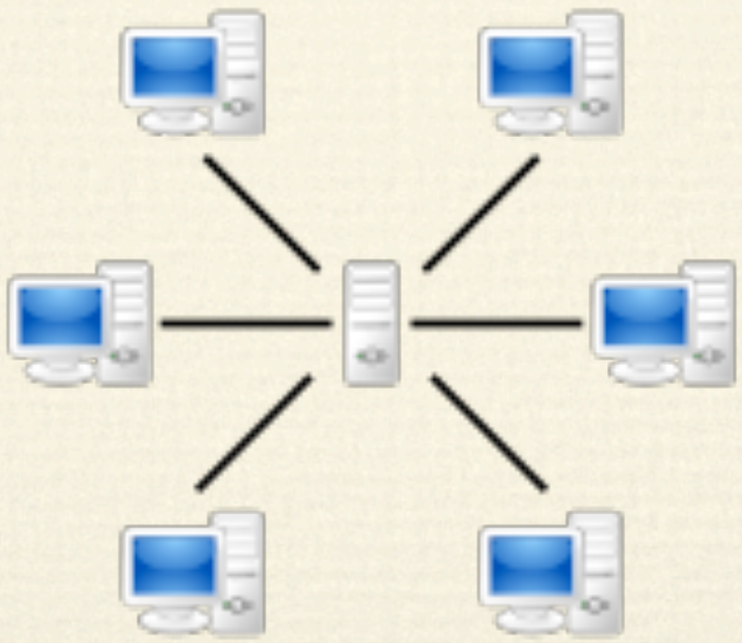
5. Applications web

M1. Outils de l'Internet
lundi 18 octobre 2010

victor.poupet@lif.univ-mrs.fr

I. Contexte

Client-serveur



Description

- Architecture d'applications
  - Entre 2 parties distinctes : un client et un serveur
  - À travers un réseau informatique ou localement

Graphe

- Version simple : uniquement des clients et serveurs (*two-tier*)
- Version complexe : d'autres nœuds (n-tier, MVC, etc.)
- Éventuellement structure récursive (le serveur interroge un second serveur, etc.)

Exemples

- World Wide Web
- Bases de données
- FTP
- Domain Name System
- Mail (partiellement)
- Jeux en ligne (serveurs temporaires ou dédiés)

