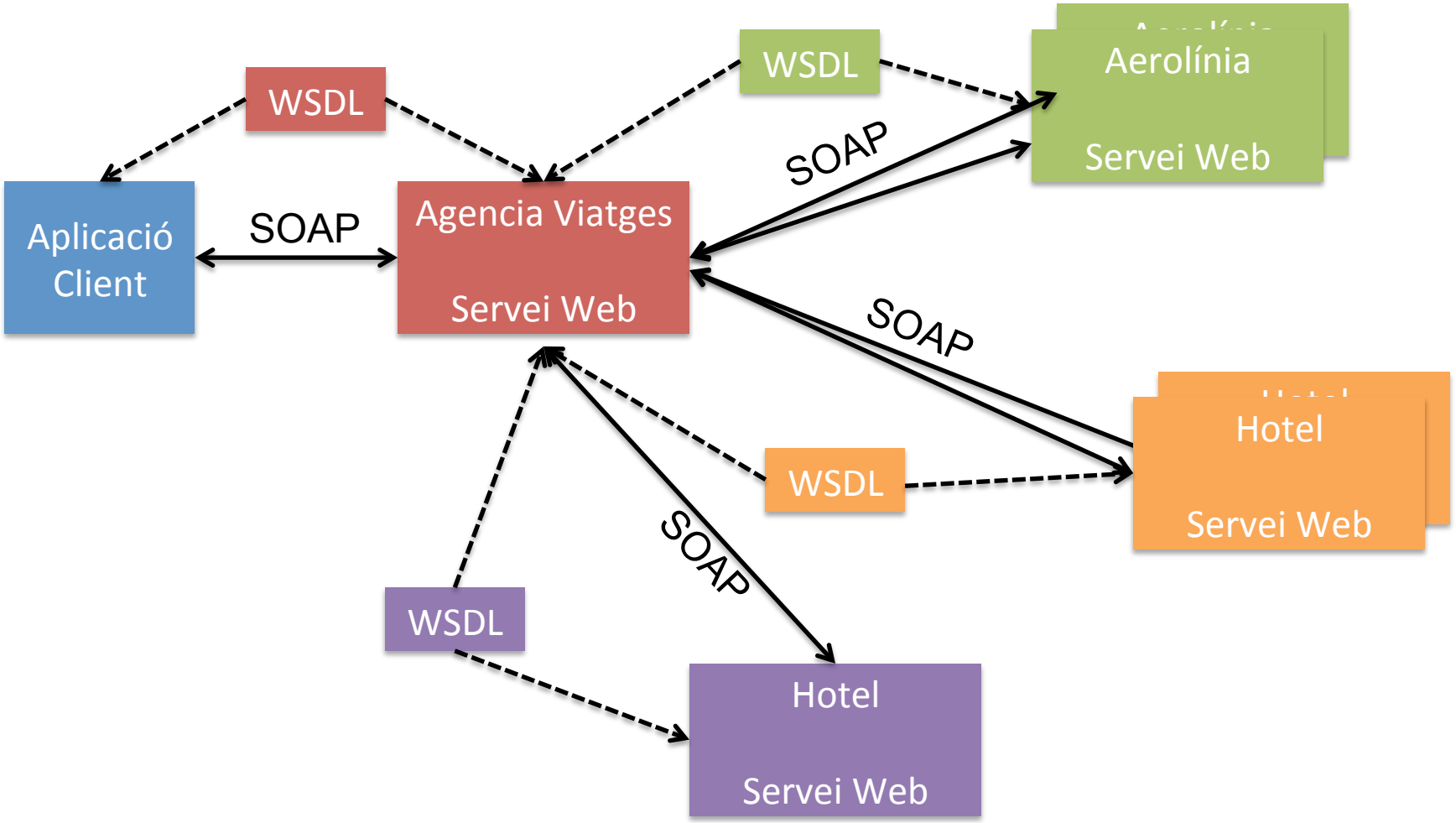


Per poder intercanviar la informació necessària calen bàsicament dues tecnologies: Web Services Description Languages (**WSDL**) i Simple Object Acces Protocol (**SOAP**).

- Web Services Description Languages (**WSDL**): descriu quina és la informació requerida pel servei. No podem saber que fa el servei, però si disposem de la informació necessària per poder interactuar amb ell.

- Simple Object Acces Protocol (**SOAP**): és un protocol basat en XML, què permet la interacció entre diferents dispositius i que té la capacitat de transferir informació complexa.

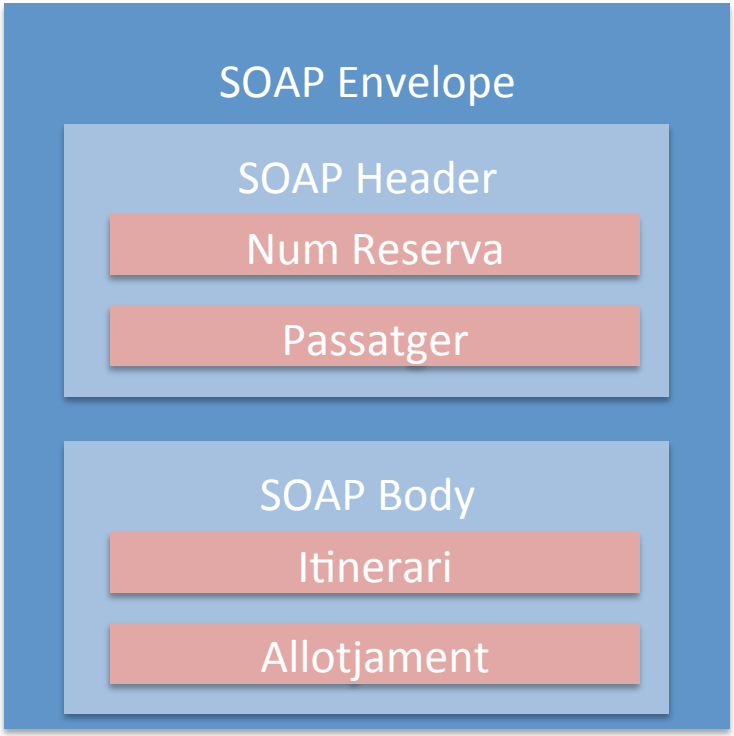
Arquitectura Orientada a Serveis:



Arquitectura Orientada a Serveis:

- Un missatge de tipus SOAP està compost per un **envelope**, on cada envelope té un **header** i un **body**.

