Ecole d'Ingénieurs du Canton de Vaud
Département d'Electricité et d'Informatique
Filière Informatique Logicielle

MECANISMES DE PRESENCE ET COMMUNICATION PEER-TO-PEER

TRAVAIL DE DIPLOME

Yves Bresson, diplômant
Yves Dennebouy, professeur responsable

Septembre – Décembre 2003

Haute Ecole Spécialisée
de Suisse Occidentale

1. Cahier des charges

Les outils de présence et de messageries sont très répandus actuellement (ICQ, MSN-Messenger, Yahoo Messenger, etc…), mais ils s'appuient sur une architecture client-serveur classique qui n'est ni souple ni tolérante aux pannes. Le but de ce projet est de définir et mettre en place un système de présence et de messagerie en utilisant un protocole de communication peer-to-peer et une architecture sans serveur dédié.

Les objectifs et buts de ce projet sont donc les suivants :

- Définir et mettre en place un réseau peer-to-peer.
- Définir un mécanisme de présence sans serveur s'appuyant sur ce réseau.
- Développer un système de messagerie instantanée peer-to-peer.
- Intégrer ces points au sein d'une même application, en privilégiant la portabilité (compatibilité inter-plateformes).

2. Table des matières

3. Résumé

L'architecture la plus répandue actuellement est l'architecture client-serveur, qui a comme principaux désavantages un coût de mise en place élevé et une tolérance aux pannes très faible. L'architecture peer-to-peer, où les participants coopèrent et interagissent dans un but commun, permet de remédier à ces problèmes. Les applications de messagerie instantanée existantes (telles que ICQ ou Yahoo! Messenger) sont pour la plupart basées sur une architecture client-serveur. Le but de ce projet est donc de définir et mettre en place un réseau peer-to-peer qui servira de support à la réalisation d'une application de messagerie instantanée sans serveur dédié.

Le résultat de ce travail, entièrement implémenté en Java, est décomposé en deux modules principaux. Tout d'abord PDP[1] : un module de création / gestion d'un réseau peer-to-peer. Ce module permet la découverte et la recherche de peers, ainsi que l'échange de messages entre peers. La communication est possible même entre peers "cachés" derrière des firewalls, grâce à l'utilisation d'un système de relais. Le deuxième module, PIM[2], est une application graphique réalisée au-dessus du module PDP. Similaire dans son aspect aux systèmes de messagerie instantanée existants, PIM permet de gérer, visualiser, et "utiliser" les peers.

Les objectifs initiaux ont globalement été atteints, et parfois même dépassés, comme c'est le cas pour le module PIM. En effet, PIM est basé sur un système de plugins. De ce fait, son utilisation n'est pas limitée à un service de messagerie instantanée. Il est possible d'ajouter toutes sortes d'applications se prêtant à un fonctionnement peer-to-peer (échange de fichiers, chat, tableau noir partagé, etc). De plus, l'API de PIM est publique, ce qui signifie que n'importe qui peut développer de nouveaux plugins.

[1] Peer Discovery Protocol
[2] Peer-to-peer Instant Messaging

4. Introduction

Cette partie débute par une présentation des concepts et domaines abordés par le présent travail de diplôme. Il y sera question notamment de peer-to-peer, de "présence" sur Internet et de ce qu'est un système de messagerie instantanée. Enfin, cette introduction se termine par un survol de quelques-unes des principales applications existantes dans les domaines de l'échange de fichiers peer-to-peer et de la messagerie instantanée.

4.1. Peer-to-peer (P2P)

Ce point présente le concept du peer-to-peer.

4.1.1. Définition

Un environnement peer-to-peer est une architecture réseau dans laquelle chaque ordinateur (ou "nœud", ou encore "peer") possède des capacités et des responsabilités équivalentes, et où chaque peer participe aussi bien en tant que client qu'en tant que serveur.

Tous les nœuds participent en mettant à disposition au moins une partie de leurs ressources, ces ressources pouvant être aussi bien physiques (par exemple de l'espace disque, de la bande passante, de la puissance de calcul) que logiques (par exemple des services ou des "connaissances").

On peut définir un système peer-to-peer par quelques-unes de ses propriétés :

- Pas de coordination centralisée
- Aucun peer n'a une vue globale du système
- Toute l'information existante est disponible depuis n'importe quel peer
- Le comportement global découle des interactions locales
- Les peers sont autonomes

On peut ajouter à cette liste la propriété suivante, importante à prendre en compte lors du développement d'applications peer-to-peer :

- Les peers sont non fiables

Enfin, une propriété souvent présente (et intéressante pour les participants) dans les applications peer-to-peer, mais qui n'est pas une propriété du concept peer-to-peer en soi, est la

- Possibilité pour les participants de garder l'anonymat

Un environnement peer-to-peer peut donc être représenté comme sur la figure ci-dessous. Cette figure montre bien qu'il n'y a pas de coordination centrale, et que les peers n'ont pas de vue globale du système.

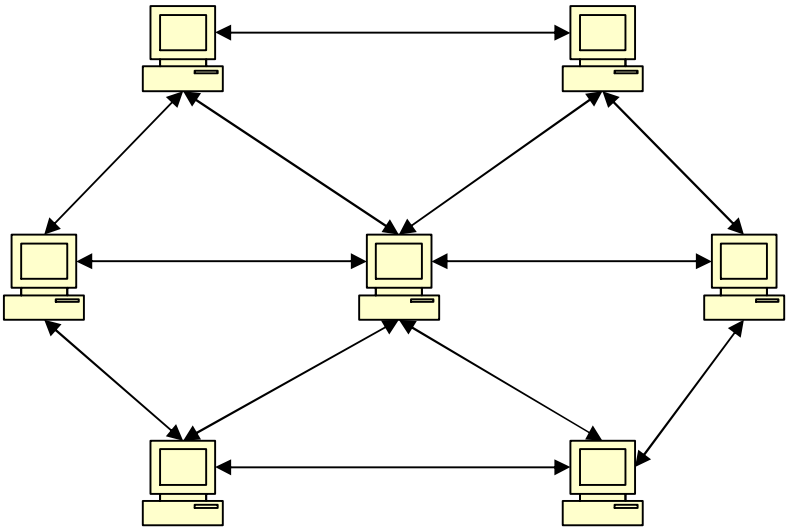


Figure 1. Réseau peer-to-peer

4.1.2. Peer-to-peer VS Client-Serveur

L'architecture la plus connue et la plus répandue actuellement (de par son utilisation par le protocole http pour l'accès aux pages web) est l'architecture client-serveur.

Pour mémoire, l'architecture client-serveur se présente comme indiqué sur le schéma ci-dessous.

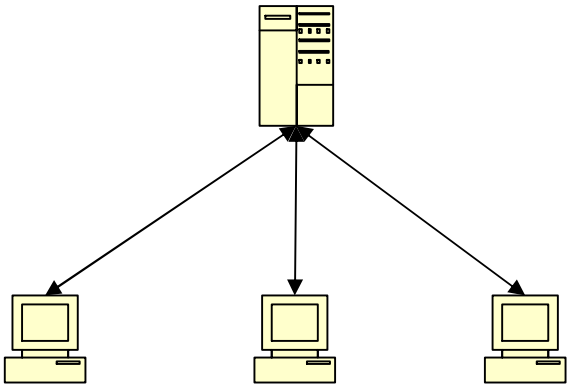


Figure 2. Architecture client-serveur

Cette architecture est de type 1-n : un serveur face à un ensemble de n clients lui envoyant leurs demandes.

Contrairement au peer-to-peer, les réseaux client-serveur dépendent d'un serveur central qui sert d'entrepôt de données et se charge de transmettre ces données aux ordinateurs clients. De plus, les ordinateurs clients ne communiquent pas les uns avec les autres, mais tout le trafic passe par le serveur central.

Les avantages et les inconvénients du peer-to-peer face à la technologie client-serveur sont résumés dans le tableau ci-dessous :

Tableau 1. Comparaison client-serveur et peer-to-peer

	Avantages	Inconvénients
Client-Serveur	- très fiable (l'information est correcte) - facile à organiser (le serveur est seul à décider)	- dépendant du serveur - cher à mettre en place - puissance de calcul limitée
Peer-to-peer	- indépendant du serveur - bon marché (pas de serveur) - grande puissance de calcul	- fiabilité pas à 100% - difficile à organiser

Comme on le constate, les deux technologies ont leurs avantages et leurs inconvénients et l'architecture client-serveur ne va donc pas disparaître. Les deux technologies sont en effet complémentaires et l'une ne va pas remplacer l'autre. Par contre, le peer-to-peer permet d'envisager de nouvelles perspectives.

4.1.3. Types d'applications peer-to-peer

Les applications peer-to-peer peuvent être classées dans plusieurs catégories, les principales étant les suivantes :

- les systèmes de partage de fichiers
- les systèmes de communication
- les systèmes distribués
- les systèmes collaboratifs

On peut ajouter à ces catégories une catégorie un peu spéciale : les systèmes de peer-to-peer "humain", tel que les sites de vente aux enchères (par exemple eBay). Cette catégorie est cependant peut intéressante, car si l'interaction entre participants relève bien du concept peer-to-peer l'application et le système d'information sous-jacents ne sont pas du tout peer-to-peer.

4.1.3.1. Systèmes de partage de fichiers

Cette catégorie est sans doute la plus connue et la plus contestée. Les systèmes de partage de fichiers permettent aux participants de mettre à disposition et de trouver toutes sortes de fichiers (généralement des fichiers musicaux mp3, des films, des applications, des jeux etc. d'où la polémique autour de ces logiciels).

Par exemple : Kazaa, eDonkey

4.1.3.2. Systèmes de communication

Cette catégorie regroupe toutes les applications permettant à deux participants ou plus d'entrer en contact, par exemple par l'utilisation de la téléphonie sur Internet, de la vidéo-conférence ou encore de la messagerie instantanée qui est présentée plus loin dans ce document.

Par exemple : ICQ, Yahoo!Messenger

4.1.3.3. Systèmes distribués (grid computing)

Les applications de cette catégorie permettent la mise en commun des ressources (généralement les ressources non exploitées) des ordinateurs participants, telle que la puissance de calcul ou la mémoire, en vue de faire tourner de "grosses" applications nécessitant d'importantes ressources.

Par exemple : SETI@Home (découverte d'intelligence extraterrestre), Genome@Home (décoder le génome humain)

4.1.3.4. Systèmes collaboratifs

Cette catégorie concerne le partage d'application, c'est-à-dire l'utilisation collaborative par plusieurs participants d'une même application.

4.1.4. Les modèles peer-to-peer

Jusqu'ici le modèle peer-to-peer a été considéré comme un seul modèle, mais en réalité ce modèle peut se présenter sous différentes formes, qui sont brièvement présentées ci-dessous :

- le modèle centralisé
- le modèle décentralisé
- le modèle hiérarchique
- le modèle structuré

4.1.4.1. Le modèle centralisé

Le modèle centralisé est un peu similaire au modèle client-serveur, en effet ses caractéristiques sont les suivantes :

- index global maintenu par une autorité centrale
- communication directe entre le peer "client" et le peer en possession de la réponse

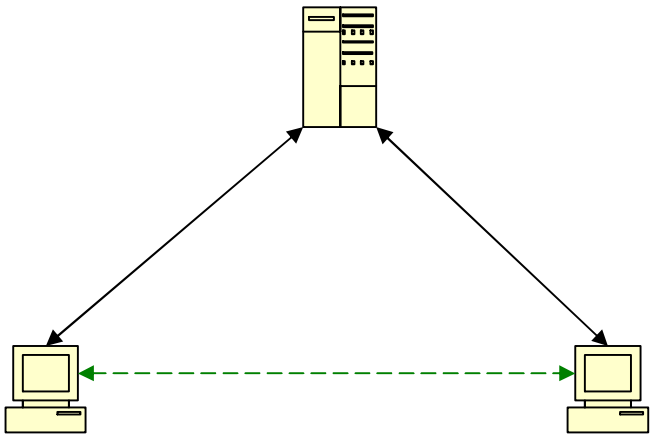


Figure 3. Réseau centralisé (en noir : recherche, en vert : échange de données)

Par exemple : Napster, précurseur des logiciels d'échange de fichiers (voir plus loin)

4.1.4.2. Le modèle décentralisé (sans structure)

Ce modèle est en quelque sorte le "vrai" modèle peer-to-peer, sans autorité centrale.

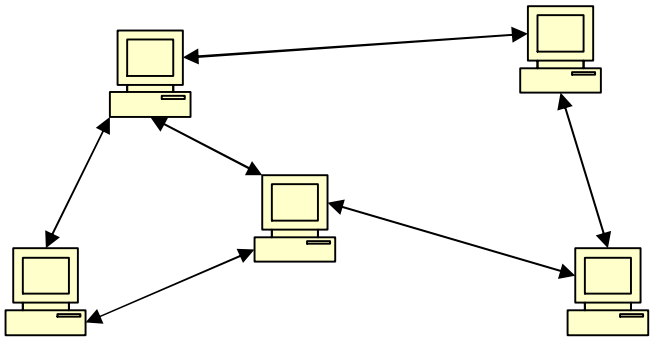


Figure 4. Réseau décentralisé (recherche et données empruntent les mêmes voies)

Par exemple : Gnutella, partage de fichiers (voir plus loin)

4.1.4.3. Le modèle hiérarchique

Ce modèle est un mélange des deux modèles précédents. Il introduit la notion de "super-peers", qui sont des peers prenant une place privilégiée pour former une sorte de réseau de super-peers à l'intérieur du réseau de départ des peers. Ces super-peers ayant à charge des tâches similaires à celle d'un serveur (indexation, réponse au clients, …).