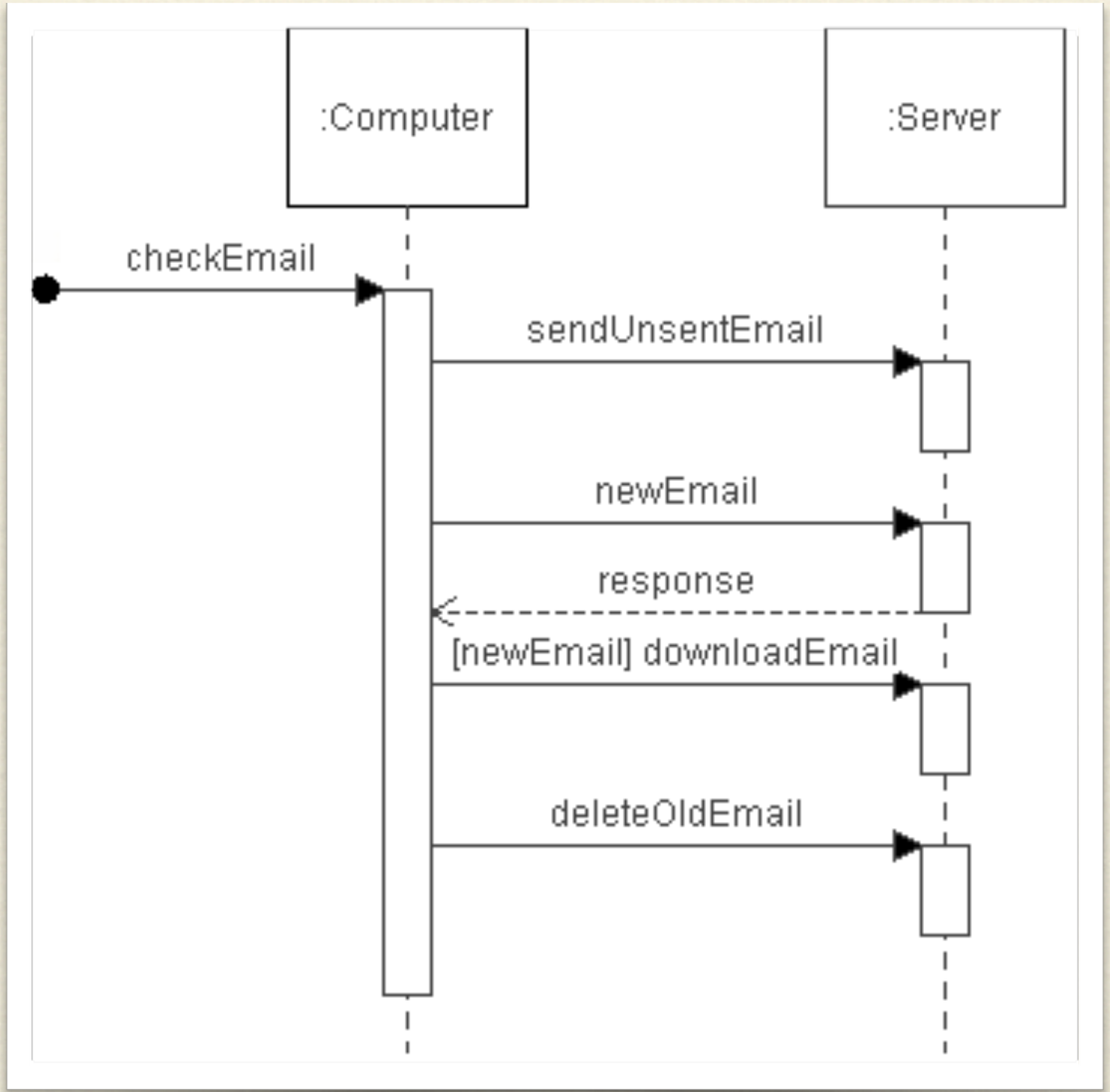
Fonctionnement (client)

- Emet une requête
- Attend et reçoit les réponses
- Peu de connexions en même temps
- Communication directe avec l'utilisateur (GUI)
- Ouvre et ferme la connexion

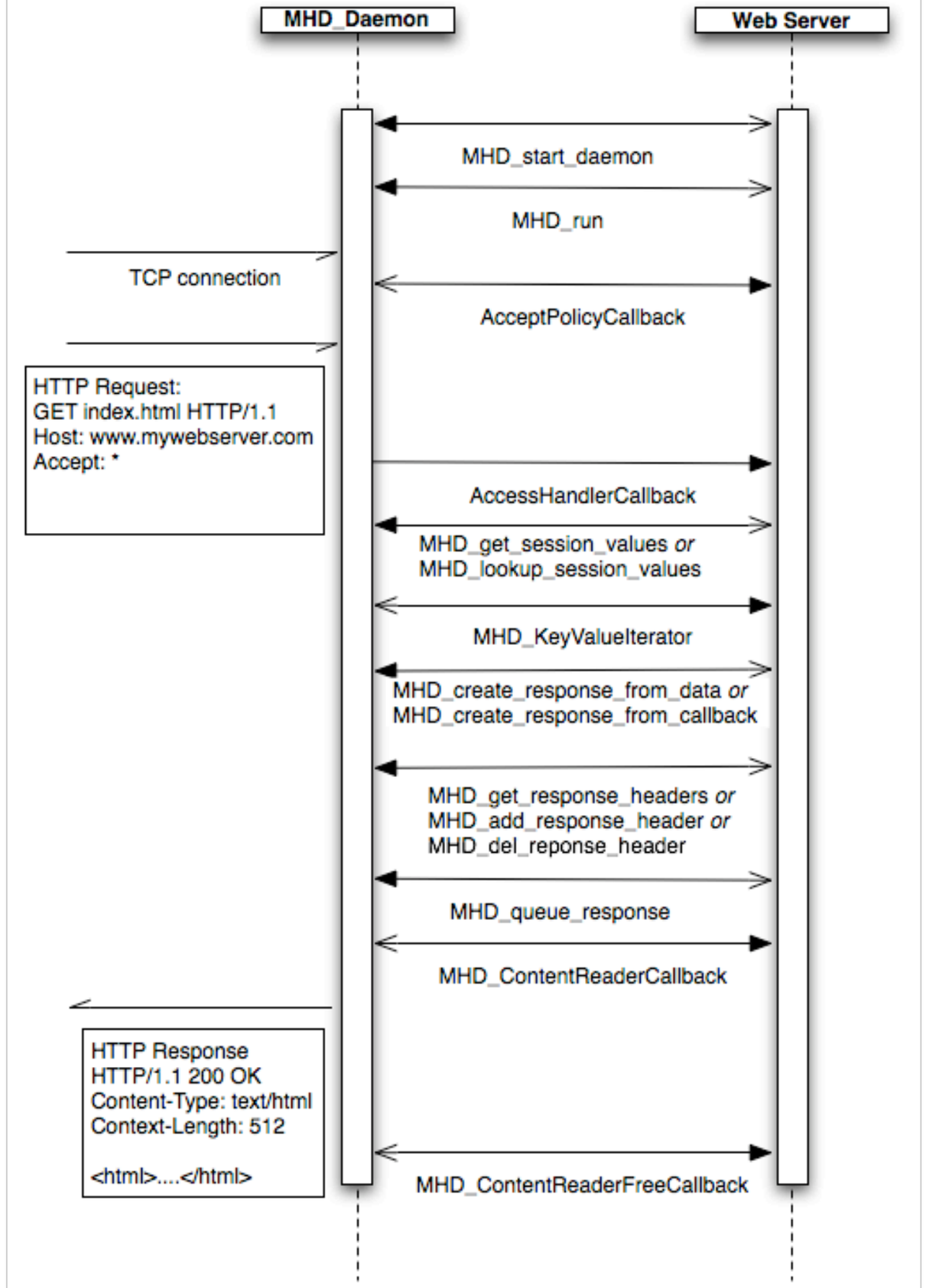
Fonctionnement (serveur)

