**Nom** : Djebien
**Prénom** : Tarik
**Groupe** : MIAGE M1-FA
**Sujet** : CAR-TP
**Date** : 04/07/2012

# Documentation.

i. TP carnet d'adresses client/serveur messages

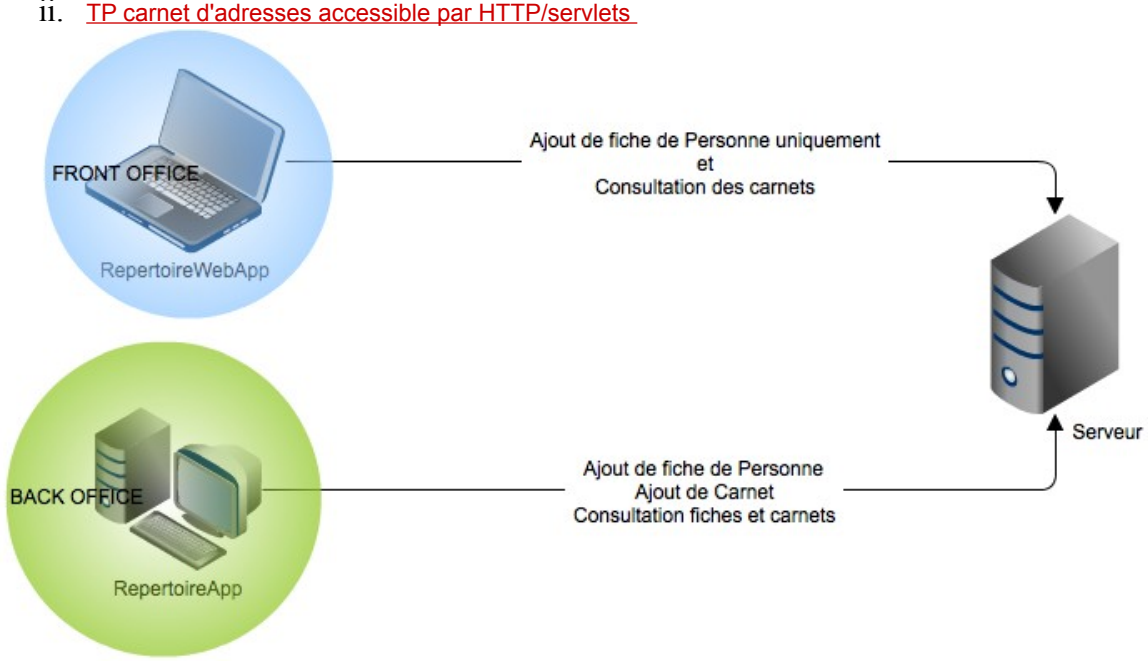
- ***Définition du protocole*** :
  (Voir : /RepertoireApp/src/tp2/client/mvc/model/net/tcp/**ClientProtocole.java**)

| Type du message | Exemples | Remarques |
|---|---|---|
| *MESSAGE_AJOUTER_PERSONNE* | **ajouterPersonne : <idCarnet>;<Nom>;<Email>;<url>;<info>** | Ajoute la personne dans le carnet <idCarnet> s'il existe |
| *MESSAGE_MODIFIER_PERSONNE* | **modifierPersonne : <idCarnet>;<Nom>;<Email>;<url>;<info>** | Modifie la personne dans le carnet <idCarnet> s'il existe et si elle existe |
| *MESSAGE_RETIRER_PERSONNE* | **retirerPersonne : <idCarnet>;<Nom>** | Retire la personne dans le carnet <idCarnet> s'il existe et si elle existe |
| *MESSAGE_CHERCHER_PERSONNE* | **chercherPersonne : <idCarnet>;<Nom>** | Cherche la personne dans le carnet <idCarnet> s'il existe et si elle existe |
| *MESSAGE_LISTER_PERSONNE* | **listerPersonne : <idCarnet>** | Liste toute les personnes existantes dans <idCarnet> |
| *MESSAGE_AJOUTER_CARNET* | **ajouterCarnet** | **Le serveur traite lui même l'identifiant du carnet pour assurer son unicité.** |
| *MESSAGE_RETIRER_CARNET* | **retirerCarnet : <idCarnet>** | Retire le carnet s'il existe |
| *MESSAGE_CHERCHER_CARNET* | **chercherCarnet : <idCarnet>** | Chercher le carnet s'il existe |
| *MESSAGE_LISTER_CARNET* | **listerCarnet** | Lister tous les carnets existants |
| *MESSAGE_ORDER_SUCCESS* | **OK** | **Réponse du serveur en cas de succès, l'ordre de la réponse vaut OK** |
| *MESSAGE_ORDER_FAILED* | **KO** | **Réponse du serveur en cas d'échec, l'ordre de la réponse vaut KO** |
| *MESSAGE_ORDER_ERROR* | **ERROR** | **Réponse du serveur en cas d'erreur, l'ordre de la réponse vaut ERROR** |
| *PARAMS_SEPARATOR* | **;** | **Séparateur entre l'ordre et les valeurs <...> envoyé ou reçu** |
| *ORDER_SEPARATOR* | **:** | **Séparateur entre chacune des valeurs <x>;<y>** |
| *MESSAGE_AUTHENTICATE_USER* | **connectUser : <login>;<password>** | **Pour se connecter à l'application** |
| *MESSAGE_EXIT_APP* | **EXIT** | **Pour quitter l'application** |

- C'est un protocole sans états.
- On assure le déballage et l'emballage des messages avec vérification de la présence ou non des mots clés, séparateurs.