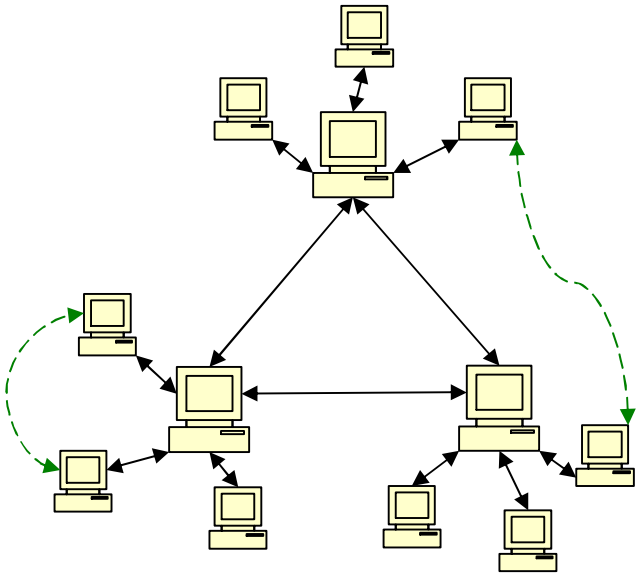


Figure 5. Réseau hiérarchique (en noir : recherche, en vert : échange de données)

Par exemple : Kazaa, partage de fichiers (voir plus loin)

4.1.4.4. Le modèle structuré

Ce modèle est un modèle décentralisé auquel s'ajoute une structure, mais une structure qui n'est ni fixe ni déterminée à priori. Le modèle structuré est encore un sujet de recherche, dont fait partie le projet P-Grid (réseau peer-to-peer décentralisé et auto organisationnel) développé

à l'EPFL (Ecole Polytechnique Fédérale de Lausanne). Dans ce modèle, le réseau peut être vu comme une sorte d'arbre, où chaque peer est responsable de connaître l'emplacement d'un sous-ensemble de peers de même qu'un (des) peer(s) responsable(s) du (des) sous-ensemble(s) complémentaire(s).

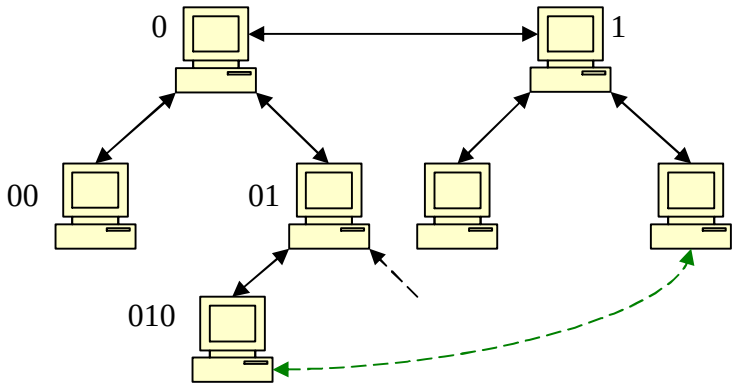


Figure 6. Réseau structuré (en noir : recherche / structure, en vert : échange de données)

4.2. "Présence" sur Internet

Le concept de "présence" sur Internet est très simple : il s'agit de savoir si un ordinateur (un participant, un peer) est présent en ce moment, et si c'est bien le cas de connaître sa position (typiquement son adresse IP de manière à pouvoir le joindre).

La résolution du problème de "présence" nécessite la mise en place de plusieurs éléments :

- un protocole d'annonce de présence
- un protocole de requête de la présence d'un participant

Une manière simple de résoudre ce problème consiste à utiliser un serveur de présence qui traite toutes les demandes et annonces de présence.

4.3. Messagerie instantanée (IM)

Les systèmes de messagerie instantanée (*Instant Messaging* ou IM) sont des systèmes permettant l'échange de messages entre participants, et ceci de manière instantanée (d'où le nom). Ces systèmes peuvent être vus comme un équivalent du système SMS (*Short Message Service*) disponible sur les réseaux de téléphonie mobile, à la différence que dans un système de messagerie instantanée les utilisateurs doivent pouvoir connaître l'état actuel de leurs correspondants (savoir lesquels parmis leurs "amis" sont connectés et lesquels ne le sont pas).

Les propriétés d'une application de messagerie instantanée qu'il faut considérer lors de la construction d'une telle application sont notamment les suivantes :

- gestion de l'authentification des participants
- résolution du problème de présence
    - annonce de présence

    - demande de présence
- transmission des messages

A nouveau, la solution la plus simple consiste à mettre en place un serveur central responsable de l'ensemble du travail.

4.4. Applications existantes : messagerie instantanée

Cette partie est une petite étude comparative de quelques applications existantes.

Ces dernières années, le nombre d'applications de messagerie instantanée n'a cessé d'augmenter. De nouveaux "concurrents" sont apparus les uns après les autres tandis qu'en parallèle plusieurs logiciels "passerelle" (sensés permettre l'utilisation simultanée et transparente des différents systèmes) étaient développés (par exemple Trillian). Cette section ne traite pas de ces logiciels "passerelle" ou multi-systèmes, étant donné qu'ils n'apportent rien de nouveau du point de vue des techniques / protocoles de communication utilisés. En effet, ils se contentent d'appliquer les protocoles définis par les applications qu'ils "copient". A relever toutefois que ce sont souvent les importants travaux de reverse engineering mis en œuvre pour le développement de tels logiciels "passerelle" qui ont permis de découvrir les techniques employées par les applications de messagerie instantanée les plus répandues, le code source de ces dernières étant rarement du domaine publique (open-source).

La grande majorité des applications de messagerie instantanée étant très similaire, cette partie présentera tout d'abord un peu plus en détail la première d'entre elles (ICQ), avant de mentionner plus rapidement les autres en précisant uniquement les éventuelles différences d'importance.

4.4.1. ICQ

ICQ fut le premier, ou en tout cas l'un des tout premiers systèmes de messagerie instantanée.

4.4.1.1. Historique

On peut dire que le concept de messagerie instantanée a été crée par les pères de ICQ, quatre jeunes israéliens (Yair Goldfinger, Arik Vardi, Sefi Vigiser et Amnon Amir, âgés respectivement de 26, 27, 25 et 24 ans) qui ont fondé la société Mirabilis pour publier la première version de ICQ en novembre 1996 (le nom "ICQ" étant l'abréviation de "I seek you", qui signifie "Je te cherche").

En quelques mois, ICQ a gagné une énorme popularité et, selon les chiffres de Mirabilis, après seulement 6 mois d'exploitation leur système comptait déjà environ 850'000 utilisateurs enregistrés. Aujourd'hui, on compte plus de 250'000'000 de comptes enregistrés dans le système (ce qui est hautement fantaisiste, il faudrait en réalité tenir compte du grand nombre de comptes "perdus" qui ne sont plus utilisés).

Toujours est-il qu'avec un succès pareil la concurrence ne s'est pas faite attendre bien longtemps. En juin 1998, Mirabilis a été racheté par America Online qui avait pourtant déjà crée son propre système comme nous le verrons plus loin.

4.4.1.2. Fonctionnement

D'un point de vue utilisateur, et cet aspect est grosso modo le même quel que soit le logiciel utilisé, l'utilisation de ICQ se décompose en 7 étapes :

1. Téléchargement et installation du client ICQ.
2. Obtention d'un compte utilisateur auprès du serveur ICQ central (lors de la première utilisation).
3. Connexion au serveur ICQ et vérification des informations de login (nom d'utilisateur et mot de passe).
4. Le client transmet ses informations de connexion (IP et port local utilisé par ICQ) au serveur, de même que la liste des contacts dont il aimerait connaître le statut.
5. Le serveur vérifie la présence des contacts demandés et transmet le résultat de cette recherche au client (en incluant les IPs et ports associés), le serveur prévient également les éventuels clients intéressés de votre arrivée en ligne.
6. Le client peut maintenant communiquer directement avec ses contacts puisqu'il connaît les IPs et ports à utiliser.
7. Lors de la fermeture de ICQ, le client prévient le serveur qui met à jour sa liste de présences et transmet l'information de votre départ aux clients concernés.

Figure 7. ICQ

Dans ses premières versions (jusque à la version ICQ99), ICQ était basé à la fois sur UDP et TCP. La connexion au serveur et les messages tels que les demandes de présence étaient envoyés par UDP. Par contre, les connexions entre utilisateurs étaient directes et utilisaient TCP. Il était toutefois possible de passer par le serveur pour l'envoi de messages en cas de problème avec la connexion directe.

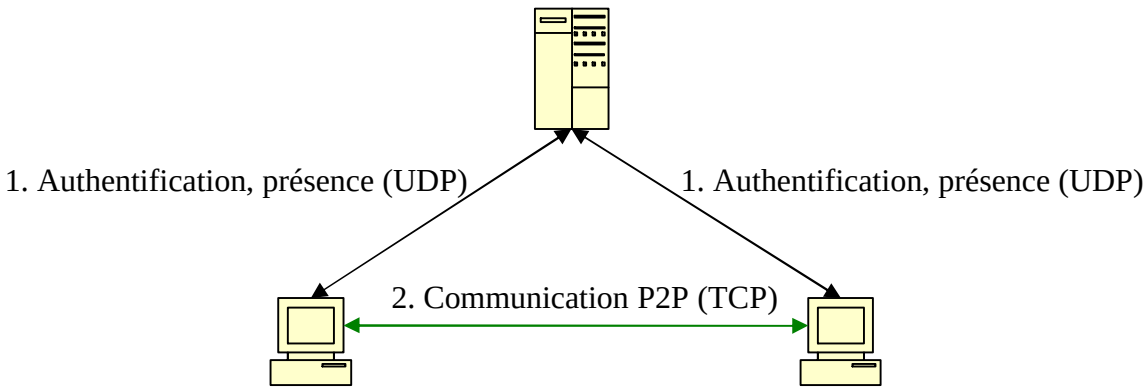


Figure 8. Schéma de communication ICQ 99 (centralisé)

Ceci a changé depuis la version ICQ2000, qui semble maintenant utiliser le protocole OSCAR de AOL (voir plus loin).

4.4.2. AOL Instant Messenger

Peu après le lancement de ICQ, America Online a lancé son propre système de messagerie instantanée, appelé AIM (AOL Instant Messenger), avant de racheter Mirabilis et ICQ. ICQ et AIM utilisent donc aujourd'hui un protocole commun, qui permet une certaine compatibilité entre les deux logiciels (au départ, et ceci jusque à l'été 2003 encore, AOL bloquait la communication entre ICQ et AIM).

Le protocole utilisé est le protocole OSCAR, qui est basé uniquement sur TCP. Ce protocole décompose la connexion au système en deux phases :

- Connexion au serveur d'autorisation (login.oscar.aol.com) qui vérifie le login et le mot de passe de l'utilisateur, puis lui transmet un "cookie", sorte de code d'accès au reste du système.
- Connexion au serveur BOS (Basic Oscar Service) indiqué par le serveur d'autorisation (le client doit fournir un "cookie" valide) qui se charge de gérer les problèmes de présence.

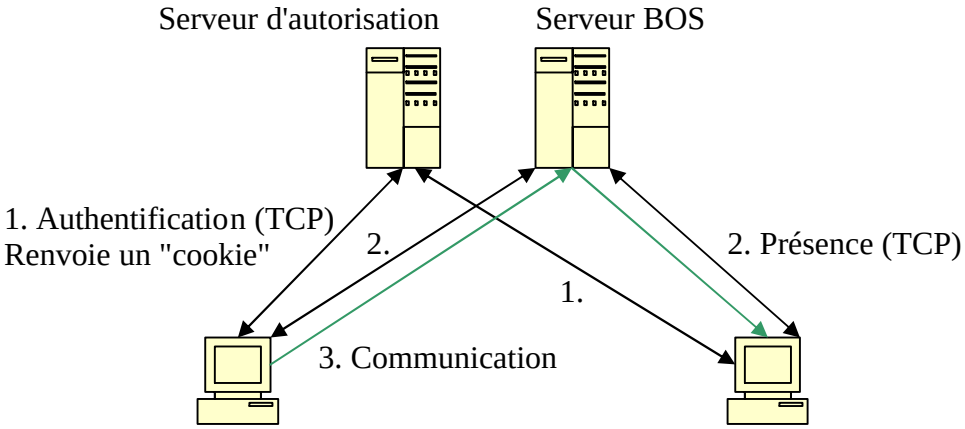


Figure 9. Schéma de communication OSCAR (AIM, ICQ 2000) (client-serveur)

A partir de là, le système est prêt à fonctionner. Tous les messages entre utilisateurs passent par le serveur BOS qui les relayent vers leur destination (aucun aspect peer-to-peer).

4.4.3. Yahoo! Messenger

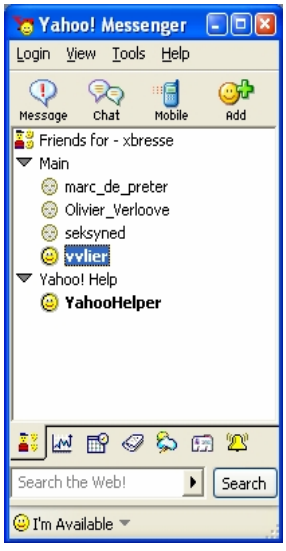


Figure 10. Yahoo!