Seguint l'exemple anterior l'estructura del missatge de tipus SOAP per sol·licitar un viatge podria ser quelcom semblant a:



Arquitectura Orientada a Serveis:

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
envelope">
 <env:Header>
 <m:reserva xmlns:m="http://empresaviajes.ejemplo.org/reserva"
     env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
     env:mustUnderstand="true">
  <m:referencia>
     uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d
  </m:referencia>
  <m:fechaYHora>2001-11-29T13:20:00.000-05:00</m:fechaYHora>
 </m:reserva>
 <n:pasajero xmlns:n="http://miempresa.ejemplo.com/empleados"
     env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
     env:mustUnderstand="true">
  <n:nombre>Pepe Ejemplo</n:nombre>
 </n:pasajero>
</env:Header>
                                        .
                                        .
                                        .
                                        .
                                        .
                                        .
                                        .
                                        .
                                        .
                                        .
<env:Body>
  <p:itinerario
    xmlns:p="http://empresaviajes.ejemplo.org/reserva/viaje">
   <p:ida>
     <p:salida>Nueva York</p:salida>
     <p:llegada>Los Angeles</p:llegada>
     <p:fechaSalida>2001-12-14</p:fechasalida>
     <p:horaSalida>última hora de la tarde</p:horaSalida>
     <p:preferenciaAsiento>pasillo</p:preferenciaAsiento>
   </p:ida>
   <p:vuelta>
     <p:salida>Los Angeles</p:salida>
     <p:llegada>Nueva York</p:llegada>
     <p:fechaSalida>2001-12-20</p:fechaSalida>
     <p:horaSalida>media-mañana</p:horaSalida>
     <p:preferenciaAsiento/>
   </p:vuelta>
  </p:itinerario>
  <q:alojamiento
    xmlns:q="http://empresaviajes.example.org/reserva/
hoteles">
    <q:preferencia>ninguna</q:preferencia>
  </q:alojamiento>
 </env:Body>
</env:Envelope>
```

Arquitectura Orientada a Serveis:

- Una de les principals problemàtiques una web basada en SOA són:

1. Especificacions de cada servei. Requereix un coneixement elevat de la API del servei sol·licitat.

2. Moltes vegades no estan visibles.

3. Cal mantenir un llistat amb els serveis operatius disponibles.

4. Als costos de desenvolupament. Calen coneixements elevats de tots els elements involucrats SOA.

Arquitectura Orientada Web:

- L'arquitectura orientada a web (WOA) pretén assolir els mateixos objectius que el SOA, però amb menys costos. WOA es pot veure com una evolució del SOA. Les principals diferències entre el WOA i el SOA són:

  1. WOA representa la informació en recursos, SOA en serveis.
  2. Cada recurs esta disponible a la web mitjançant els URI.
  3. HTTP és el llenguatge per accedir als URI.
  4. Manipulació dels URI (recurs) mitjançant: GET, PUT, POST, DELETE, etc. Amb tècnica REST.
  5. WOA només utilitza les propietats del protocol HTTP. SOA utilitza SOAP per l'intercanvi de dades i HTTP com a capa de transport.

¡PREGUNTA D'EXAMEN!

Quina és la funció del Web Service Description Language?

¡PREGUNTA D'EXAMEN!

Simple Object Access Proctocol és un llegnuatge que permet fer crides RPC? Justifica la resposta.

¡PREGUNTA D'EXAMEN!

Quina és la principal diferència entre SOA i WOA? Justificala.

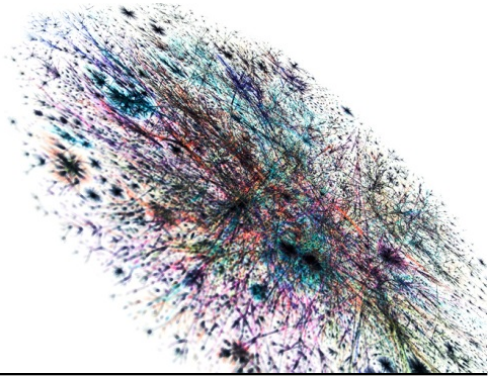
4. DISSENY D'APLICACIONS WEB