- Passif (*slave*)
- Attend les requêtes des clients
- Traite les requêtes et envoie les réponses
- Accepte de nombreuses connexions
- Ne communique pas directement avec l'utilisateur

Diagrammes UML



source : http://en.wikipedia.org/wiki/Sequence_diagram



source : http://web.cs.du.edu/~chrisg/classes/06/comp3621/check3.html

Points forts

- Possibilité de répartir les différentes fonctions d'un système sur un réseau : encapsulation (facilité de mise à jour/modification/réparation)
- Les données sont sur les serveurs : sécurité
- Facilité de mettre à jour les données

Points forts

- Les modifications sont propagées rapidement
- Un même serveur peut fonctionner avec différents clients
- Technologie bien connue et très répandue (donc facile à développer)

Points faibles

- Risque d'encombrement du réseau
- Manque de robustesse

Clients légers (*thin clients*)

Idée

- Le client n'effectue presque aucun calcul
- Unique rôle : intermédiaire entre l'utilisateur et le serveur
- Exemples : navigateur web, bureau virtuel, etc.
- Pas de disque dur

Histoire

- Très gros ordinateurs, accès à distance par des terminaux
- Début des années 90 : terminaux X
- Initialement appelés “terminaux graphiques” (les terminaux “pur texte” sont également des clients légers...)
- Renommé par la suite (plus attrayant)

Côté logiciel

- Fonctionne sans disque local
- Conçu pour être le plus simple possible
- Laisse la charge de traitement au serveur
- Certains clients légers fonctionnent en réalité sur des PC (ex : pilotlinux)

Côté matériel

- Pas de disque dur
- Matériel minimaliste (uniquement capable de lancer une application client léger)
- Client ultra léger : client de connexion réseau en matériel (sans OS)

Fonctionnement

- OS contenu dans de la mémoire flash ou DOM
- Mémoire protégée en écriture (accès aux admins)
- Aucune installation n’est nécessaire sur le client
- Possibilité de préparer des images de boot

Mémoire flash

- Plus résistant qu'un disque dur (pas de partie mécanique)
- Consomme peu d'énergie
- En cas de dégât physique les informations ne peuvent pas être récupérées
- Plus cher à fabriquer

Interface

- Sortie : écran
- Entrée : clavier, souris, ...
- Lecteurs virtuels sur le réseau
- Lecteurs de ROM (CD, DVD)