Yahoo Messenger utilise lui aussi un protocole propriétaire, protocole qui est très semblable au protocole OSCAR utilisé par AIM, à la différence qu'il semble n'utiliser qu'un seul et même serveur aussi bien pour l'authentification que pour la transmission des messages. Le protocole Yahoo! Messenger est donc un protocole TCP (il semble aussi utiliser http dans certains cas) dont le principe de fonctionnement est le suivant :

- phase "authentification"
  - le client transmet son nom d'utilisateur au serveur.
  - le serveur répond avec une chaîne de hachage ("challenge string").
  - le client hache ("hash") son nom d'utilisateur et son

mot de passe à l'aide de cette chaîne et transmet le résultat au serveur.
  - en cas de succès, l'application passe en phase "messagerie".

- phase "messagerie"
  - le client est prêt et il peut envoyer / recevoir des messages (qui transitent par le serveur).

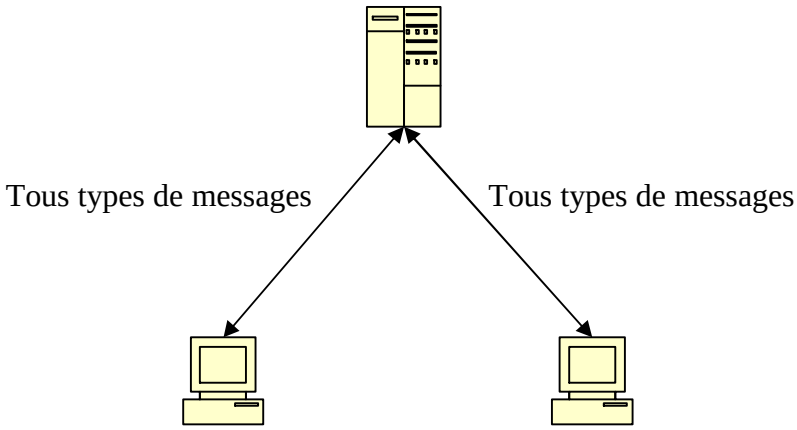


Figure 11. Schéma de communication Yahoo! Messenger (client-serveur)

Ce protocole n'est donc pas du tout peer-to-peer.

4.4.4. Microsoft Messenger

Le phénomène de la messagerie instantanée prenant de l'ampleur, Microsoft se devait de proposer son propre logiciel, et ce fut chose faite avec Microsoft Messenger, qui est maintenant massivement distribué et pré-installé avec les nouvelles versions de Windows.

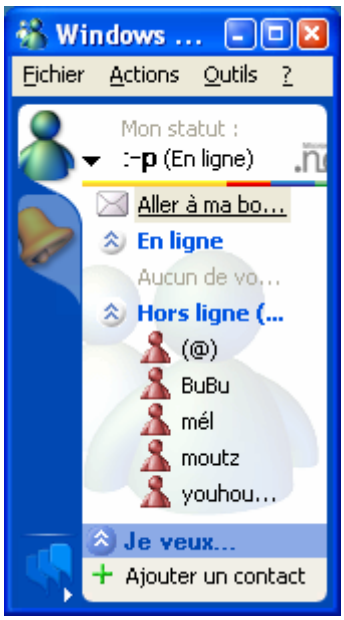


Figure 12. Messenger

Microsoft Messenger fonctionne selon le même principe que AIM (OSCAR) en utilisant deux serveurs distincts (la seule différence provient de la répartition des tâches sur les deux serveurs, qui n'est pas exactement la même) :

- un serveur de connexion (messenger.hotmail.com:1863, ce serveur peut éventuellement rediriger le client vers un serveur de connexion annexe) qui sert à s'authentifier et à récupérer les informations de présence (cette connexion doit donc être maintenue).

- un serveur de messagerie (SwitchBoard Server) qui relaye les messages entre utilisateurs.

Il est inutile de s'étendre d'avantage sur le sujet si ce n'est pour faire remarquer que Microsoft Messenger ne fait pas non plus usage des avantages du peer-to-peer.

4.4.5. Jabber

Jabber n'est pas vraiment une application, mais plutôt un protocole (XML sur TCP) pour l'échange temps réel de messages et d'information de présence sur Internet. L'avantage de Jabber est qu'il est open-source et que n'importe qui peut développer un client utilisant ce protocole.

Le protocole Jabber suit une approche assez différente de celle adoptée par les applications vues jusqu'ici, en effet Jabber utilise un modèle similaire à celui utilisé pour la messagerie électronique (email). Il y a un nombre quelconque de serveurs, pas forcément interconnectés, qui s'occupent chacun d'une partie des utilisateurs.

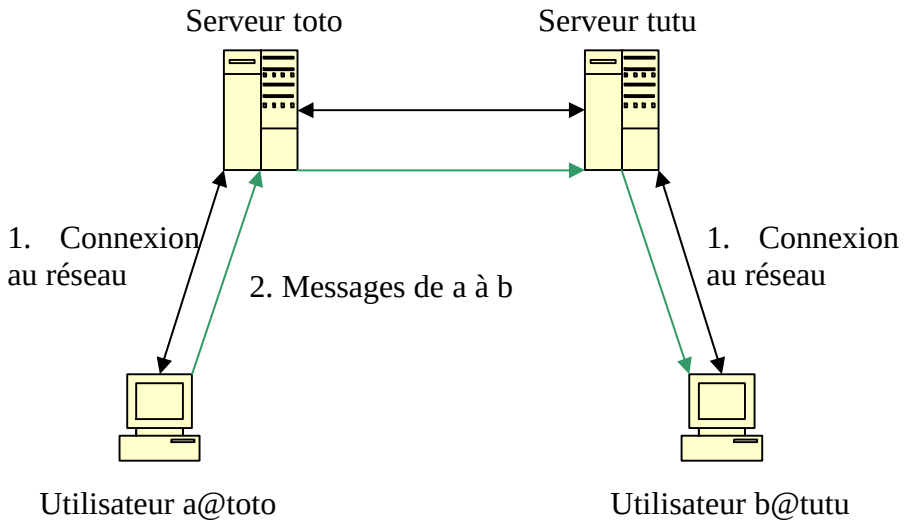


Figure 13. Schéma de communication Jabber

Chaque utilisateur est identifié par un nom auquel, comme pour l'email, est ajouté le nom du serveur auquel cet utilisateur se connecte. L'acheminement des messages entre utilisateurs se fait ensuite par les serveurs en se basant sur le nom de serveur, puis lorsque le bon serveur est atteint celui-ci remet le message à l'utilisateur concerné.

On peut donc malheureusement conclure que Jabber n'est aucunement un système peer-to-peer.

4.4.6. Systèmes "privés"

En dehors des applications bien connues de messagerie instantanée présentées jusqu'ici, il existe toute une panoplie d'applications à vocation "privée", c'est-à-dire ayant pour but d'être utilisées dans un environnement restreint, tel que le réseau local d'une entreprise.

4.4.7. Comparatif

Le tableau ci-dessous présente un récapitulatif des principales caractéristiques des différents systèmes de messagerie instantanée étudiés :

Tableau 2. Comparaison des systèmes de messagerie instantanée

		Identification	*Présence*	*Messages*	*UDP / TCP*
ICQ 99		centralisé	centralisé	P2P	Les deux
ICQ 2000		centralisé	centralisé	centralisé	TCP
AIM		centralisé	centralisé	centralisé	TCP
Yahoo! Messenger		centralisé	centralisé	centralisé	TCP (HTTP)
Microsoft Messenger		centralisé	centralisé	centralisé	?
Jabber		N serveurs	N serveurs	N serveurs	TCP

4.4.8. Conclusion

L'élément primordial à relever est l'absence d'utilisation du concept peer-to-peer dans les applications de messagerie instantanée existantes.

Cela peut s'expliquer par la plus grande facilité à implémenter une telle application en se basant sur une architecture client-serveur classique, mais un tel choix rend le système extrêmement vulnérable (sauf peut-être dans le cas de Jabber, où la perte d'un serveur n'entraînerait la perte que des utilisateurs rattachés à ce serveur particulier).

4.5. Applications existantes : échange de fichiers

Les logiciels d'échange de fichiers sont vraiment les logiciels qui ont fait parler du peer-to-peer et qui l'ont popularisé. Cette section présente donc un petit survol des principales applications apparues ces dernières années dans ce domaine très controversé.

4.5.1. Napster

Napster est l'original, par qui la tempête est arrivée. Le but de Napster lors de son apparition en 1999 était de permettre à ses utilisateurs d'échanger des fichiers en toute simplicité. Pour se faire, Napster était basé sur une architecture très simple :

- un serveur central, responsable de :
  - gérer les connexions
  - indexer les fichiers mis à disposition par les clients
  - répondre aux demandes de recherche de fichiers des clients
- n clients connectés sur le serveur

Cette architecture est illustrée dans la figure suivante, qui représente le serveur Napster et deux clients désireux de s'échanger des fichiers.

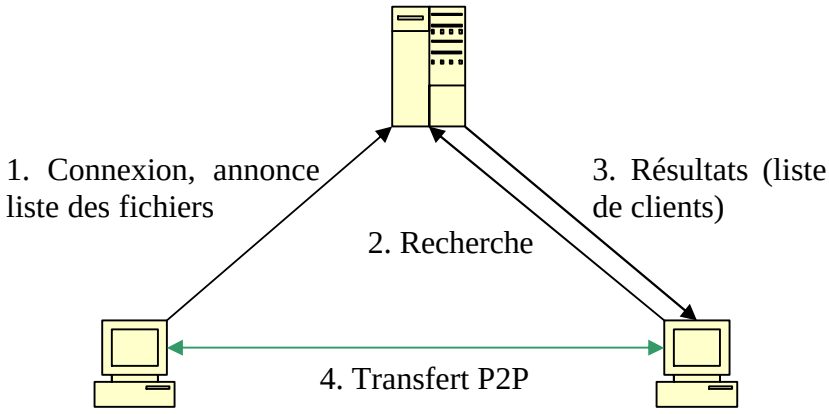


Figure 14. Schéma de communication Napster (centralisé)

Lors de la connexion d'un client, ce dernier annonçait au serveur la liste des fichiers qu'il mettait à disposition. Ensuite, lors d'une recherche demandée par un autre client, le serveur répondait à ce client en lui transmettant une liste de clients ayant en leur possession le fichier demandé. La connexion entre deux clients se faisait alors en peer-to-peer.

Cette architecture avait l'avantage d'être très simple, mais elle était par contre très peu résistante (un serveur central étant le point critique du système), aussi bien aux pannes qu'à d'autres problèmes d'ordre judiciaire par exemple (problèmes judiciaires qui ont amené à la fermeture du Napster original).

4.5.2. Gnutella

Gnutella, apparu fin 1999, introduit de nombreux changements dans l'architecture d'échange de fichiers. En effet, avec Gnutella, il n'y a plus de serveur : tous les clients sont au même niveau et chaque client (ou peer) est typiquement connecté à quatre autres peers auxquels il transmet tous les messages qu'il reçoit, pour autant que la durée de vie (TTL, Time To Live) du message ne soit pas écoulée et que le message ne décrive pas une boucle (les messages portent un identifiant dans le but de détecter les boucles).

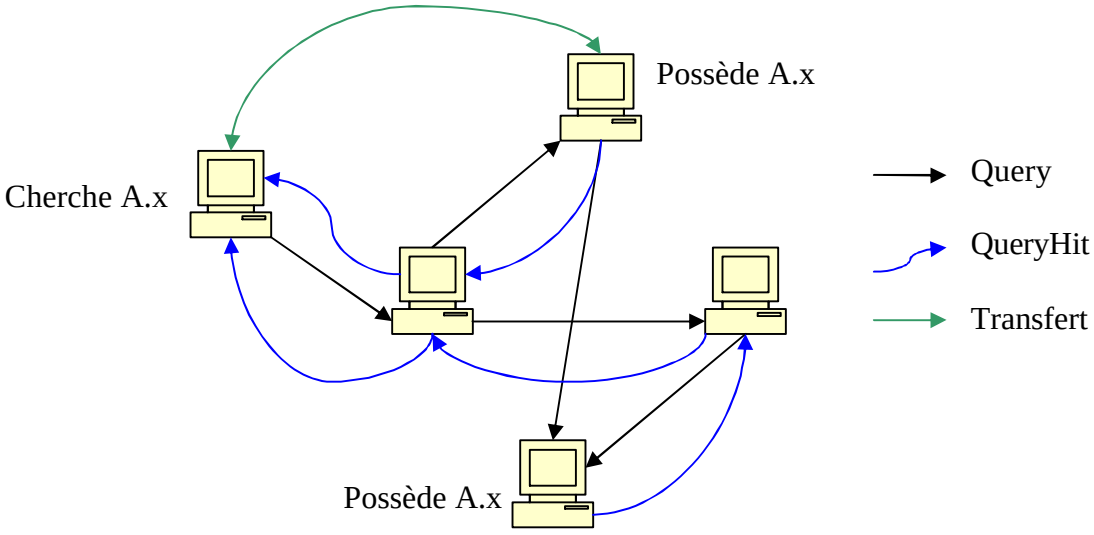


Figure 15. Schéma de recherche avec Gnutella (décentralisé)

Lorsqu'un client se connecte au réseau (au travers d'un autre hôte qu'il a dû trouver par un moyen qui ne fait pas partie du protocole Gnutella), il annonce sa présence à ce peer par un message PING. Ce peer répond par un message PONG et transmet le PING reçu à ses propres voisins. De cette manière, après quelques itérations, le nouvel arrivant connaît suffisamment de voisins.

La recherche se fait selon le même principe, à l'aide des messages 'Query' (demande de fichier) et 'QueryHit' (réponse "j'ai ce fichier"). Une fois que le peer demandeur a reçu une liste de réponses, il choisi un ou plusieurs peers auxquels il se connecte directement pour récupérer le fichier voulu.