4.1 Paradigmes de programació.

4.2 El model vista controlador.

4.3 Virtual Hosting.

4.4 Aplicacions Web per a dispositius mòbils.



Model vista controlador. Descripció i ús.

El Model Vista Controlador (MVC) és un patró d'arquitectura de software que separa Les dades d'una aplicació, interfície d'usuari i la lògica de control en tres components.

S'inicia el 1969 i es públic el 1980.

Es definida per XEROX PARC durant el desenvolupament de *interactive Smalltalk-80 interface*.

Microsoft

Model Vista Controlador

Model vista controlador. Descripció i ús.

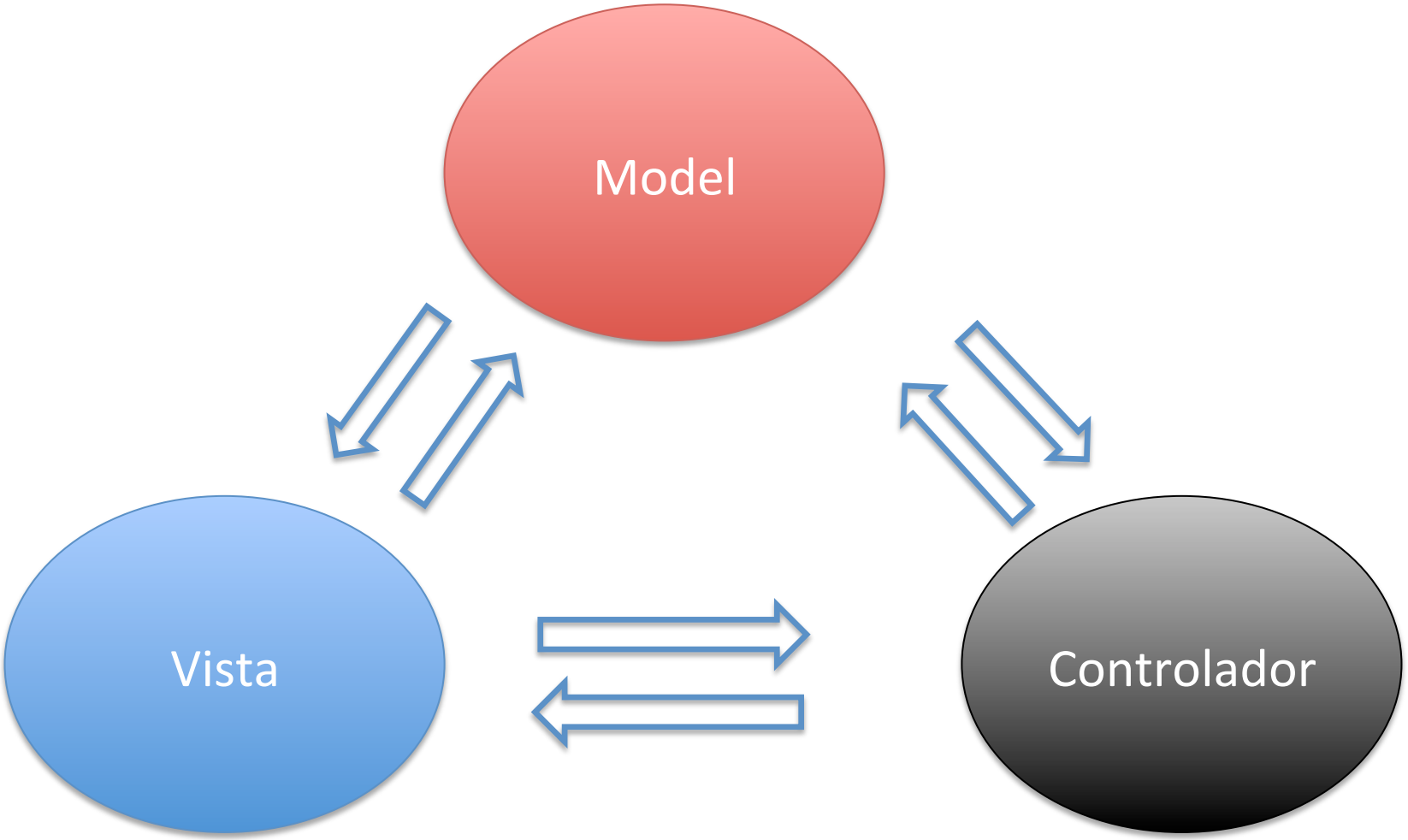
- 1981 PARC presenta el xerox (1969-1981).

- 1983 apareix Apple Lisa (1978-1983).
- 1984 apareix el primer Macintosh (1979-1984).

Microsoft

- 1985 apareix el primer Windows com a complement del MS-DOS (1981-1985).

Model vista controlador. Descripció i ús.



Model vista controlador. Descripció i ús.

Model: Gestiona la informació amb la què treballa el sistema. La lògica permet assegurar la integritat de les dades. Gestió de la BD i del model de negoci.

Vista: part amb la que interactua el usuari, la interfície. El nostre cas la web.

Controlador: respon a events del usuari, impliquen canvis en el model i en la vista.

L'objectiu del MVC és la de crear un model que permeti millorar la reusabilitat gestionant de forma independent la vista i el model.

Model vista controlador. Descripció i ús.

En el cas d'una aplicació web les funcions de cada element són les següents.

Funcions del model:

1) Accedir a les dades emmagatzemades.
2) Definir les regles de negoci.
3) Gestionar els diferents controladors i vistes.
4) Permet que les accions i les vistes siguin independents del model.

Funcions del controlador:

1) Rebre els events d'entrada (un click, inserció de text, etc).
2) Conté regles de gestió d'events (p.e. Si event A acció B). Aquestes accions poden suposar accions al model (p.e sol·licitar els temps d'entrega) o a les vistes (p.e actualitzar la web).

Model vista controlador. Descripció i ús.

Funcions de la vista:

1) Rebre les dades del model i mostrar-les a la vista.
2) Tenir un registre del controlador (la vista normalment instancia el controlador).

Avantatges del MVC:

1) És un model totalment genèric, que pot ser utilitzat en múltiples escenaris.
2) Permet disposar de diferents vistes (PC, dispositiu mòbil) per un mateix model.
3) Es poden generar diferents vistes sense haver de modificar el model.

Model vista controlador. Descripció i ús.

El flux d'una aplicació web seguint el MVC podria ser el següent:

1) El usuari interactua amb la interfície d'usuari (Vista).

2) El controlador rep la notificació del event produït pel usuari. El controlador gestiona el event (normalment amb un handler).

3) El controlador accedeix al model, operant de forma adequada al event sol·licitat pel usuari (p.e actualitzant el carro de la compra del usuari).

4) El controlador delega a la vista la visualització de la informació que ha obtingut en el pas anterior (p.e el carro actualitzat).

5) La vista espera una nova interacció per part del usuari.

Model vista controlador. Exemple.

Aplicació web que mostra el contingut d'una BD utilitzant el MVC.

1. El següent script PHP és connecta a una suposada BD de dades, en aquest cas a la de la UAB, i llista els alumnes i la seva data de naixement.

Model vista controlador. Exemple.