Avantages

- Facilité d'administration (uniquement sur le serveur)
- Protection contre les attaques logicielles
- Sécurité des informations : rien sur le client
- Economique : clients durables
- Partage de mémoire (une application pour plusieurs clients)

Avantages (2)

- Faible consommation
- Echange de client en cas de panne (transparence)
- Pas de perte d'information en cas de perte du client
- Peu de valeur (moins de vol)
- Résistance aux environnements hostiles (pas de pièces mobiles ni ventilation)

Avantages (3)

- Faible utilisation de la bande passante (mouvements de l'utilisateur et mises à jour de l'écran, très compressibles)
- Meilleure répartition de l'utilisation des ressources
- Extensibilité
- Moins de bruit (pas de DD, pas de ventilateur)
- Ecologique (matériel dure plus longtemps)

Inconvénients

- Besoin d'une bonne performance du serveur
- Peu adapté au multimedia (videos ou jeux)
- Parfois peu compatible avec les applications pour clients ayant leurs propres ressources (conflits de bibliothèques)

Incovénients (2)

- Dépendant de la connexion réseau
- Difficulté d'ajouter des périphériques
- Utilisation des ressources par un utilisateur peut handicaper les autres

Clients hybrides

- Pas de disque dur (ni de ventilation)
- Le système d'exploitation et les données sont sur le serveur
- Le traitement est fait localement

Mises à jour

- L'image disque est lue par toutes les machines (1:N)
- Elle peut être accédée en écriture par un administrateur sur un client (1:1)
- L'administrateur modifie l'image sans affecter les machines qui ont déjà démarré
- Lorsque les clients démarreront ils prendront la nouvelle image

Avantages (vs. clients légers)

- Pas de charge sur le serveur (sauf les accès disque)
- Bonne réactivité (multimedia)
- Facilité de gestion des périphériques externes

Inconvénients (vs. clients légers)

- Plus chers
- Pas *future-proof*
- Le trafic réseau est plus important

Model-View-Controller

Présentation

- Modèle architectural (1979, Xerox)
- Séparation de l'interface (*view*) et les données (*model*)
- Nécessité de pouvoir modifier l'un sans affecter l'autre : introduction d'un élément intermédiaire
- Logique de traitement (*controller*)

Description : *model*

- Représentation des informations de l'application
- Données brutes (BDD)
- Logique sémantique (donne un sens aux données)
- Traitement des données (interprétation)
- La couche d'accès aux données est incluse dans le modèle

