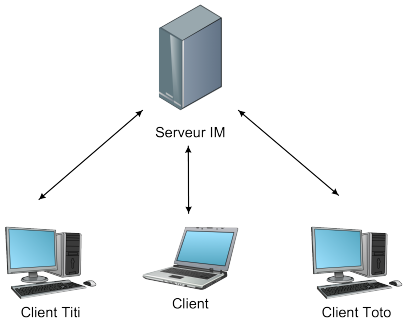
Projet Client-Serveur
Développement d'un Mini-Chat

Dans un premier temps, ce projet a pour but de prendre en main les concepts de la programmation réseau par l'utilisation de l'interface des sockets. Il permettra, aussi, de mettre en place les notions de programmation-objet avancée. Cette prise en main se fera par l'intermédiaire d'un projet : création d'une application de messagerie instantanée ou aussi nommée *Instant Messaging* (abbr. *IM*).

Objectifs :
- créer un serveur de messagerie instantanée,
- créer un client (graphique ou non) répondant au(x) protocole(s) du serveur.

Le serveur mini-chat doit permettre à plusieurs clients de se connecter au serveur et de discuter entre eux à la manière des multiples salon de discussions disponibles sur Internet.

Le principe de fonctionnement est le suivant : un utilisateur écrit un message au serveur puis le serveur s'occupe de renvoyer ce message a chacun des clients connectés. Par exemple (cf. Figure ci-jointe), pour que *Titi* reçoive un message de *Toto*, *Toto* doit d'abord envoyer ce message au serveur.



Une fois connecté, un utilisateur peut se déconnecter à tout moment. Au cours d'une session, un utilisateur peut envoyer certaines commandes particulières comme, notamment, une commande permettant de changer de pseudonyme ou récupérer la liste des utilisateurs connectés.

Deadline et remise :

1. Date de remise (stricte) : 29 mai 2011 (minuit),
2. Créez une archive de nom $login_1$-$login_2$-$login_n$ (3 membres par groupe max.),
3. Déposez cette archive de le dossier `/net/Remise/AS-TeleinfoUML`.

Contenu de l'archive :
- rapport explicatifxs en *PDF* regroupant les réponses aux questions, les choix effectués, les problèmes rencontrés, les eventuelles solutions,
- code source **commenté** et **compilable** du serveur et du client,
- si besoin, divers fichiers (images, scripts, Makefile, ...).

Remarque : Le choix du langage de programmation n'est pas imposé. Il est simplement requis d'utiliser un *langage objet* du type *C++, Java* ou *C#*.

Questions : {foucaud, morsellino}@labri.fr

1 Conception et Modélisation du Mini-Chat

Dans cette partie, vous allez mettre en application les notions vues au cours des TDs de modélisation et conception avancée. L'idée, ici, n'est pas de faire un état de l'art des différents outils appris (diagrammes, patterns) mais plutôt de les utiliser à bon escient et comprendre leur intérêt dans un projet plus complexe.

1. Etablissez les *diagrammes des cas d'utilisation* pour le serveur et un client.

2. Donnez une première ébauche du ou des *diagramme(s) de classe(s)*. Expliquez vos choix (ex. les design patterns utilisés, ...).

3. Donnez les *diagrammes de séquences* de la connexion/déconnexion d'un client puis de l'envoi/réception d'un message sur le client et le serveur.

2 Développement du Mini-Chat

Avant de se lancer dans le développement à proprement parler, il est nécessaire de mettre en place un dialecte qui est à la fois compris par le serveur et utilisé par les clients. Ce dialecte est ce que l'on appelle un protocole de communication. Au cours des TDs, plusieurs protocoles ont été étudiés (ex. POP3 et FTP) dont les aspects techniques sont répertoriés dans un document particulier : RFC (Request For Comments).

1. Définissez votre propre *protocole de communication* pour votre service de messagerie instantanée répondant à un minimum de fonctionnalités :
   - connexion/déconnexion d'un utilisateur,
   - envoi/réception d'un message,
   - récupération de la liste des personnes connectées,
   - changement de pseudonymes.

2. Développez le serveur avec le protocole défini dans la question précédente. Un exemple de serveur simple (Java) avec gestion de processus légers (threads) est disponible à l'adresse : `http://dept-info.labri.fr/~morselli`.

3. Développez le client qui permettra d'interagir avec le serveur et utilisant le protocole mis en place.

Problème : Nous souhaiterions pouvoir utiliser d'autres protocoles de communications. De plus, notre réseau étant fiable, nous aimerions pouvoir utiliser le protocole de transport *UDP*.

1. Si ce n'est pas déjà fait, modifiez le(s) diagramme(s) de classe(s) établi en première partie du projet afin de prendre en compte ce nouvel aspect.

2. Changez votre client et votre serveur pour qu'ils soient modulables aux différentes combinaisons de protocoles. Par exemple, supposons que vous ayez nommé vos protocoles de messagerie *IM1* et *IM2*, votre serveur devra comprendre les combinaisons *IM1/TCP*, *IM2/UDP* et ainsi de suite.

Pour les plus avancés : agrémentez votre projet de nouvelles fonctionnalités (échanges de fichiers, gestion avatars, couleur/polices de caractères, ...).