```php
<?php
// Connectar amb la BD i seleccionar la BD desitjada
$connexio = mysql_connect('localhost', 'usuari', 'password');
mysql_select_db('UAB_db', $connexio);
// Executar la consulta SQL
$resultat = mysql_query('SELECT nom, data_naix FROM alumnes', $connexio);
?>
<html>
<head>
<title>Llistat alumnes de la UAB</title>
</head>
<body>
<h1>Llistat d'alumnes</h1>
<table>
<tr><th>Nom</th><th>Data Naixament</th></tr>
<?php
// Mostrar els resultats amb HTML
while ($fila = mysql_fetch_array($resultat, MYSQL_ASSOC))
{
echo "\t<tr>\n";
printf("\t\t<td> %s </td>\n", $fila['nom']);
printf("\t\t<td> %s </td>\n", $fila['data_naix']);
echo "\t</tr>\n";
}
?>
</table>
</body>
</html>
<?php
// tancar la connexio
mysql_close($connexio);
?>
```

Model vista controlador. Exemple.

L'script anterior és fàcil d'escriure i ràpid d'executar, però és díficil de mantenir i actualitzar.

No existeix gestió d'errors. Què passa si falla la connexió amb la BD?

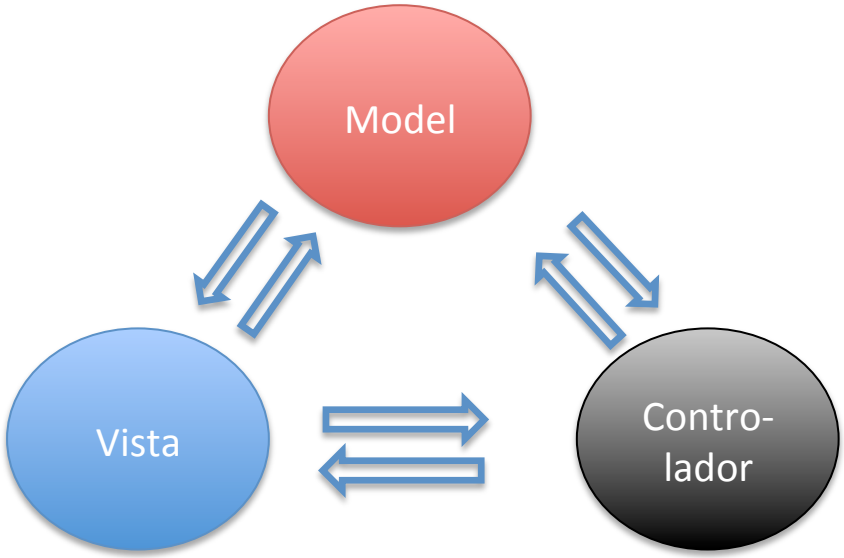
El codi HTML i PHP es troben barrejats en un mateix arxiu. L'accés a la BD i com es mostren les dades al usuari està "maxembrat" en un únic arxiu.

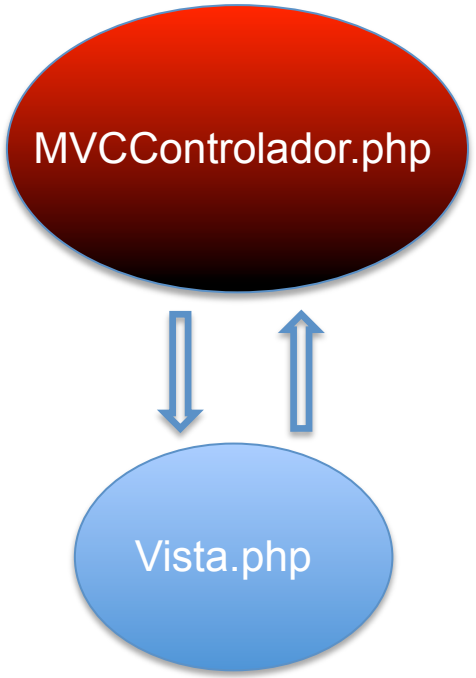
El codi només funciona si la BD és mySQL.

Cal identificar la VISTA, el MODEL i el CONTROLADOR!!!!!!!!!

Model vista controlador. Exemple.

2. Primerament separerem la vista del model i el controlador.





Model vista controlador. Exemple.

MVCControlador.php

```php
<?php
// Connectar amb la BD i seleccionar la BD desitjada
$connexio = mysql_connect('localhost', 'usuari', 'password');
mysql_select_db('UAB_db', $connexio);
// Executar la consulta SQL
$resultat = mysql_query('SELECT nom, data_naix FROM alumnes', $connexio);
// Crear un array d'elements que enviarem a la vista amb el contigut de la
// BD que volem seleccionar.
$alumnes = array();
while ($fila = mysql_fetch_array($resultat, MYSQL_ASSOC))
{
    $alumnes[] = $fila;
}
// Tanquem la conexión
mysql_close($connexio);
// Incluir la lógica de la vista
require('vista.php');
?>
```

Vista.php