Le danger de cette architecture est qu'elle implique un nombre potentiellement très élevé de messages transmis, mais son avantage est qu'elle est à la fois simple et robuste, de par l'utilisation d'un véritable réseau virtuel décentralisé peer-to-peer (Gnutella est utilisé par des applications telles que Morpheus ou Limewire).

4.5.3. Kazaa

Le protocole utilisé par le logiciel Kazaa s'appelle FastTrack. FastTrack est un protocole propriétaire sur lequel aucune information "officielle" n'est disponible. Il semble être une évolution de Gnutella, ou plus exactement une sorte de mélange entre Napster et Gnutella. En effet, Kazaa se base sur un réseau décentralisé de nœuds (peers), parmis lesquels certains deviennent dynamiquement ce que l'on peut appeler des "super nœuds" faisant office de serveurs (d'indexeurs de fichiers) pour un ensemble de peers. Le résultat de tout ceci est que le réseau Kazaa est mieux à même de résister à une augmentation du nombre de peers (meilleure dimensionnabilité) étant donné que les messages entre peers sont moins nombreux (filtrés par les super nœuds, qui eux communiquent selon un protocole similaire à Gnutella).

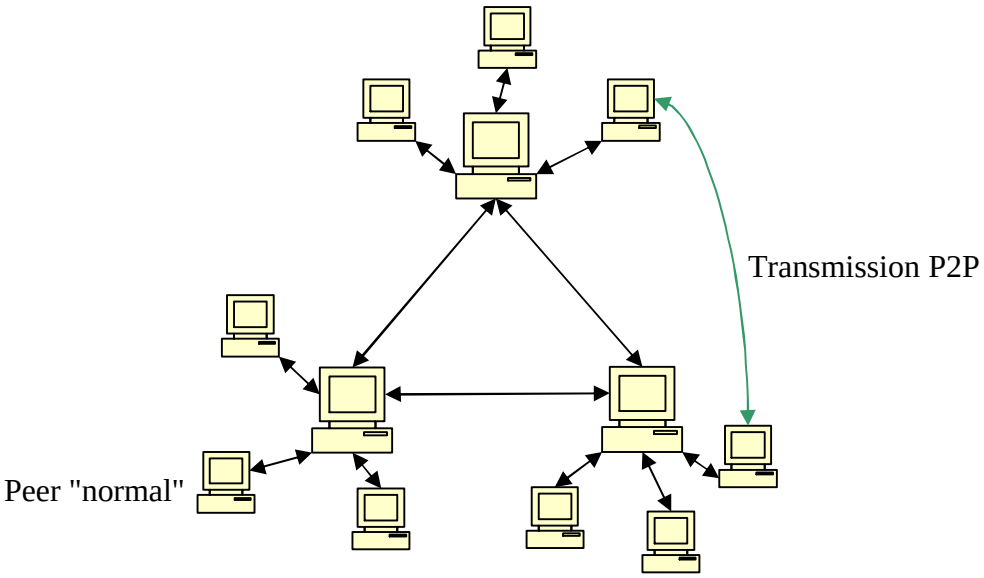


Figure 16. Architecture du réseau Kazaa (hiérarchique)

Kazaa utilise en fait une architecture correspondant au modèle peer-to-peer hiérarchique présenté dans la première partie de ce rapport, où les clients se connectent à un serveur parmis un ensemble de serveurs, lesquels communiquent entre eux pour répondre aux demandes des

clients (il semble qu'il existe encore un unique "super super-peer" dont le rôle n'est pas clairement établi).

4.5.4. eDonkey

Pour terminer, voyons le cas de eDonkey (ou encore eMule, qui est un client utilisant le même réseau). Le logiciel eDonkey se base sur une structure similaire à celle de Kazaa (réseau hiérarchique) mais dans laquelle deux éléments ont été modifiés :

- les serveurs (super-peers) sont fixes et constituent une application distincte du client eDonkey.
- une recherche lancée par un peer connecté à l'un des serveurs n'est pas transmise aux autres serveurs sauf demande explicite du client, et même dans ce cas, le serveur distant contacté peut limiter ses réponses.

Le fait que les serveurs soient fixes, et non pas dynamiquement "élevés" au rang de serveur, constitue une faiblesse potentielle du système. Pour limiter cette faiblesse, lors de la connexion d'un peer à un serveur ce dernier transmet au client tous les serveurs qu'il connaît (ceci afin que s'il venait à disparaître, le client puisse se reconnecter au travers d'un autre serveur du réseau).

4.5.5. Comparatif

Le tableau ci-dessous présente un récapitulatif des principales caractéristiques des différents systèmes d'échange de fichiers étudiés :

Tableau 3. Comparaison des systèmes d'échange de fichiers

		Echanges P2P entre utilisateurs	Gestion de l'information P2P
Napster		✓	✗
Gnutella		✓	✓
Kazaa		✓	✗ Semi-peer-to-peer (hiérarchique)
eDonkey		✓	✗ Semi-peer-to-peer (hiérarchique)

4.5.6. Conclusion

Contrairement aux logiciels de messagerie instantanée, les applications d'échange de fichiers utilisent réellement le concept peer-to-peer, bien que certains aspects (utilisation de serveurs fixes ou "bien connus" pour l'accès au réseau de peers) de ces applications relèvent encore du modèle client-serveur.

5. Développement

5.1. Environnement

Ce point présente l'environnement de réalisation du projet, les outils et langages utilisés.

5.1.1. Langage

Le développement de ce projet a été réalisé entièrement en JAVA, d'une part par intérêt personnel, et d'autre part pour atteindre l'objectif de portabilité souhaité.

5.1.2. Outils

Dans un premier temps, j'avais envisagé d'utiliser Borland JBuilder 9, mais après quelques heures d'essai, et après être tombé par hasard sur NetBeans IDE (une alternative des plus valables), j'ai abandonné cette idée et j'ai utilisé NetBeans IDE 3.5.1 tout au long de ce projet.

Mon choix d'utiliser NetBeans plutôt que JBuilder a été dicté par une impression immédiate de plus grande facilité d'utilisation avec NetBeans et de plus grande intuitivité de l'environnement. En ce qui concerne les fonctionnalités, NetBeans contient tout ce dont j'ai eu besoin : débuggeur, synchronisation des interfaces, générateur de javadoc, design visuel d'interface, etc.

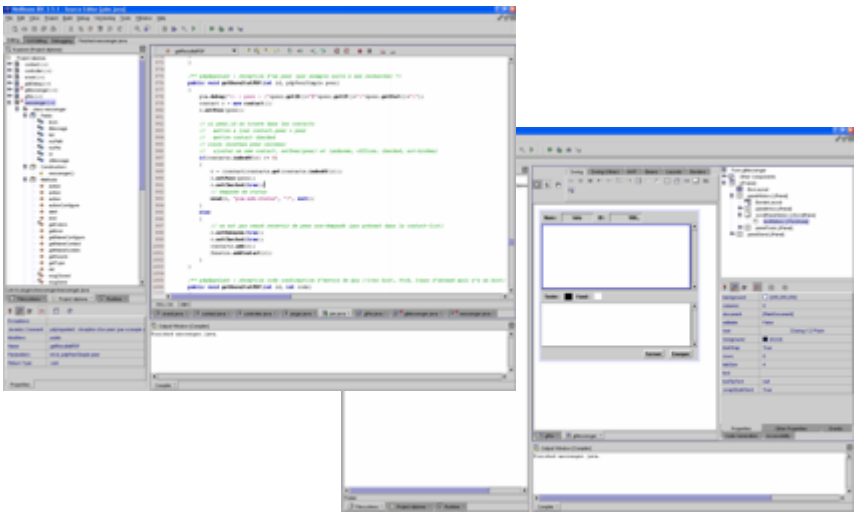


Figure 17. NetBeans IDE 3.5.1

J'ai tout de même dû utiliser un logiciel supplémentaire pour la modélisation UML, NetBeans n'incluant aucun outil de ce type. Le logiciel que j'ai utilisé est Poseidon for UML 2.1 de Gentleware.

J'ai également utilisé un logiciel séparé pour construire l'assistant d'installation pour Windows de mon projet. Le logiciel que j'ai utilisé est Inno Setup 4.0.10, qui est un excellent constructeur d'installeur, gratuit qui plus est.

5.2. Structure générale

Le projet est séparé en deux parties principales, ou modules, auxquels peuvent venir s'ajouter des plugins.

Le premier module, PDP (qui est une abréviation de "Protocole de Découverte de Peers"), est responsable de la gestion du réseau de peers (découverte, recherche, identification, maintenance d'un état à jour des informations du réseau).

Le deuxième module, P2PIM[3] (qui signifie "Peer-to-Peer Instant Messaging") est responsable de la gestion et de l'utilisation, au niveau utilisateur, du réseau de peers sous-jacent.

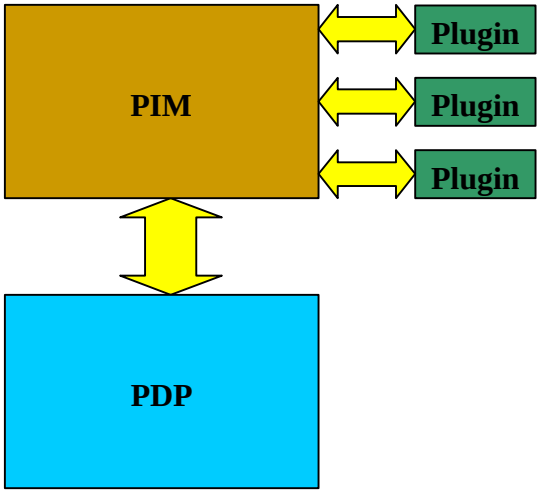


Figure 18. Schéma de la structure générale

Les plugins enfin, qui viennent se greffer sur l'application PIM, constituent les véritables applications ou services (service de messagerie instantanée par exemple) offerts à l'utilisateur final et faisant usage du réseau de peers mis en place par PDP.

5.3. Module PDP

Cette section présente le module PDP, responsable de la gestion du réseau de peers (découverte, recherche, identification, maintenance d'un état à jour des informations du réseau).

5.3.1. Vue d'ensemble

Le module PDP est réalisé dans le paquetage `yb.pdp` qui contient deux classes publiques, `pdp` et `pdpPeerSimple`, une interface, `pdpAppelant`, et un ensemble de classes privées utilisées par le paquetage.

Les tâches principales de PDP sont les suivantes :

[3] Dans mon idée, P2PIM signifiait au départ "Peer-to-Peer Instant Messaging", que j'ai transformé par la suite en PIM pour en faire le nom de mon application. PIM a également l'avantage de pouvoir prendre le sens plus large de "Peers Information Manager".

- découverte de peers
- recherche de peers
- surveillance des peers connus

A cette liste s'ajoute une tâche indispensable à la réalisation des tâches indiquées :
- l'envoi et la réception de messages

Le fonctionnement général du module PDP est le suivant : PDP annonce régulièrement sa présence, écoute les annonces des autres peers, construit une liste des peers présents, demande leur liste aux peers de sa propre liste et vérifie la validité des membres de la liste de peers.

Le module PDP peut être utilisé (ou appelé) par toute classe qui implémente l'interface `pdpAppelant`.

PDP est organisé est un ensemble de classes ou sous-modules relativement indépendants dirigés par la classe principale `pdp`, chaque sous-module étant responsable de l'un des aspects du fonctionnement général du module PDP. Ces classes sont :
- un serveur TCP (`pdpServeurTCP`)
  - qui envoie et reçoit des `pdpMessage` au travers d'une `pdpConnexion`
- un serveur UDP (`pdpServeurUDP` et `pdpEcouteurUDP`)
  - qui envoie et reçoit des `pdpMessage`
- un contrôleur pour la vérification de l'état des peers (`pdpControleur`)
- deux "crawlers" ou "chercheurs" de peers (`pdpCrawler` et `pdpCrawlerUnicast`)

A cette liste viennent encore s'ajouter quelques classes utilitaires ou représentants des types d'objets utilisés par les autres sous-modules (telle que la classe `pdpPeer` représentant un peer du réseau).

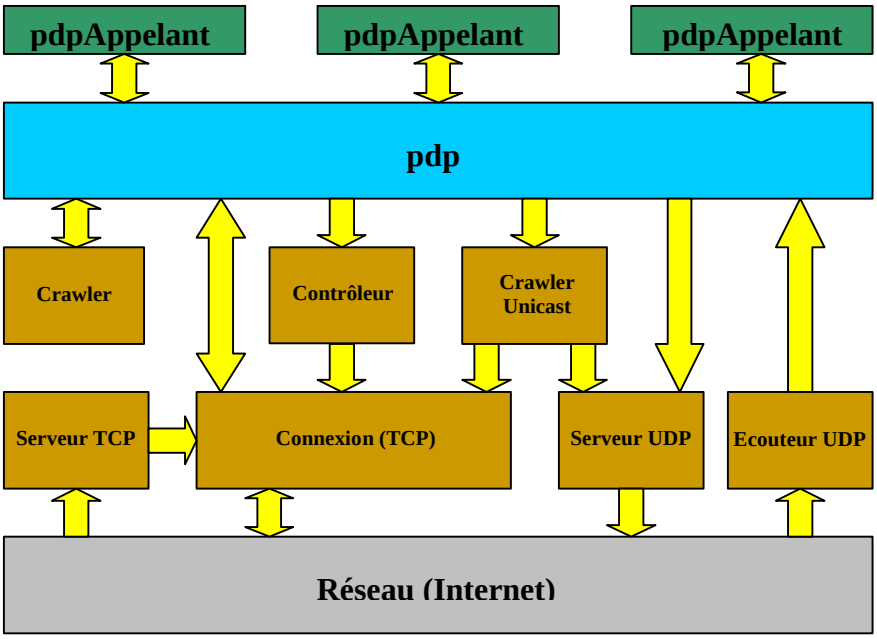


Figure 19. Structure du module PDP

Le point suivant présente les principaux composants de PDP plus en détails.