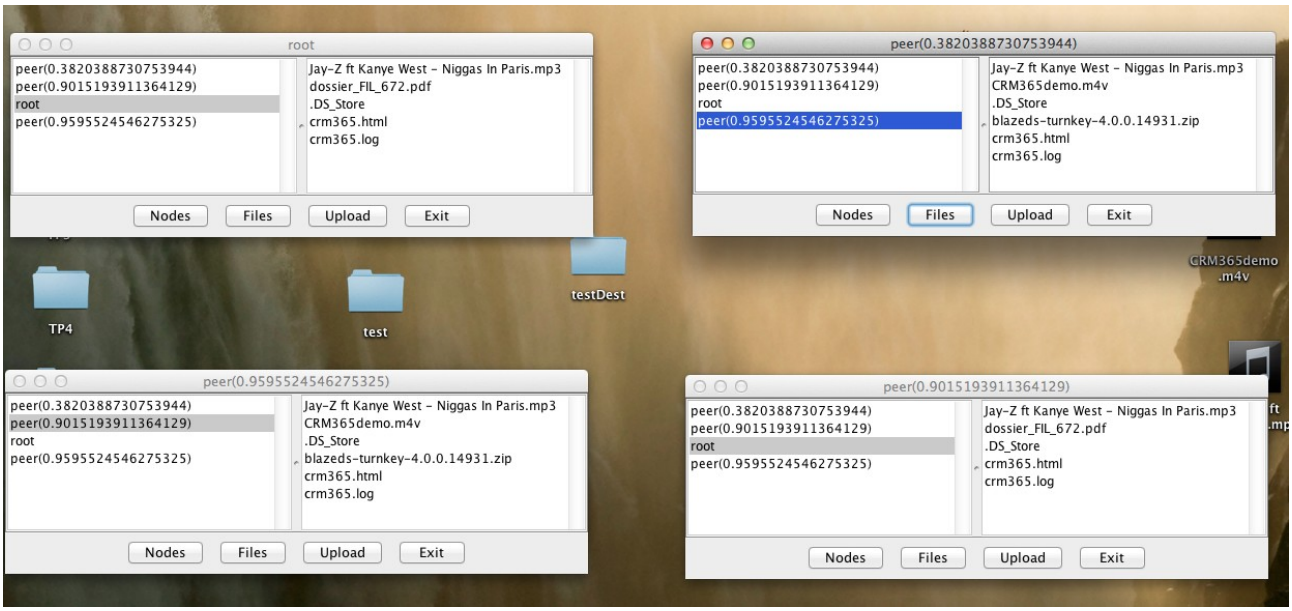
- **Architecture :**
  **- MVC** avec design pattern Observer sur le client SWING.
  - **Multithreadé** avec gestion des transactions sur le serveur JAVA.
  - Le Modèle **M** est en 4 couches :
    - Services
    - Data Access Object
    - Entité
    - Fabrique

ii. TP carnet d'adresses accessible par HTTP/servlets




- *Ici on a une architecture MVC web avec requête/réponse via HTTP vers nos contrôleurs, implémentés par des Servlets, notre client léger se base sur des vues Java Server Page. Le modèle M est exactement le même que le TP précédent, on réutilise donc directement nos services qui utilisent notre proxy pour interroger notre serveur via TCP/IP. Chaque servlet a un rôle particulier pour un traitement spécifique :*
  *- l'authentification*
  *- la gestion des opérations d'ajout, mise a jour et suppression.*
  *- la construction du modèle de donnée à afficher sur la vue. Celle ci n'effectue aucun traitement par respect du MVC, on lui prépare directement les données.*

iii. TP peer2peer / Java RMI




- Limitations :
  - Centralisation sur le nœud racine, si on quitte root, tous les peers s'éteignent.
  - Pour chercher un fichier, il faut parcourir chaque peer un à un, c'est long.
- Améliorations :
  - Chaque peer est une racine potentielle, on forme un réseaux P2P où si le nœud jouant le rôle de racine est shutdown, un autre fonctionnel prends immédiatement le relai, la racine n'est plus statique mais dynamique.
  - Pour la recherche, on peut faire l'analogie au web, comme une « table de routage », où chaque nœud indique vers quel nœud il faut se diriger pour obtenir ce fichier un peu comme la résolution DNS sur un réseau. Un algorithme du plus court chemin augmentera les performances si notre graphe est modéliser par nos peers remote Object, les arcs sont alors les communications via RMI.