```php
<html>
<head>
<title>Llistat alumnes de la UAB</title>
</head>
<body>
<h1>Llistat d'alumnes</h1>
<table>
<tr><th>Nom</th><th>Data Naixament</th></tr>
<?php foreach ($alumnes as $alumne): ?>
<tr>
<td><?php echo $alumne['nom'] ?></td>
<td><?php echo $alumne['data_naix'] ?></td>
</tr>
<?php endforeach; ?>
</table>
</body>
</html>
```

Model vista controlador. Exemple.

La vista ha de contenir el mínim codi en PHP possible, de manera que un dissenyador web sense coneixements de PHP la pugui entendre.

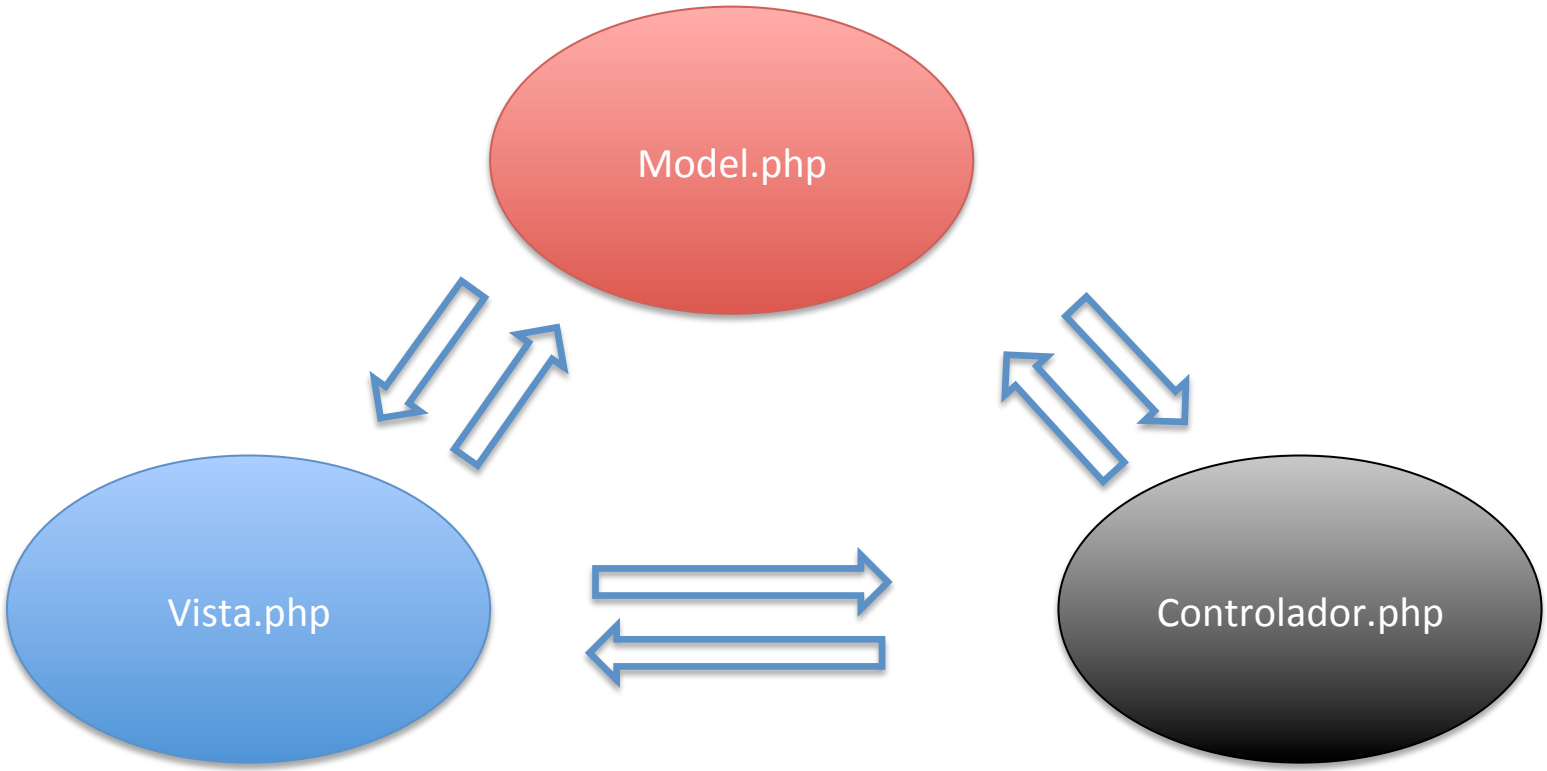
Com a molt poden haver if/else, foreach/enforeach.

No hi hauran mai comandes PHP que generin tags HTML.

Que passa si hem d'accedir a una altre BD per fer una altre consulta?

El controlador s´ha d'encarregar de sol·licitar les dades al model i fer-les accessibles a la vista.

Model vista controlador. Exemple.



Model vista controlador. Exemple.

Model.php

```php
<?php
function getTotsElsAlumnes()
{
// Connectar amb la BD i seleccionar la BD desitjada
$connexio = mysql_connect('localhost', 'usuari', 'password');
mysql_select_db('UAB_db', $connexio);
// Executar la consulta SQL
$resultat = mysql_query('SELECT nom, data_naix FROM alumnes', $connexio);
// Crear un array d'elements per la vista
$alumnes = array();
while ($fila = mysql_fetch_array($resultat, MYSQL_ASSOC))
{
$alumnes[] = $fila;
}
// Tancar la connexio
mysql_close($connexio);
return $alumnes;
}
?>
```

Vista.php