5.3.2. Détails et sous-modules

Fonctionnement de la classe `pdp`, description du type `pdpPeer`, présentation des techniques de découverte, recherche et gestion des peers.

5.3.2.1. pdp

La classe `pdp` est le chef d'orchestre du module. Son rôle est de coordonner l'action des autres classes, de centraliser les demandes de traitement et de conserver les informations sur l'état du réseau.

Système de requêtes

Tous les messages reçus, aussi bien en provenance du réseau qu'en provenance d'un objet `pdpAppelant` local, conduisent à l'émission d'une requête auprès de la classe `pdp` (une requête est une instance de la classe `pdpRequete`, qui indique le type de requête, les paramètres de la requête, l'appelant local ou l'appelant distant ainsi que les identifiants locaux et distants de cette requête). Ces requêtes sont stockées dans une liste des requêtes en attente et sont traitées selon leur ordre d'arrivée (FIFO).

Différents mécanismes d'appels ont été envisagés et c'est un système de callback qui a été retenu. Les différentes options étudiées étaient les suivantes :

- Appel de procédure normal, avec résultat retourné par la méthode appelée.
  - problème : une demande telle que la recherche d'un peer peut potentiellement prendre beaucoup de temps, ce qui bloquerait inutilement l'appelant.

- Demande de requête avec retour immédiat, le résultat étant ensuite disponible par une méthode/variable d'état.
  - problème : oblige l'appelant à aller consulter régulièrement une variable d'état afin de savoir si un résultat est disponible, de plus ce mécanisme induit un retard dans la réception de la réponse (un résultat peut être disponible depuis un temps quelconque avant que l'appelant ne "pense" à aller en vérifier la disponibilité).

- Utilisation du modèle `Observer`/`Observable` mis à disposition par JAVA. Permet de ne pas bloquer l'appelant, tout en assurant que l'appelant sera informé du résultat dès que celui-ci sera disponible.
  - problème : manque de souplesse (obligation d'utiliser les méthodes prévues par le modèle Observer pour transmettre les résultats), mais l'idée est celle à la base de la solution retenue.

- Requêtes et callback. Les appelants doivent implémenter l'interface `pdpAppelant`. Lors d'une demande, `pdp` enregistre la requête et l'appelant puis retourne immédiatement, en précisant à l'appelant l'identifiant attribué à sa requête. Lorsque le résultat est disponible, `pdp` transmet celui-ci à l'appelant concerné (en incluant le numéro de requête d'origine) par l'appel de l'une des méthodes prévues par l'interface `pdpAppelant`.

C'est cette dernière option qui a été retenue, car elle présente l'avantage de ne pas bloquer les appelants tout en offrant une bonne flexibilité dans la manière de transmettre les résultats (interface spécialement conçue plutôt qu'interface générique `Observer`).

Les requêtes internes où appels en provenance de sous-modules de PDP sont traités comme des requêtes émises par un `pdpAppelant` externe, c'est pourquoi la classe `pdp` elle-même implémente l'interface `pdpAppelant` (au travers de la classe privée `pdpAppelantImpl`).

Données conservées

La classe pdp stock les informations suivantes :
- les options de configuration courantes.
- une liste des requêtes en attente, comme déjà mentionné.
- une liste des requêtes en cours de traitement, utile pour le traitement des requêtes longues (typiquement les demandes de recherche de peers).
- une liste des peers (qui constituent le "réseau" de peers).
- une liste des abonnements (voir sous "Protocole PDP" plus loin dans cette section).
- une liste des clients de relais et un serveur relais (voir sous "Les peers").

5.3.2.2. Les peers

Les peers sont représentés par des instances de la classe privée `pdpPeer`. Un peer est caractérisé par :
- un identifiant publique
- un identifiant privé
- une adresse IP
- un numéro de port
- un relais (optionnel)
- un score
- un état (actif ou non)

Identifiants

Les identifiants publique (`ID`) et privé (`SID`) sont une paire de clés publique/privée. L'identifiant publique est connu des autres peers et sert à identifier un peer et à crypter les messages destinés à ce peer. L'identifiant privé n'est connu qu'en local et n'est jamais transmis, la propriété `pdpPeer.SID` est donc inconnue pour toutes les instances de `pdpPeer` contenues dans la liste des peers, la seule instance de `pdpPeer` pour laquelle ce champ est renseigné est `peerLocal` qui représente les informations du peer local. L'identifiant privé sert à décrypter les messages reçus et à authentifier les messages transmis.

Adresse IP et port

Informations de contact de ce peer.

Remarque : l'adresse IP d'un peer est initialisée automatiquement et risque d'être une adresse classe C privée (telle que 192.168.x.x) si la machine fait partie d'un réseau local ou si son accès à Internet se fait par l'intermédiaire d'un routeur (ADSL typiquement). C'est pourquoi,

lorsque deux peers entrent en contact, ils enregistrent de préférence l'adresse IP source apparente plutôt que l'adresse IP annoncée par leur correspondant.

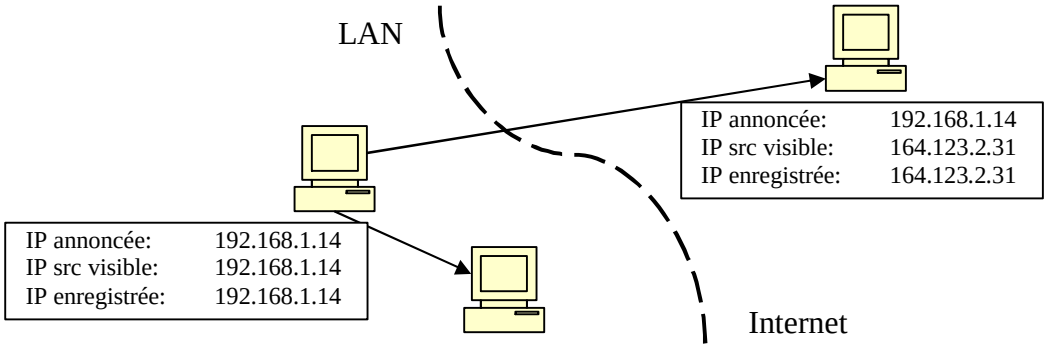


Figure 20. Adresses IP annoncées et visibles

Relais

Ce champ indique un éventuel peer relais pour le peer concerné. C'est-à-dire un peer auquel envoyer les messages destinés au peer concerné (ce peer n'étant pas joignable de manière directe, probablement à cause d'un firewall). Voir sous "Découverte de peers", "Protocole PDP" et "Système de Relais" pour plus de détails sur l'utilisation des relais.

Score et état

Ces deux valeurs sont utilisées pour la gestion des peers.

5.3.2.3. Découverte de peers

La découverte de peers se déroule selon deux méthodes différentes, utilisables simultanément ou non selon les options de configuration, et dont le comportement est légèrement différent si le peer local fonctionne en mode "normal" ou en mode "firewalled" (peer local "caché" derrière un firewall, et donc non joignable de l'extérieur).

Mode normal

Une première technique consiste à s'annoncer en UDP multicast et écouter les réponses d'annonce et annonces d'autres peers sur le même canal multicast. Les peers annoncés sont ajoutés à la liste des peers et on leur demande leur liste de peers, ce qui permet d'étendre plus rapidement le nombre de peers découverts (méthode implémentée dans `udpCrawler`).

En théorie, cette technique est idéale, mais en pratique le trafic multicast est souvent bloqué entre réseaux différents, ce qui limite l'utilisation de cette méthode. Elle reste toutefois très efficace en réseau local.

La deuxième technique utilisée par PDP consiste à essayer de s'annoncer en UDP unicast auprès d'adresses IP pseudo aléatoires (en réalité les adresses testées sont croissantes) et d'adresses de broadcast d'autres réseaux. Si un peer répond, il est ajouté à la liste et on lui demande sa liste de peers (cette méthode est implémentée dans `udpCrawlerUnicast`).

Mode "firewalled"(relais)

Ce mode de fonctionnement est utilisé lorsque le peer local se trouve derrière un firewall (ou un autre dispositif) qui bloque les messages entrants (il est impossible de contacter le peer depuis l'extérieur).

Dans ce cas de figure, la priorité est de trouver un peer acceptant de servir de relais. Pour ce faire, les deux méthodes modifient leur comportement.

La première technique remplace ses demandes de liste de peers aux peers connus par des demandes de relais (les annonces UDP multicast sont maintenues), tandis que la deuxième technique remplace l'envoi d'annonces de présence UDP par des tentatives de demande de relais en TCP (ce qui est plus lent, mais est nécessaire afin d'établir une connexion par laquelle l'éventuel correspondant pourra répondre et peut-être accepter de devenir le relais du peer local).

Ces méthodes, surtout la deuxième, peuvent prendre un certain temps avant de découvrir un autre peer, d'autant plus en mode "firewalled". C'est pourquoi il est important de garder trace des anciens peers devenus invalides et d'essayer de les contacter à nouveau périodiquement. Ce travail de gestion de la liste des peers est assuré par le `pdpControleur` qui est décrit dans le point suivant.

5.3.2.4. Gestion des peers

La gestion de la liste des peers est réalisée par la classe `pdpControleur`. L'idée de base est d'attribuer un score à chaque peer, score qui doit indiquer la qualité du peer concerné. Plus un peer est présent (disponible) longtemps et/ou plus il est utile (il répond aux demandes de recherche, il fournit de nouveaux peers), plus son score sera élevé.

A intervalles réguliers, le contrôleur lance un mécanisme de vieillissement des peers qui diminue les scores de tous les peers, ceci afin de se débarrasser des peers les moins intéressants de la liste.

Le contrôleur essaie également de contacter tous les peers de la liste (aussi bien les peers actifs que les peers inactifs) afin de déterminer leur état. Si un peer marqué comme actif ne répond plus, il est marqué comme invalide et son score est diminué. Au contraire, si un peer marqué comme inactif répond à nouveau, il est repassé à l'état actif et son score est augmenté.

Le fonctionnement du contrôleur est modifié par divers paramètres. Le contrôleur peut décider de ne vieillir que les peers inactifs si la liste est par trop remplie de tels peers. Le contrôleur peut aussi ne pas vieillir du tout les peers si la liste est trop peu remplie (cas dans lequel il est important de garder les rares peers que l'on a réussi à trouver).

Le fonctionnement du contrôleur peut être paramétré en modifiant les variables adéquates du fichier de configuration (bonus/malus de score, seuils de nettoyage, etc. Voir sous "Configuration").

Afin de faciliter la découverte de peers, PDP conserve la liste des peers découverts d'une exécution à l'autre. En effet, il y a une forte probabilité qu'un peer soit à nouveau présent plus tard (surtout si l'intervalle entre deux utilisations du module PDP est court).

En plus de la gestion de la liste des peers, le contrôleur s'occupe également de :

- vérifier l'état des éventuels connexions vers nos clients relais (peers pour lesquels le peer local fait office de relais).

- vérifier la connexion vers un éventuel serveur relais (peer faisant office de relais pour le peer local).

- vérifier la liste des requêtes en attente de traitement : suppression des dernières requêtes émises si la liste est trop remplie.

- vérifier la liste des requêtes en cours de traitement : suppression des requêtes les plus anciennes s'il y a trop de requêtes (ce sont probablement des requêtes "mortes", par exemple des demandes de recherche qui n'ont pas abouti).

5.3.2.5. Recherche d'un peer

La recherche d'un peer d'ID donnée est très simple. Lors de la réception d'une requête de recherche, PDP vérifie tout d'abord si le peer demandé est contenu dans la liste de peers :

- si tel est le cas le résultat est transmis et le traitement de cette requête est terminé.

- sinon PDP transmet la requête aux peers de sa liste

Les requêtes de recherche sont accompagnées d'un identifiant permettant de détecter les boucles (auquel cas la requête n'est pas propagée), ainsi que d'un compteur de durée de vie (TTL : time to live) pour limiter la propagation de la recherche.

A noter que les réponses à une demande de recherche ne sont pas transmises directement au peer original, mais "remontent" jusqu'à ce peer en empruntant le chemin inverse du chemin parcouru par la demande de recherche (voir sous "Protocole PDP" pour un schéma illustrant ce mécanisme).

5.3.2.6. Service de transmission de messages

Le module PDP offre un service de transmission de messages. Les messages envoyés et reçus sont associés à une catégorie et lors de la réception d'un message, PDP remet le message ainsi que les informations sur l'émetteur du message aux objets `pdpAppelant` inscrits à la catégorie concernée.