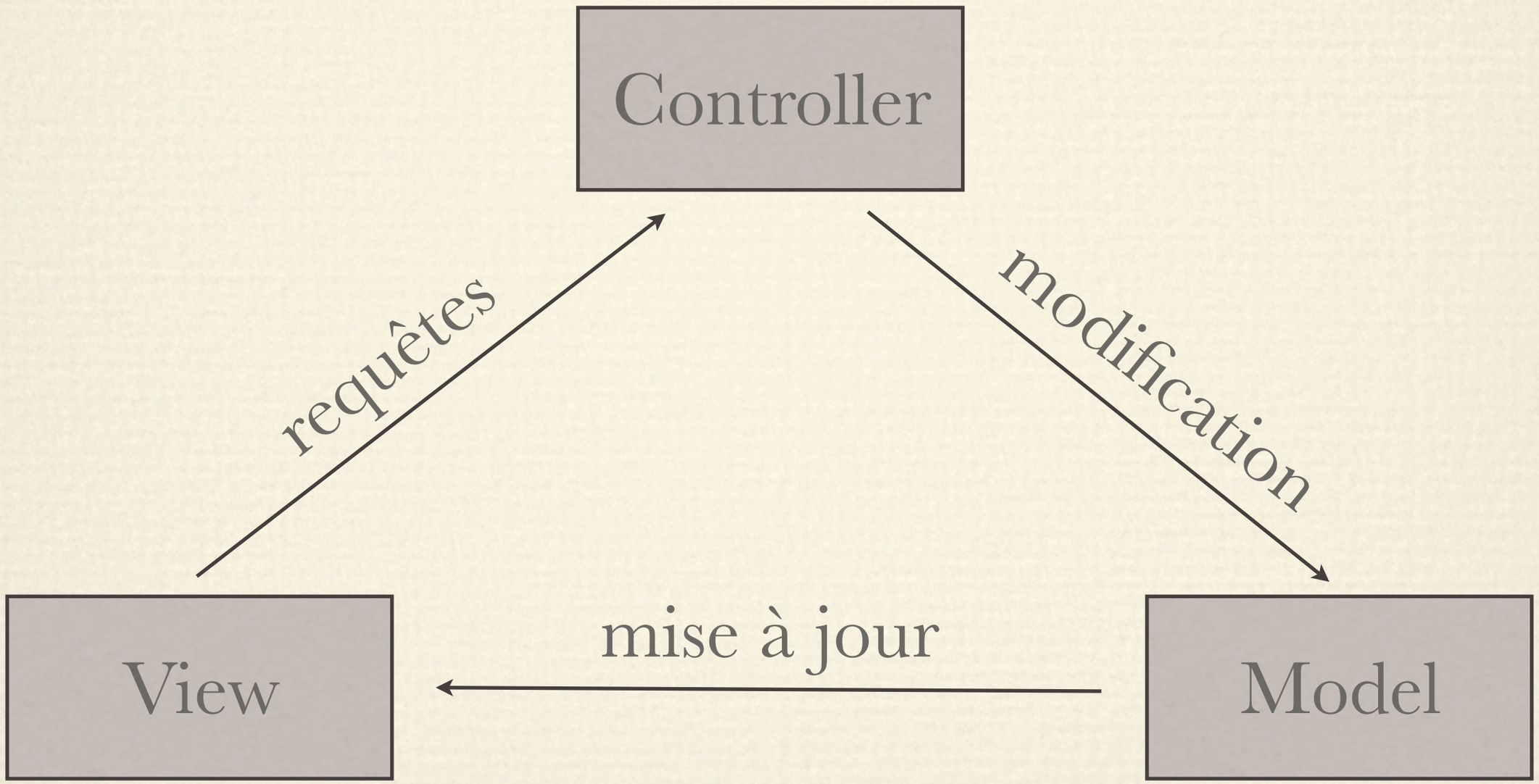
Description : *view*

- Interface utilisateur (UI)
- Présentation du modèle
- Plusieurs vues différentes peuvent correspondre à un même modèle

Description : *controller*

- Traite les instructions de l'utilisateur
- Lit et modifie le modèle

Vue d'ensemble



Exemple

- Un site web avec scripts
- Vue : pages html générées dynamiquement
- Contrôleur : serveur web (reçoit requetes)
- Modèle : base de données et scripts qui produisent la sortie

Intérêt

- Modèle architectural
  - Permet de développer séparément chaque partie
  - Simplifie la conception
  - Facilite la modification

n-Tier

Présentation

- Architecture client-serveur
- Modèle de développement d’applications
- La plupart du temps : 3-tier

3 couches

- Présentation : interface utilisateur (UI)
- Logique : application (Business Logic)
- Informations : base de données
- Les trois couches sont bien distinguées (souvent même sur des machines différentes)

1. Interface

- Interaction avec l'utilisateur
- Traduction homme/machine
- Transmet les requêtes à la couche logique
- Principalement statique
- Localisée sur le PC ou la station de travail de l'utilisateur (navigateur ou application spécialisée)

2. Logique

- Partie programmable de l'application
- Reçoit les requêtes de l'UI et lui transmet les réponses
- Génère dynamiquement les réponses
- Communique avec la BDD
- Localisée sur le serveur
- Peut être subdivisée → *n*-tier

3. Informations

- Ne communique qu'avec la couche logique
- Contient la BDD et le système de gestion associé
- Localisée sur un serveur de BDD

Structure

Presentation tier

The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.

source : http://en.wikipedia.org/wiki/N_tier

Différence avec MVC

- Structure linéaire (pas triangulaire)
- Aucune communication directe entre la présentation et les données
- 3-tier (90s) : inspiré des systèmes distribués (fonctionnement sur des machines différentes)
- MVC (70s) : application sur une seule machine

Utilisation sur le web

- Très adapté aux plate-formes commerciales :
  1. Interface graphique statique (navigateur)
  2. Traitement dynamique des requêtes, interrogation de la BDD et génération des pages pour l'UI
  3. Gestion de la BDD commerciale