```php
<html>
<head>
<title>Llistat alumnes de la UAB</title>
</head>
<body>
<h1>Llistat d'alumnes</h1>
<table>
<tr><th>Nom</th><th>Data Naixament</th></tr>
<?php foreach ($alumnes as $alumne): ?>
<tr>
<td><?php echo $alumne['nom'] ?></td>
<td><?php echo $alumne['data_naix'] ?></td>
</tr>
<?php endforeach; ?>
</table>
</body>
</html>
```

Controlador.php

```php
<?php
// Incluir la lógica del model
require_once('model.php');
// Obtenim la llista dels alumnes
$alumnes = getTotsElsAlumnes();
// Incluir la lògica de la vista
require('vista.php');
?>
```

Model vista controlador. Exemple.

El **controlador** només s'encarrega d'obtenir les dades del model i fer-los arribar a la vista. En aplicacions més complexes el controlador també s'encarrega de processar les peticions, sessions, autentificacions, etc.

El **model** únicament s'encarrega d'accedir a les dades. Tots els paràmetres que no depenen del model de dades s'obtenen a través del controlador. Les funcions del model poder ser reutilitzades en altres controladors.

La **vista** es sol separar en diferents arxius diferenciant entre la **plantilla** i el **layout.** El **layout** conté la part comuna de l'aplicació o d'un conjunt de webs, sol contenir la capçalera i el peu de pàgina. La **plantilla** mostra de forma correcta el contingut en funció del dispositiu de visualització o de les dades que estem visualitzant.

¡PREGUNTA D'EXAMEN!

Quines avantatges aporta la utilització del model vista controlador?

¡PREGUNTA D'EXAMEN!

El MVC potser utilitzat en llenguatges Procedurals o Orientat a Objectes? Raona la resposta.

¡PREGUNTA D'EXAMEN!

Tenim una aplicació web dissenyada seguint el MVC i ens demanen que la fem accessible també per a dispositius mòbils? Què haurem de modificar?

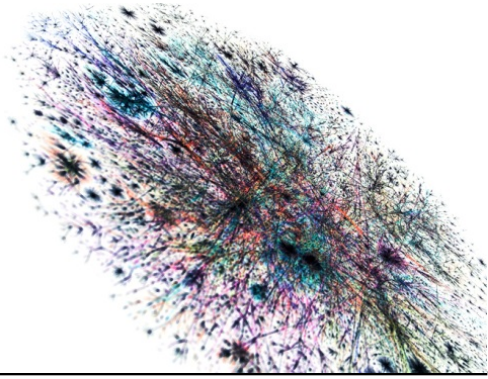
4. DISSENY D'APLICACIONS WEB

4.1 Paradigmes de programació.

4.2 El model vista controlador.

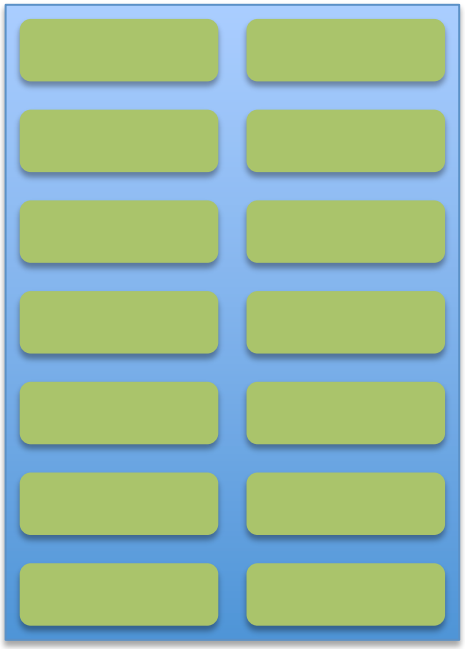
4.3 Virtual Hosting.

4.4 Aplicacions Web per a dispositius mòbils.



Virtual Hosting

És la capacitat que un servidor web té d'allotjar múltiples web. Cada web està allotjada en la seva pròpia localització/partició, per mantenir-la separada de les altres webs allotjades.



Com podem accedir a les diferents webs?

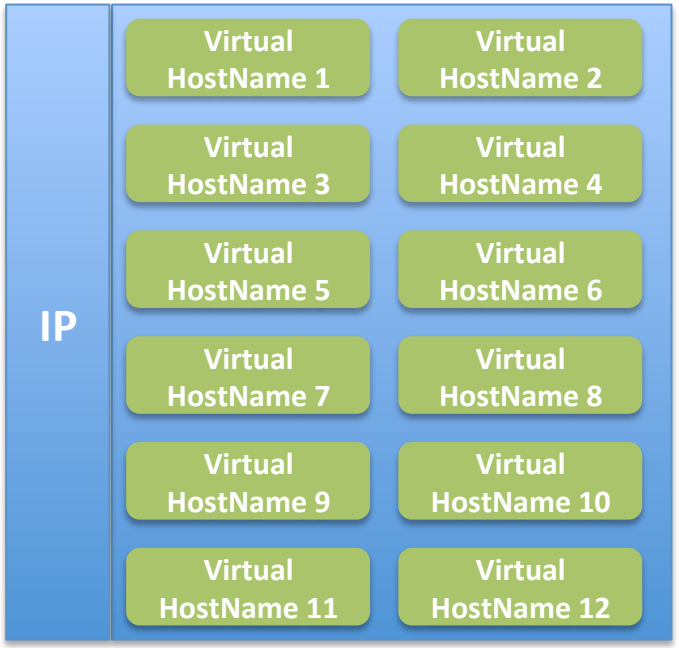
1. Name-Based
2. IP-Based

Virtual Hosting

2. **Name-Based**:

En aquest cas diferents web comparteixen una mateixa IP. Això es possible ja que el protocol HTTP quan fa la petició d'una IP inclou el Hostname.

Client



Servidor

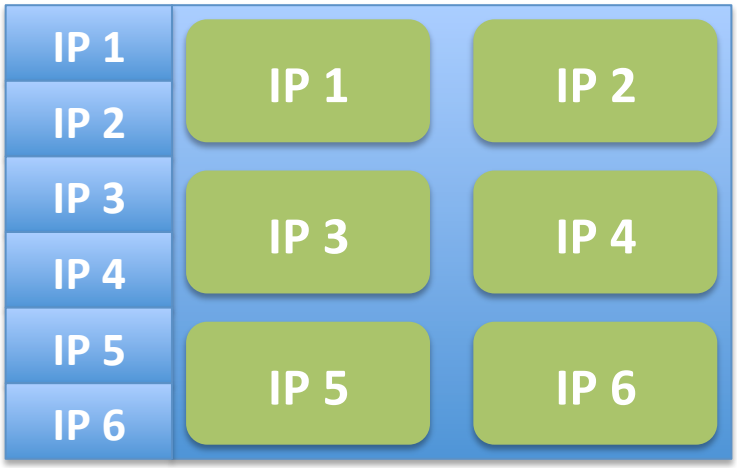
Virtual Hosting

1. IP-Based:

Cada pàgina web o virtual host té la seva pròpia IP. El servidor web és configura amb diferents interfícies de xarxa. El servidor web utilitza la adreça IP sol·licitada per retornar la web corresponent allotjada en el servidor.

El problema d'aquesta estratègia és la poca disponibilitat d'adreces IPv4.

L' utilització d'aquesta estratègia ve motiva per aspectes de seguretat. Els certificats de seguretat SSL estan vinculats a adreces IP.



¡PREGUNTA D'EXAMEN!

Un servidor conté múltiples webs. Totes elles poden compartir el mateix certificat de seguretat. Quina opció triaries per fer virtual hosting? Justifica la resposta.

¡PREGUNTA D'EXAMEN!

Un servidor fa name-based virtual hosting, de que depèn el número màxim de webs que pot allotjar.

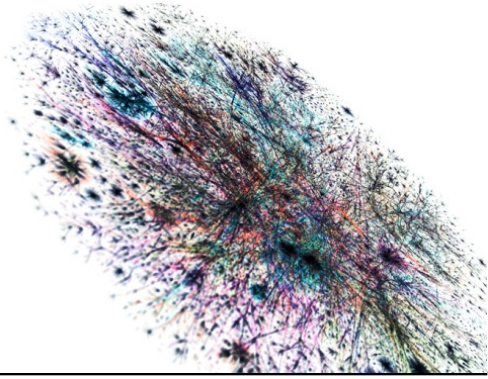
4. DISSENY D'APLICACIONS WEB

4.1 Paradigmes de programació.

4.2 El model vista controlador.

4.3 Virtual Hosting.

4.4 Aplicacions Web per a dispositius mòbils.

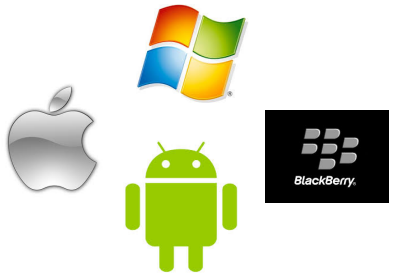


Aplicacions Web per a dispositius mòbils.

En el moment de comença a desenvolupar una aplicació web per a dispositius mòbil la primera pregunta que ens fem és:

Una aplicació nativa o una aplicació web basada en HTML5?

Cadascuna té les seves avantatges, però la elecció equivocada pot tenir uns costos molt elevats!



Aplicació desenvolupada específicament per un dispositiu què disposa de sistema operatiu propi.

Aplicació desenvolupada per ser executada mitjançant un navegador, és independent del sistema operatiu del dispositiu mòbil.

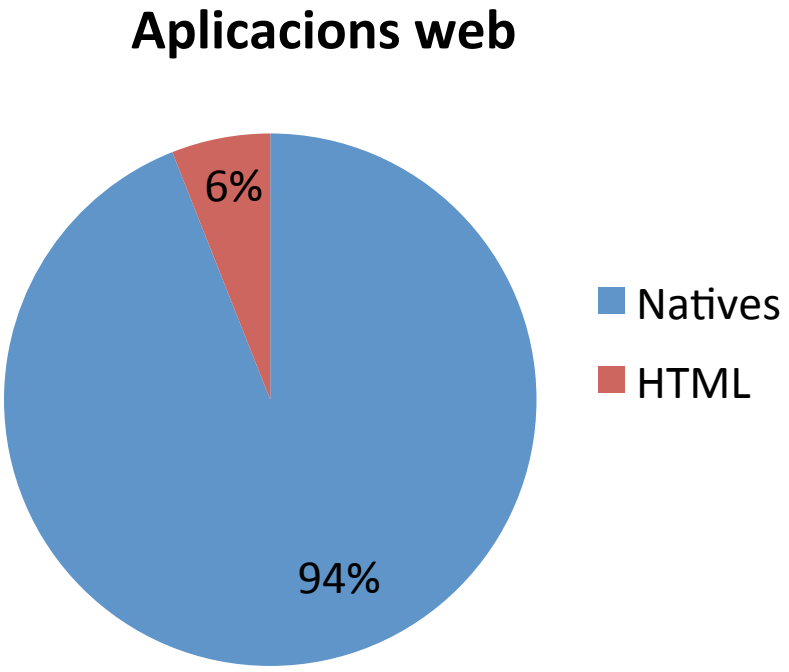
Aplicacions Web per a dispositius mòbils.

Actualment (any 2012) en el mercat hi ha al voltant de 1.25 milions d'aplicacions.

650.000 iOS

500.000 Android

100.000 Microsoft



Aplicacions Web per a dispositius mòbils.

Que hem de tenir present quan es vol desenvolupar una aplicació web?

Experiència del usuari

Rendibilitat

Portabilitat

Control de distribució

Seguretat

Aplicacions Web per a dispositius mòbils.

Experiència del usuari

Només les aplicacions natives poder aportar un experiència òptima al usuari, explotant al màxim totes les prestacions del dispositiu i del sistema operatiu i així oferir les tècniques més recents per fer les tasques desitjades.