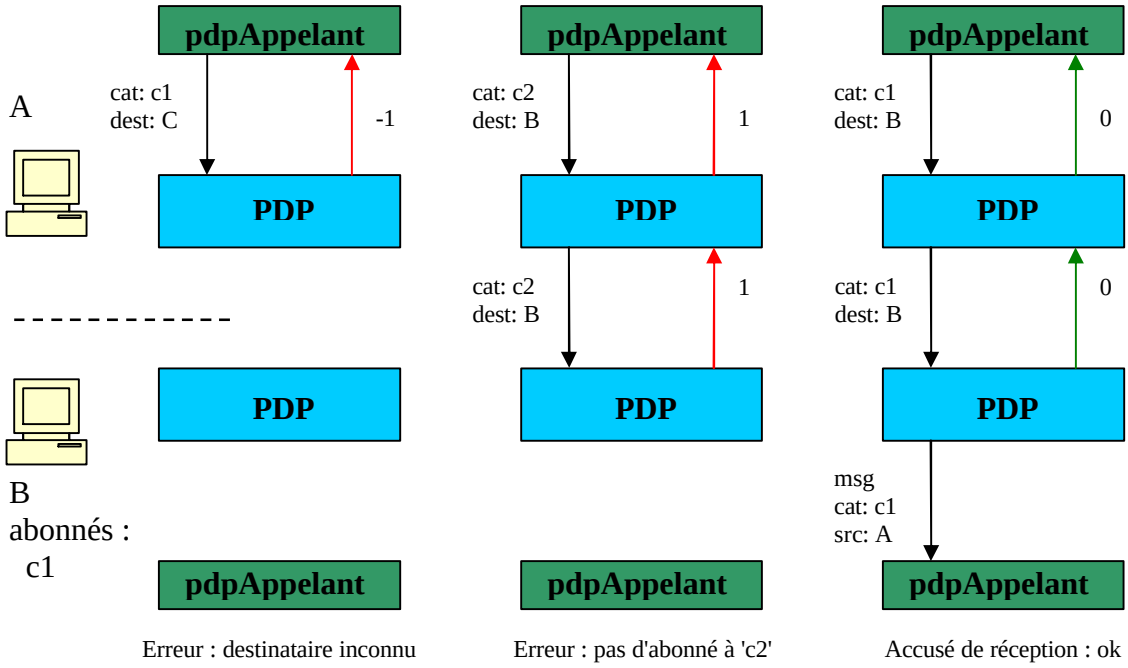


Figure 21. Service de transmission de messages de PDP

Le service de messages comprend aussi un système d'accusés de réception / d'erreur qui signale les messages transmis avec succès, les messages impossibles à transmettre (par exemple parce que le peer indiqué comme destinataire est inconnu) et les messages transmis avec succès au peer de destination, mais pour lesquels il n'y avait aucun abonné.

5.3.2.7. Protocole PDP

Les points suivants présentent le protocole de communication de PDP. Les éléments marqués en *italique* sont les variables (données) transmises. Les éléments entre crochets carrés [ ] sont optionnels. Le symbole 'X' représente le numéro de version du protocole, transmis avec chaque message. La version actuelle du protocole est la version 3.0 qui ajoute la gestion des relais à la version 2.0 du protocole, qui elle-même ajoutait la partie "Messages" à la version initiale du protocole.

Connexion / identification / salutations

```
pdp X hello id ip port [relais id ip port]
pdp X byebye code                          code : 0 = normal, -1 = erreur
```

Initiation et terminaison de communication. Le message `hello` précise l'identité du peer ainsi que son éventuel relais.

Exploration

```
pdp X exploration annonce id ip port [relais id ip port]
pdp X exploration reponse id ip port [relais id ip port]
pdp X exploration demande-liste id-requete dest-peer-id
```

```
pdp X exploration envois-liste id-requete dest-peer-id {id ip port [relais
id ip port]}+
```

Annonce d'un peer, réponse à une annonce (le peer répondant s'annonce par la même occasion), demande de liste des peers, transmission d'une liste de peers.

Recherche

```
pdp X recherche demande id-requete dest-peer-id id-peer-cherché ttl aloopid
pdp X recherche trouve id-requete dest-peer-id id ip port [relais id ip
port]
```

Demande de recherche d'un peer par son ID, réponse positive contenant les informations sur le peer trouvé (il n'existe pas de message permettant de répondre négativement à une demande de recherche, une recherche est considérée comme "négative" tant qu'aucune réponse positive n'est parvenue au peer source).

Messages

```
pdp X message envois id-requete dest-peer-id categorie message id-src ip
port [relais id ip port]
pdp X message reponse id-requete dest-peer-id crypt(id-requete & code) code
      code : 0 = message remis
            -1 = pas d'hôte à l'adresse indiquée
             1 = hôte ok, mais pas d'abonné à cette catégorie de message
```

Cette catégorie de messages, contrairement aux autres, n'est pas utilisée par PDP lui-même mais sert de support au service de transmission de messages offert aux objets `pdpAppelant` externes. Cet aspect est présenté plus en détails sous "Service de transmission de messages", plus haut dans ce document.

Relaying

```
pdp X relais demande
pdp X byebye code [{id ip port [relais id ip port]}+]
                                          code : 2 = non + liste, 3 = ok
```

Demande de relais et extension du message `byebye` pour y inclure les nouvelles réponses possibles (code 3 qui signifie que le peer contacté accepte de devenir le relais du peer source, et code 2 qui signifie que le peer contacté refuse de devenir un relais mais qu'il joint sa liste de peers à la réponse, ce qui peut aider le peer source à découvrir un relais).

Remarques

Les messages incluent un champ *dest-peer-id* qui permet de déterminer le véritable destinataire du message lors de l'utilisation de relais (dans le cas "normal" le destinataire est directement le peer auquel le message est transmis).

Le champ *id-requete* indique le numéro de référence de la requête ayant provoqué l'envoi du message. Lorsqu'un peer répond à un message (par exemple réponse à une demande de

recherche), il indique ce numéro afin que le peer destinataire sache de quoi il s'agit. Si un peer transmet un message plus loin, il génère et associe son propre numéro de requête au message transmis et mémorise les informations nécessaires (dans des objets `pdpRequete`) pour pouvoir transmettre dans l'autre sens une éventuelle réponse.

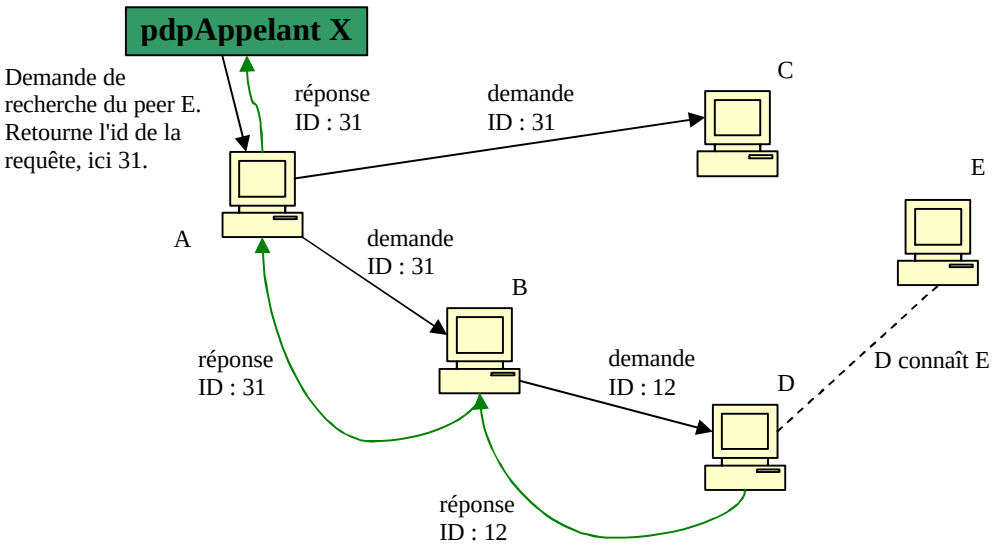


Figure 22. Propagation d'un message (par ex. recherche)

Dans l'exemple ci-dessus, le module PDP du peer A reçois l'ordre d'un objet `pdpAppelant` d'effectuer une recherche. Voici ce qui se passe suite à cet appel :

- A enregistre la requête, qui contient l'appelant (X) et l'identifiant de cette requête (31, qui est immédiatement retourné à X). A transmet ensuite le message aux peers qu'il connaît.

- Le peer B reçoit la demande, il enregistre donc une requête accompagnée de l'id distant (31), du peer distant (A) ainsi que de l'id local sous lequel B connaît cette requête (12), puis B transmet le message vers D.

- Lorsque D reçoit la demande, il n'enregistre rien puisqu'il connaît la réponse. D envoie simplement la réponse, identifiée par l'id indiqué dans la demande (12).

- B reçoit le message, recherche dans sa liste de requêtes en cours, trouve que l'id de requête 12 correspond à la requête 31 en provenance de A et transmet donc la réponse à A en remplaçant l'id 12 par l'id 31 (et supprime la requête de sa liste).

- A reçoit le message, recherche dans sa liste, découvre qu'il s'agit d'une réponse pour l'appelant local X et lui transmet cette réponse. Pour terminer, A supprime la requête de sa liste de requêtes en cours.

Remarque : si le peer C venait à répondre au peer A par la suite, ce dernier ne saurait plus de quoi il s'agit et le message serait simplement ignoré.

Les messages sont transmis en TCP, excepté les messages d'annonce de présence qui peuvent être transmis en UDP. Les communications sont très simples, et suivent toujours le déroulement suivant :

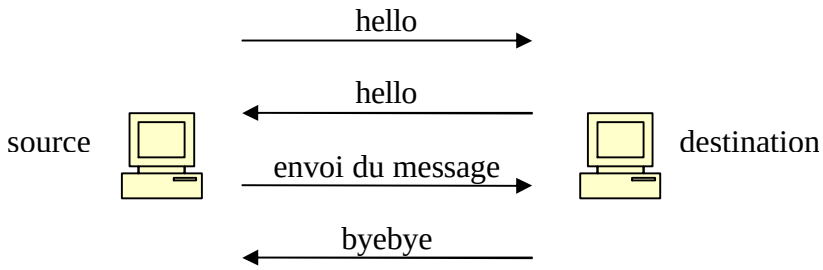


Figure 23. Communication selon le protocole PDP

5.3.2.8. Système de Relais

En situation "normale" (idéale) PDP fonctionne en peer-to-peer direct. Les transmissions se déroulent comme sur la figure ci-dessus :

- une connexion est établie
- la communication a lieu
- la connexion est fermée

La version 3.0 du protocole a ajouté un système de relais à PDP, qui permet aux peers non accessibles (messages entrants bloqués) de participer au réseau malgré tout.

Ces peers commencent par essayer de trouver un relais comme expliqué sous "Découverte de peers", puis incluent ce peer relais dans le champ "relais" de leurs annonces de présence. Le peer "client" et le peer "serveur" (ou relais) maintiennent une connexion ouverte par laquelle le relais pourra transmettre au peer "client" les messages qui lui sont destinés.

Par la suite, lorsqu'un peer cherche à contacter un peer munis d'un relais, le message est envoyé au relais plutôt qu'au véritable peer destinataire, en indiquant dans le champ *dest-peer-id* du message l'identifiant du véritable destinataire. Lorsqu'un relais reçoit un message qui ne lui est pas destiné, il consulte sa liste de "clients" et si l'un de ceux-ci correspond, il lui transmet le message. Sinon, le message est supprimé et un éventuel message d'erreur est retourné (uniquement dans le cas d'un message de type "message").

Dans une situation avec relais, les messages sont échangés selon la figure ci-dessous :

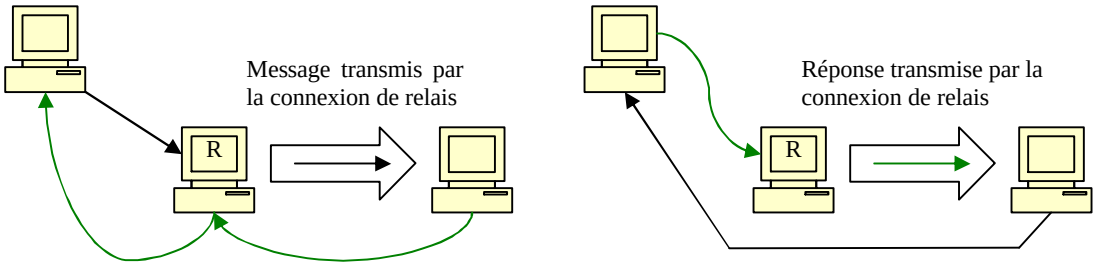


Figure 24. Echange de messages avec utilisation d'un relais

Dans le cas de gauche, les réponses se propagent comme dans le cas normal, en remontant le chemin parcouru par la demande : le destinataire répond au peer d'où il a reçu le message, ici son relais, et non pas directement au peer source.

Le cas de droite présente le cas d'une réponse nécessitant l'utilisation d'un relais, alors que la demande s'était propagée sans relais. Le peer destinataire sait qu'il doit répondre en utilisant un relais, car le peer source le lui a indiqué durant la phase d'identification (messages de type "hello").

Un cas "amusant" est illustré ci-dessous :

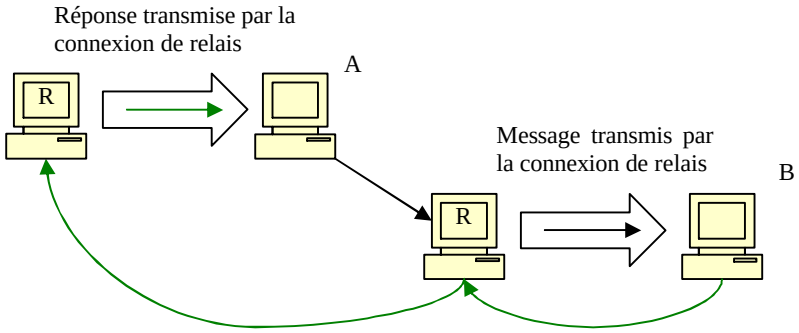


Figure 25. Echange de message entre deux peers avec relais

Ce cas se produit lorsqu'un peer "relayé" (A) désire communiquer avec un autre peer lui aussi "relayé" (B).

5.3.3. API de PDP

Ce point détaille les méthodes et classes publiques du module PDP, ainsi que l'interface `pdpAppelant` à implémenter pour utiliser PDP. Le paquetage `yb.pdp` met à disposition deux classes publiques et une interface :

- `yb.pdp.pdp` : classe principale, contient toutes les méthodes utilisables par les appelants externes.
- `yb.pdp.pdpPeerSimple` : type représentant un peer du réseau.
- `yb.pdp.pdpAppelant` : interface à implémenter par les utilisateurs de PDP.