Avantages

- L'application est structurée : simplicité de développement
- Les couches sont physiquement et logiquement indépendantes
- Possibilité de modifier une couche sans affecter les autres (encapsulation)
- Seule l'UI dépend du client
- Facilement extensible (*scalable*)

Inconvénients

- Augmente le trafic sur le réseau
- Plus difficile à tester qu'une application 2-tier

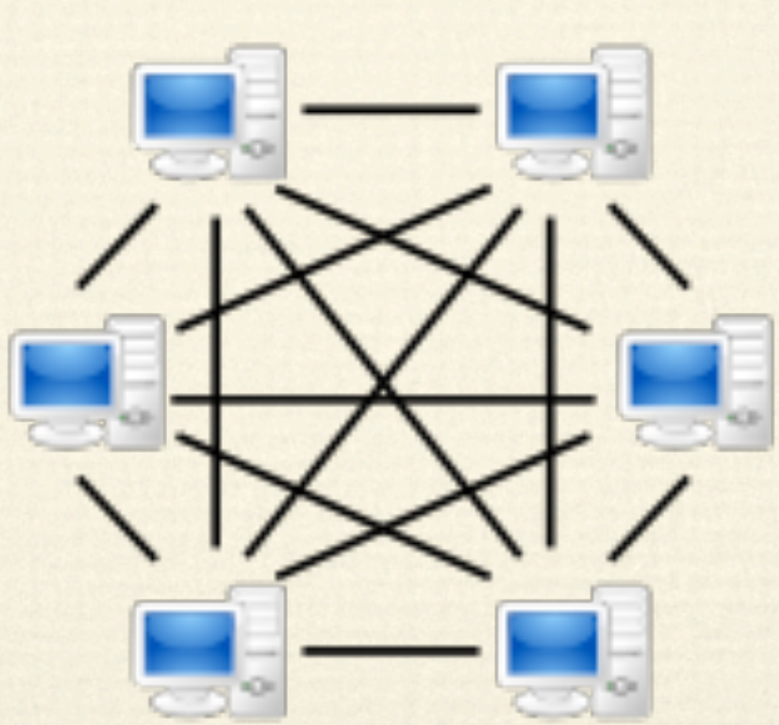
(parenthèse :) Extensibilité

- *Verticale* : augmentation de la puissance ou capacité d'un nœud
  - facile à gérer
  - cher
- *Horizontale* : ajout de nœuds dans le réseau
  - problèmes de latence (communication)
  - nécessité de paralléliser

Communication

- Choisir le protocole de communication entre les couches
- Par fichiers, connexion, etc.
- Exemples : SNMP, CORBA, Java RMI, sockets, UDP, etc.
- Choix important : possibilité d'utiliser des librairies existantes, compatibilité, performances, sécurité, robustesse, simplicité, etc.

Peer-to-Peer (P2P)



Description

- Rôle symétrique des nœuds du réseau
- Le graphe ressemble à une clique (et non pas une étoile)
- Pas de client ou serveur : seulement des *pairs*
- Vision initiale du Web par Tim Berners-Lee

Idée de base

- Les nœuds apportent des ressources (mémoire, processeur, bande passante, etc.) au réseau
- Plus il y a de machines, plus le réseau dispose de ressources

→ Naturellement extensible

Utilisations

- Usenet (premier réseau P2P)
- E-mail (SMTP, partiellement)
- Partage de fichiers
- Téléphonie
- Télévision

Recherche

- La topologie du réseau est difficile à connaître
- Elle varie souvent
- Il n'est pas facile de trouver une information sur le réseau...

P2P centralisé

- Un serveur central donne des informations aux pairs
- Les données sont principalement chez les pairs
- Les pairs informent le serveur des données qu'ils possèdent
- Le routage est partiellement centralisé
- Exemple : Napster (version originale)

P2P centralisé

- Un côté complètement client-serveur
- Les gros échanges de données se font entre les pairs
- Le réseau n'est pas robuste

Napster

- Partage de fichiers de musique
- Entre juin 1999 et juillet 2001
- Initialement seulement pour Windows

Napster

- En 2000 le groupe Metallica porte plainte contre Napster
- D'autres musiciens suivent
- La condamnation fait fermer le serveur central...
- Aujourd'hui le nom et le logo correspondent à un service de musique en ligne payant