La gent de Html5 defensa que l'experiència del usuari es prou bona amb Html5.

Una experiència per l'usuari prou bona és suficient o volem la millor?

Aplicacions Web per a dispositius mòbils.

Rendibilitat

App store de Apple és el mercat més gran d'aplicacions a Internet.

- Amb més de 400 milions de comptes vinculades a targetes de crèdit.
- Més de 5000 milions de $ pagats als desenvolupadors.

Html5 no disposa d'una app store per poder rendabilitzar l'aplicació. La manca d'un mercat específic fa que hi hagin menys desenvolupadors.

Aplicacions Web per a dispositius mòbils.

Portabilitat

Les aplicacions natives requereixen d'un nou desenvolupament per cada plataforma.

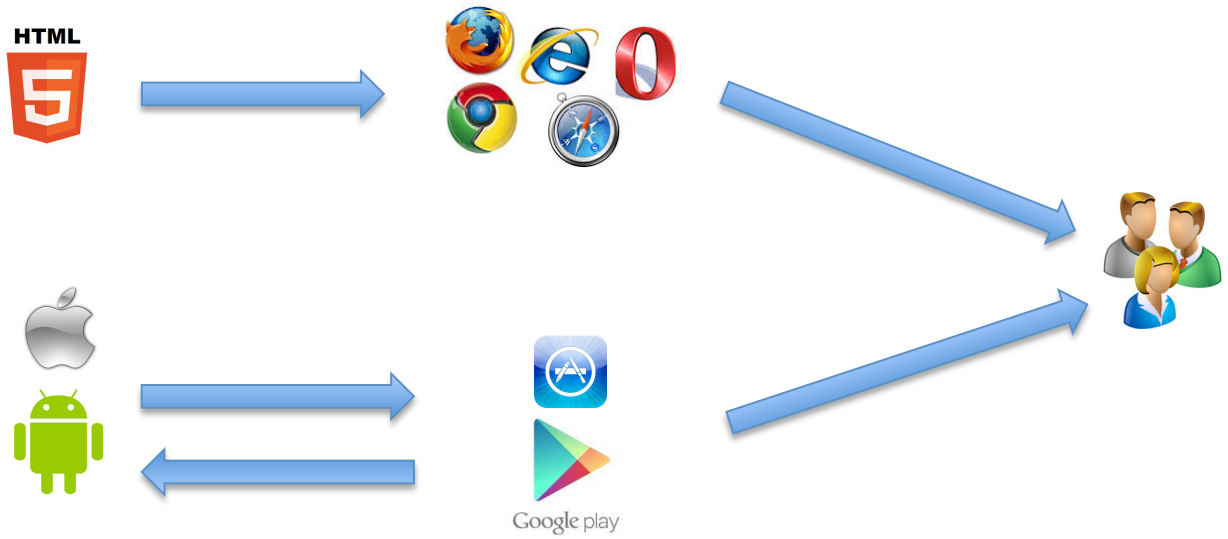
Html5 ofereix la prestació de: "desenvolupa una vegada, executa'l en qualsevol plataforma".

Degut al gran ventall de navegadors per a dispositius mòbils i les seves versions, no sempre actualitzades per part dels usuaris fa que el: "desenvolupa una vegada, executa'l en qualsevol plataforma" sigui més aviat un: "desenvolupa una vegada, optimitza'l optimitza'l optimitza'l..."

Aplicacions Web per a dispositius mòbils.

Control de distribució

Html 5 permet als desenvolupadors tenir un control complet de la distribució de l'aplicació, permeten accés immediat als usuaris via els navegadors; amb comparació a altres sistemes (app store o google play) que han de passar controls més rigorosos.



Una versió anualment amb noves prestacions.

5 anys per tenir una versió acceptada per w3c.

Aplicacions Web per a dispositius mòbils.

Seguretat

El codi Html5 és més fàcil d'accedir-hi i corrompre'l.

Les dades que es troben en cache poden ser assegurades i encriptades amb aplicacions natives, però no en aplicacions web en Html5.

Les transmissions (downloading i uploading) poden ser segures tant en aplicacions Html5 com natives, però en Html5 cal utilitzar el protocol https, penalitzant la velocitat de transmissió.

Aplicacions Web per a dispositius mòbils.

Quan els usuaris no volen descarregar l'aplicació.

Quan la informació s'actualitza constantment.

Si no volem estar controlat per terceres entitats (app store, google play, etc.).

Desenvolupament ràpid per varies plataformes.

Aportar la millor experiència al usuari.

Quan es vol obtenir una rendibilitat de l'aplicació.

Quan totes les dades han de ser segures.