5.3.3.1. yb.pdp.pdp

Tableau 4. API de yb.pdp.pdp

`public` **`pdp`**`(String pid, String sid)` Constructeur, crée une nouvelle instance de `pdp`. Paramètres : pid identifiant de ce peer (clé publique) sid clé privée de ce peer, utilisée pour crypter les communications
`public int` **`rechercher`**`(String id_peer, pdpAppelant appelant)`

Enregistre une demande de recherche et l'appelant à qui transmettre l'éventuel résultat .
Paramètres :
- id_peer identifiant du peer recherché
- appelant objet à qui transmettre la réponse

Retourne :
un identifiant de requête (0 en cas d'erreur), l'éventuel résultat sera accompagné de cet identifiant.

```
public Vector getPeers()
```

Retourne la liste des peers
Retourne :
la liste des peers sous forme d'un `Vector` de `pdpPeerSimple.`
Remarque :
utilisait initialement `pdpAppelant.getResultatPDP(int, Vector)`.

```
public void abonner(String categorie, pdpAppelant appelant)
```

Enregistre l'objet appelant comme abonné aux messages de la catégorie indiquée
Paramètres :
- categorie catégorie de messages à laquelle s'inscrire
- appelant objet qui s'inscrit à cette catégorie

```
public void desabonner(String categorie, pdpAppelant appelant)
```

Résilie un abonnement
Paramètres :
- categorie catégorie de messages de laquelle se désinscrire
- appelant objet qui annule son abonnement

```
public int envoyer(   pdpPeerSimple peer,
                      String message,
                      String categorie,
                      pdpAppelant appelant)
```

Enregistre une demande d'envoie de message et l'appelant à qui remettre l'éventuel résultat.
Paramètres :
- peer peer à qui envoyer le message
- message message à transmettre
- categorie catégorie du message à transmettre
- appelant objet à qui transmettre la réponse

Retourne :
un identifiant de requête (<= 0 en cas d'erreur), l'éventuel résultat sera accompagné de cet identifiant.

```
public void setOption(String clef, String valeur)
```

Enregistre une nouvelle valeur pour une option.
Paramètres :
- clef la clef à entrer dans la liste des options (nom de l'option)
- valeur la valeur de cette option

`public String` **`getOption`**`(String clef)` Retourne la valeur d'une option. Paramètres : clef la clef (le nom) de l'option dont on veut la valeur Retourne : la valeur de l'option demandée
`public void` **`demarrer`**`()` Démarre l'exécution de PDP.
`public void` **`arreter`**`()` Met fin à l'exécution de PDP.

5.3.3.2. yb.pdp.pdpPeerSimple

Le seul but de cette classe est de servir de type d'échange entre PDP et les objets `pdpAppelant`. Elle ne contient donc que très peu de méthodes et n'est accessible qu'en lecture.

Tableau 5. API de yb.pdp.pdpPeerSimple

`public` **`pdpPeerSimple`**`( String idPeer,`
`String ipPeer,`
`int portPeer,`
`pdpPeerSimple relais)`
 	
Constructeur, crée une nouvelle instance de `pdpPeerSimple` en précisant l'ID, l'IP, le port et l'éventuel relais.
Paramètres :
idPeer identifiant du peer
ipPeer adresse IP du peer
portPeer port du peer
relais peer servant de relais (peut valoir 'null')

`public` **`pdpPeerSimple`**`(pdpPeerSimple source)` Constructeur, crée une nouvelle instance de `pdpPeerSimple`. Paramètres : source le peer à copier
`public String` **`getIP`**`()` Retourne : l'IP du peer
`public String` **`getPort`**`()` Retourne :

Le port du peer
`public String` **`getID`**`()` Retourne : l'ID du peer
`public String` **`getRelais`**`()` Retourne : l'éventuel relais du peer

5.3.3.3. yb.pdp.pdpAppelant

Interface que les objets désireux d'utiliser PDP doivent implémenter. Les méthodes de cette interface sont appelées par PDP lorsqu'il y a un résultat à transmettre à l'appelant.

Tableau 6. Interface yb.pdp.pdpAppelant

`static int` **`ACCUSE_OK`** `static int` **`ACCUSE_PAS_HOTE`** `static int` **`ACCUSE_PAS_ABONNE`** Accusés de réception : - ok : message bien transmis (0) - pas d'hôte : aucun hôte de destination (-1) - pas d'abonné : pas d'abonné sur l'hôte de destination (1)

`public void` **`getResultatPDP`**`(String cat,`
`String msg,`
`pdpPeerSimple src)`
 	
Signal un message.
Paramètres :
cat la catégorie de ce message
msg le message
src le peer source

`public void` **`getResultatPDP`**`(int id, int code)` Signal un accusé de réception (suite à l'envoi d'un message). Paramètres : id l'identifiant de la requête d'origine code -1 = pas d'hôte, 0 = ok, 1 = pas d'abonné sur l'hôte de destination
`public void` **`getResultatPDP`**`(int id, pdpPeerSimple resultat)` Transmission d'un peer (suite à une demande de recherche). Paramètres : id l'identifiant de la requête d'origine resultat le peer demandé

```
public void getResultatPDP(int id, Vector resultat)
```

Transmission d'une liste de peers (suite à une demande de liste).
Paramètres :

id	l'identifiant de la requête d'origine
resultat	la liste des peers

5.3.4. Configuration

Le comportement de PDP est configuré par le fichier `pdpDefaut.cfg` qui se trouve dans `yb/pdp/`. Ce comportement par défaut peut être modifié en créant un fichier de configuration, `pdpConfiguration.cfg`, à placer dans le répertoire de base du compte d'utilisateur (par exemple dans "/users/toto" sous UNIX ou "C:\Documents and Settings\toto" sous Windows XP).

Les options reconnues sont les suivantes :

Tableau 7. Options de configuration de PDP

Option	Description	Valeur par défaut
NO_PORT	numéro de port	4043
GROUPE	groupe multicast UDP	226.5.4.3
SEPARATEUR_1	séparateurs de champs (fichier des peers)	###
SEPARATEUR_2		#relais#
TTL	ttl des messages de recherche	7
ANTIBOUCLE_N	taille de la liste anti-boucle pour les recherches	20
RECHERCHE_MAX_CIBLE	propagation des recherches (nombre de peers auxquels transmettre la demande)	15
CONTROLEUR_DELAIS	période d'activation du contrôleur (en ms)	55000
PEERS_MAX	taille maximale de la liste de peers	100
REQUETES_MAX_ATTENTE	nombre maximal de requêtes en attente	100
REQUETES_MAX_TRAITEMENT	nombre maximal de requêtes en cours de traitement	100
PEER_SCORE_INITIAL	score initial d'un nouveau peer	40
PEER_SCORE_MAX	score maximal d'un peer	600
PEER_SCORE_MALUS_AUTO	vieillissement des peers actifs	-4
PEER_SCORE_MALUS_EFFACER	malus si peer devient invalide	-2
PEER_SCORE_MALUS_AUTO_INACTIF	vieillissement des peers inactifs	-2
PEER_SCORE_BONUS_HELLO	bonus pour message `hello`	2
PEER_SCORE_BONUS_REPONSE	bonus pour réponse à une recherche / liste de peers	30
ACTIFS_ALERTE	si la liste est pleine mais qu'elle contient moins de ce seuil de peers	25

	actifs, nettoyage des peers inactifs	
PEERS_MIN	si la liste contient moins de ce seuil de peers, le vieillissement n'a plus lieu	10
SEARCH_DELAIS	période d'activation du "crawler" simple (multicast et demandes de liste) (en ms)	30000
SEARCH_ALERTE	si la liste contient moins de ce seuil de peers, la recherche a lieu	10
SEARCH_ALERTE_ACTIFS	si la liste contient moins de ce seuil de peers actifs, la recherche a lieu	5
SEARCH_FORCEE	toutes les *n* fois, la recherche a lieu (même si aucun des critères ci-dessus n'est rempli)	10
SEARCH_MULTICAST	mode de recherche par crawler simple	true
SEARCH_FIREWALLED	mode "firewalled" (recherche d'un relais)	false
SEARCH_UNICAST	mode de recherche par crawler unicast (aléatoire)	false
SEARCH_UNICAST_DELAIS	période d'activation (en ms)	11000
SEARCH_UNICAST_TAILLE_LOT	nombre d'IP à tester par invocation du crawler unicast	8500
RELAYING_AUTHORISE	indique si ce peer est autorisé à fonctionner comme relais	true
RELAYING_POSSIBLE	modifié par PDP durant l'exécution	false
RELAYING_MAX_CLIENTS	nombre maximal de "clients" acceptés par ce peer	10
RELAYING_MIN_ACTIFS	nombre minimal de peers actifs que ce peer doit connaître pour être autorisé à servir de relais	5
DEBUG	0 = rien 1 = affichage sur stdout 2 = fichier 3 = fichier + stdout	0 2 et 3 non implémentés

Le format de ce fichier est le suivant :

```
Option1 = valeur
Option2 = valeur
...
```

PDP utilise un deuxième fichier, `pdpPeers.cfg`, situé dans le même répertoire et qui contient la liste des peers découverts.

5.4. Module P2PIM (PIM)

Cette section présente le module PIM, qui est une application graphique construite au-dessus du réseau crée par le paquetage `yb.pdp`.

5.4.1. Vue d'ensemble

PIM offre une gestion visuelle des peers auxquels l'utilisateur s'intéresse. Les peers peuvent être ajoutés, supprimés et "utilisés" (au travers de l'un des plugins installés). Le module PIM en soit se contente d'afficher, de maintenir et de permettre la gestion de la liste des contacts.

Le module PIM est réalisé dans le paquetage `yb.p2pim` qui contient 8 classes, dont 2 sont publiques (`pim` et `contact`), ainsi qu'une interface, `plugin` (les plugins de PIM sont des classes qui implémentent cette interface).

Les éléments principaux de PIM sont :

- la classe `pim`, qui gère le fonctionnement général de l'application, et qui contient :
  - le dialogue `gIdDialog` qui démarre PIM
  - la classe interne `pimReceiver` qui gère les messages en provenance du réseau
- un contrôleur (classe `controller`), qui gère les contacts
- la classe `gPim` qui s'occupe de l'interface graphique

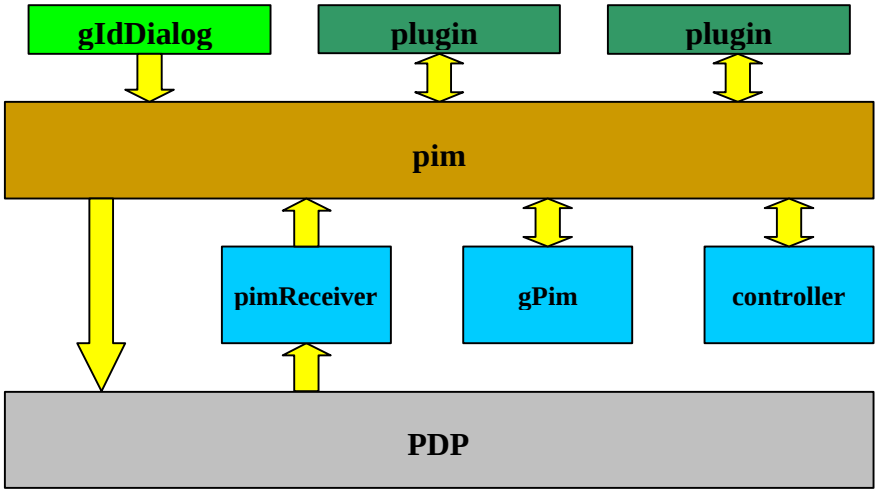


Figure 26. Structure du module PIM

A cette liste, il faut ajouter quelques classes représentant des types d'objets, notamment :

- `contact`, qui représente un des éléments de la liste des contacts (en gros cela correspond à un peer)
- `event`, qui représente un événement reçu du réseau

5.4.2. Détails et sous-modules

Ce point décrit le fonctionnement des principaux éléments du module PIM.

5.4.2.1. pim

La classe `pim` commence par être initialisée avec les valeurs fournies par le dialogue de login (`gIdDialog`), puis un objet `yb.pdp.pdp` est créé, après quoi les sous-modules de PIM sont initialisés, l'interface graphique (`gPim`) est affichée et enfin l'objet `pdp` est démarré.

Après cette phase de démarrage, la classe `pim` devient passive et se contente de réagir aux appels (événements) reçus.

Traitement des événements

Les événements à traiter peuvent être les suivants :
- message du réseau :
  - destiné à PIM lui-même (voir sous "Gestion des contacts")
  - destiné à un plugin : PIM détermine le plugin concerné, puis transmet le message ou le met de côté pour plus tard, selon le mode de fonctionnement du plugin (voir la section consacrée aux Plugins)
- appel interne :
  - depuis l'interface (typiquement, ajout ou suppression d'un contact)
  - depuis le contrôleur (typiquement, envoi d'un message)
- appel externe :
  - depuis un plugin (le plus souvent, envoi d'un message)

Les événements sont traités immédiatement par PIM, il n'y a pas de liste des requêtes en attente comparable à celle de PDP.

Données conservées

La classe `pim` stocke les informations suivantes :
- les options de configuration courantes.
- les informations d'identification du contact local.
- la liste des contacts.
- la liste des plugins.
- une liste des abonnements : chaque plugin peut s'abonner auprès de PIM aux catégories de messages voulues, et PIM s'abonne ensuite à l'ensemble de ces catégories auprès de PDP (les plugins n'ont pas accès directement à PDP).
- une liste des messages envoyés pour le compte des plugins : permet de remettre les accusés de réception éventuellement transmis par PDP au bon plugin source.
- une liste des événements (messages) mis de côté (voir la section consacrée aux Plugins).