P2P structuré

- Les données d'indexation sont réparties sur les pairs
- En général on utilise des tables de hachage
- Chaque nœud est responsable d'une partie de l'indexation
- Redondance des responsabilités (pour être robuste et dynamique)
- Exemple : Freenet

P2P structuré

- Le réseau de connaissances des nœuds est complexe
- Compliqué à maintenir et à gérer dynamiquement
- La recherche est efficace

Tables de hachage distribuées

- Hash : mot clé → chaîne de caractères
- Lorsqu'un nœud partage un fichier, il prévient les nœuds responsables des hashs des mots-clés correspondant
- La recherche se fait en interrogeant les nœuds responsables

Freenet

- Développé par Ian Clarke (répandu depuis 2000)
- Réseau P2P anonyme pour assurer la liberté d'expression
- Les données sont cryptées et réparties sur le réseau (chaque nœud offre une certaine place mais ne contrôle pas le contenu)

Freenet

- Les données uploadées par un utilisateur sont fractionnées et réparties sur le réseau
- La recherche se fait par une interface web (FProxy)
- Les transferts ne se font pas directement mais en passant par des intermédiaires pour maintenir la confidentialité

P2P pur

- Pas de serveur central
- Pas de routage centralisé
- Exemple : Gnutella (jusqu'à la version 0.4)

Fonctionnement

- Nouveaux nœuds copient une liste de nœuds, puis construisent leurs liens
- Facile à mettre en place
- Difficulté de recherche d'une information (flood)

Gnutella

- Partage de fichier en P2P pur
- Initialement développé par Nullsoft en 2000
- Interdit par AOL (qui a racheté Nullsoft), le protocole a été refait par *reverse-engineering*
- Aujourd'hui, *Gnutella* désigne le protocole

Gnutella (v0.4)

- Chaque nouveau nœud se connecte à un nombre déterminé d'autres nœuds (environ 5)
- Recherche par flood
- On limite la profondeur d'une requête (max : 7)

Gnutella (v0.6)

- Des *feuilles* et des *ultra-nœuds*
- Une feuille se connecte à 3 ultra-nœuds
- Un ultra-nœud se connecte à plus de 32 autres ultra-nœuds
- Chaque feuille transmet la liste des fichiers disponibles (en pratique, des hash de mots-clés)
- Chaque ultra-nœud transmet sa liste à ses voisins

Gnutella (v0.6)

- Une requête est transmise au plus 4 fois
- La liaison entre les deux machines est établie (éventuellement en passant par un ultra-nœud pour éviter les pare-feux)

Avantages du P2P

- L'ajout de nœuds augmente la capacité du réseau (mémoire, puissance et bande passante)
- Redondance des données
- Tolérance aux pannes (principalement P2P décentralisé)

II. Applications web

Définition

- Application disponible par le web (Internet ou intranet)
- Pas d'installation côté client

Exemples

- Webmail
- Wikis
- Forums de discussion
- Blogs
- Jeux
- ...

D'habitude

- Côté serveur + côté client
- Modification serveur oblige une modification client (donc bcp de modifications !)
- Difficulté technique (nécessité d'être rétro-compatible)
- Coût supplémentaire

Avec les applications web

- Rien du côté client
- Le serveur génère dynamiquement les pages
- Possibilité dynamiques chez le client : Javascript, Ajax, etc.
- Formats standards (HTML/XHTML)
- Compatibilité facile

Interface

- Beaucoup de possibilités (grâce aux langages dynamiques côté client) :
  - interaction clavier, souris, ...
  - multimedia
  - possibilité d'imiter les OS (glisser/déposer)
  - ...

Indépendance

- L'application est exécutée par le navigateur
- Standards indépendants de l'OS
- Mais parfois les standards sont mal respectés :
  - mauvaise représentation
  - conflits avec les paramètres du navigateur

Structure

- Couramment 3-tier :
  - Navigateur
  - Technologie dynamique (Python, ASP, JSP/Java, CGI, etc.)
  - Base de données

Application Service Provider

- Présenter une application classique sous forme d'application web
- Simplicité pour l'utilisateur (pas d'installation)
- Mise à jour simplifiée
- Application gratuite ou payante sous forme d'abonnement