Les options de configuration, la liste des contacts et la liste des messages mis de côtés sont enregistrées sur disque pour chaque utilisateur de PIM lorsque l'application termine son exécution. Ces options seront rechargées lors de la prochaine exécution de PIM par le même utilisateur.

5.4.2.2. Les contacts

Les contacts sont représentés par des instances de la classe publique `contact`. Un contact est caractérisé par :
- un peer
- un statut
- des propriétés optionnelles
- trois indicateurs booléens :
  - checked

- unknown
- hidden

Peer associé

Le peer, au format `yb.pdp.pdpPeerSimple`, représenté par ce contact.

Statut

Statut du contact, tel que "absent", "en ligne". Les valeurs autorisées sont définies dans la classe `contact`.

Propriétés optionnelles

Cet attribut, accessible également par les plugins, permet d'enregistrer des couples (clé, valeur) dans un objet contact (par exemple des informations supplémentaires, telles que le "profil" de ce contact). La seule clé utilisée par PIM est "`displayName`", qui contient le nom sous lequel afficher ce contact dans la liste des contacts.

Indicateurs booléens

Les trois attributs `checked`, `unknown` et `hidden` indiquent respectivement si le peer associé à un contact est valide (champ utilisé par le contrôleur), si le contact est membre de la liste de l'utilisateur et si le contact est visible dans la liste ou caché.

5.4.2.3. Les événements

Les messages reçus du réseau sont traités immédiatement, sauf dans le cas où le message est destiné à un plugin qui fonctionne en mode `plugin.EVENTS_USER`, auquel cas le message est conservé sous forme d'un objet de la classe privée `event` (voir la section dédiée aux Plugins). Un événement contient :

- le message
- la catégorie du message
- le peer ayant envoyé ce message

Les événements mis de côté et non encore consommés sont enregistrés dans la liste des événements, qui est sauvegardée sur disque si nécessaire (lorsque PIM se termine).

5.4.2.4. Gestion des contacts

La gestion des contacts est assurée conjointement par la classe `controller` et par la classe `pim`. Le contrôleur vérifie périodiquement la liste des contacts de la manière suivante :

- les contacts valides (`checked`) sont interrogés quand à leur statut et marqués comme invalides.
- les contacts déjà invalides sont passés en mode `contact.OFFLINE`, et le contrôleur émet une demande de recherche du peer concerné auprès de PDP (au travers de `pim`).

Lorsqu'un message d'annonce de statut est reçu par PIM, le contact source est marqué comme valide et son statut est mis à jour.

Tableau 8. Protocole de contrôle des contacts de PIM

Catégorie	Message	Remarque
`pim-status`	Un entier	Indique le statut du contact source
`pim-ask-status`	- (ignoré)	Demande de statut

Le petit protocole de messages utilisé pour maintenir le statut des contacts est décrit dans le tableau ci-dessus.

En plus de ce mécanisme spécifique de contrôle, les échanges de messages habituels permettent également à PIM de contrôler l'état des contacts. En effet :

- si un message envoyé provoque une erreur (accusé de réception négatif), le contact est marqué comme invalide.
- si PIM reçoit un message (quelconque), le contact source est marqué comme valide.

5.4.2.5. Interface graphique

L'interface graphique de PIM est implémentée par la classe `yb.p2pim.gPim`.

Cette classe définit quelques classes privées très simples qui étendent `JLabel` et `JMenuItem` afin d'afficher correctement les contacts de la liste et les plugins dans les différents menus. En particulier, redéfinition de `getText()` et `getIcon()` combinés à l'utilisation d'un `Timer` afin de produire les animations de la liste des contacts.

La classe `gPim` définit également tous les gestionnaires d'événements nécessaires pour "écouter" les actions de l'utilisateur.

Cette classe définit enfin les objets nécessaires pour afficher la liste des contacts (définition d'un objet `ListCellRenderer` et d'un `ListModel` à utiliser avec la `JList` des contacts).

Toutes les "décisions" sont prises par la classe `pim`. La classe `gPim` se contente de transformer les événements utilisateurs en appels de méthodes de la classe `pim`. Si ces appels nécessitent une modification de l'interface, `pim` indique les modifications à effectuer et provoque le re-affichage (`repaint`).

Figure 27. PIM

Pour les détails d'utilisation de PIM, voir l'aide en ligne disponible sur le site de PIM à l'adresse `http://change.to/pim`.

Remarque

Dans une situation "simple" (application dirigée par les événements de l'utilisateur), toutes les opérations se déroulent automatiquement dans le thread de traitement des événements (le "Event-Dispatching Thread") ce qui tombe bien puisque seul ce thread devrait être autorisé à modifier des composants graphiques.

Malheureusement, PIM n'est pas une application "simple", car elle doit pouvoir réagir et modifier son interface graphique en réponse à des événements non standard (événements provenant du réseau). C'est pourquoi la classe `gPim` contient de petites classes privées implémentant `Runnable`, utilisées au travers d'appels à la méthode `SwingUtilities.invokeLater()`, qui permettent de garantir l'exécution des mises à jour de l'interface à l'intérieur du "Event-Dispatching Thread".

5.4.3. API de PIM

Ce point détaille les principales classes et méthodes offertes par PIM et présente l'interface à respecter par les plugins (`yb.p2pim.plugin`).

Le paquetage yb.p2pim met à disposition deux classes publiques et une interface :
- `yb.p2pim.pim` : classe principale, contient les méthodes utilisables par les plugins.
- `yb.p2pim.contact` : type représentant un contact de la liste.
- `yb.p2pim.plugin` : interface à implémenter par les plugins pour PIM.

5.4.3.1. yb.p2pim.pim

Tableau 9. API de yb.p2pim.pim

`public` **`pim`**`(String id, String sid)` Constructeur, crée une nouvelle instance de pim. Paramètres : id identifiant publique du peer sid identifiant secret du peer
`public String` **`getPath`**`()` Retourne : le chemin d'accès au répertoire de configuration de l'utilisateur actuellement connecté, de forme "{user}/.pim/{id}/". Les plugins peuvent par ex. sauver leurs fichiers d'options ici.
`public contact` **`getLocal`**`()` Retourne : le contact local. La seule opération d'écriture autorisée sur ce contact est `setProperty` (définir une propriété) qui permet aux plugins de sauver des options directement dans le contact.

`public java.util.Vector` **`getContacts`**`()`

Retourne :
la liste des contacts.

`public int` **`getStatus`**`()`

Retourne :
le statut du contact local (c'est-à-dire le statut de PIM).

`public void` **`setStatus`**`(int status)`

Change le statut du client local.
Paramètres :

status	le nouveau statut

`public void` **`register`**`(String cat, plugin source, int mode)`

Enregistre un plugin comme abonné à une catégorie d'événements.
Paramètres :

cat	la catégorie de messages auxquels ce plugin s'intéresse
source	le plugin qui s'inscrit
mode	le mode de transmission des messages au plugin, qui peut être `plugin.EVENTS_USER, EVENTS_DIRECT` ou `EVENTS_BOTH`

`public void` **`unregister`**`(String cat, plugin source, int mode)`

Supprime une inscription.
Paramètres :

cat	la catégorie de messages auxquels ce plugin ne s'intéresse plus
source	le plugin qui se désinscrit
mode	plugin.EVENTS_USER, EVENTS_DIRECT ou EVENTS_BOTH

`public int` **`send`**`(contact dest, String cat, String msg, plugin source)`

Envois un message à un peer.
Paramètres :

cat	le type de message
msg	le message
source	le plugin source

Retourne :
l'identifiant du message envoyé (<= 0 en cas d'erreur)

`public String` **`getProperty`**`(String key)`

Retourne l'option demandée.
Paramètres :

key	nom de l'option demandée

Retourne :

la valeur de l'option demandée, ou "" si non trouvée
`public void` **`setProperty`**`(String key, String value)` Définit l'option indiquée. Paramètres : key le nom de l'option à définir value la valeur de l'option à définir
`public String` **`getNetworkProperty`**`(String name)` Retourne l'option du réseau sous-jacent (PDP) demandée. Paramètres: name nom de l'option demandée Retourne : la valeur de l'option demandée, ou "" si non trouvée
`public void` **`setNetworkProperty`**`(String name, String value)` Définit l'option du réseau sous-jacent (PDP) indiquée. Paramètres : name nom de l'option à définir value la valeur de l'option à définir
`public void` **`addContact`**`(String id)` Demande l'ajout du contact d'id donnée. Paramètres : id identificateur du contact à ajouter
`public void` **`addContact`**`(pdpPeerSimple p)` Demande l'ajout du peer donné comme contact. Paramètres : p le peer à ajouter
`public void` **`removeContact`**`(contact c)` Demande la suppression du contact donné. Paramètres : c le contact à supprimer
`public boolean` **`getNextEvent`**`(String cat, contact c, plugin p)` Demande de transmettre au plugin donné le prochain événement de la catégorie spécifiée destiné au contact donné. Permet à un plugin activé sur un contact X de demander les éventuels autres événements de même type qui sont également destinés à ce contact X. Paramètres : cat la catégorie de l'événement c le contact concerné

p le plugin à qui remettre l'événement Retourne : 'true' s'il y avait bien un événement à remettre (et appel le plugin concerné)

5.4.3.2. yb.p2pim.contact

Cette classe représente un contact de PIM. Accessible en lecture seule, sauf en ce qui concerne les propriétés (options) qui sont utilisables en écriture, ce qui permet par exemple aux plugins d'enregistrer des informations supplémentaires à propos d'un contact.

Tableau 10. API de yb.p2pim.contact

`static int` **`OFFLINE`**`, static int` **`ONLINE`**`,` `static int` **`AWAY`**`, static int` **`BUSY`** Modes de présence (0, 1, 2, 3).
`public String` **`getProperty`**`(String key)` Retourne la valeur de la propriété demandée. Paramètres : key le nom de la propriété demandée Retourne : la valeur de la propriété demandée, ou "" si introuvable Remarque : la seule propriété utilisée par PIM est "`displayName`", qui contient le nom sous lequel afficher un contact dans la liste des contacts
`public void` **`setProperty`**`(String key, String value)` Définit une propriété. Paramètres : key le nom de la propriété à définir value la valeur de la propriété à définir
`public pdpPeerSimple` **`getPeer`**`()` Retourne le peer représenté par ce contact. Retourne : le peer représenté par ce contact
`public int` **`getStatus`**`()` Retourne le statut de ce contact. Retourne : le statut de ce contact

5.4.3.3. yb.p2pim.plugin

Interface que les plugins pour PIM doivent implémenter. Les méthodes de cette interface sont appelées par PIM pour initialiser le plugin et connaître son mode de fonctionnement, puis par la suite pour lui signaler tout type d'événements (réception de message, confirmation d'envoi, activation par l'utilisateur, etc).

Tableau 11. Interface yb.p2pim.plugin

`static int` **`EVENTS_USER`** `static int` **`EVENTS_DIRECT`** `static int` **`EVENTS_BOTH`** Modes de gestion des événements entrants : - user : l'utilisateur doit 'cliquer' l'événement avant que le plugin ne le reçoive - direct : le plugin reçoit l'événement sans que l'utilisateur en soit informé au préalable - both : le plugin reçoit un avertissement comme quoi un événement est disponible, mais l'événement ne lui est pas transmis. L'événement est également signalé à l'utilisateur. L'événement sera consommé et transmis au plugin lorsque l'utilisateur le 'cliquera' ou lorsque le plugin en fera la demande au travers de la méthode `yb.p2pim.pim.getNextEvent()` (ce qui arrivera en premier).
`static int` **`TYPE_CONTACT`** `static int` **`TYPE_SYSTEM`** `static int` **`TYPE_BOTH`** Modes d'utilisation du plugin : - contact : le plugin est accessible depuis le menu contextuel de la liste des contacts - système : le plugin est accessible depuis le menu des plugins - both : le plugin est accessible depuis la liste des contacts et depuis le menu des plugins Quel que soit le mode de fonctionnement du plugin, le plugin sera accessible dans le menu de configuration des plugins.
`public boolean` **`init`**`(pim p, String path)` Initialisation (typiquement, c'est à ce moment que le plugin enregistre ses abonnements auprès de PIM). Paramètres : p objet parent path chemin d'accès jusqu'au répertoire où se trouve le plugin Retourne : 'true' si le plugin est prêt (utilisable)
`public void` **`end`**`()` Signale au plugin que l'application se termine.
`public void` **`actionConfigure`**`()` Demande l'affichage de l'écran de configuration de ce plugin.

`public void` **`action`**`()`

Evénement local système (l'utilisateur a activé ce plugin seul, sans contact associé).

`public void` **`action`**`(contact c)`

Evénement local sur un contact (l'utilisateur a activé ce plugin sur un contact).
Paramètres :

c	objet contact sur lequel appliquer l'action du plugin

`public void` **`action`**`(contact c, String cat, String msg)`

Evénement distant (PIM a reçu un message destiné à ce plugin).
Paramètres :

c	le contact source de cet événement
cat	la catégorie du message reçu
msg	le message reçu
user	indique si c'est l'utilisateur qui a activé cet événement

`public void` **`alert`**`(contact c, String cat)`

Signale l'arrivée d'un événement distant (cas des plugins inscrits en mode EVENTS_BOTH).
Paramètres :

c	le contact source de cet événement
cat	la catégorie du message reçu

`public void` **`result`**`(int id, boolean ok, String msg)`

Code de retour (accusé de réception) suite à l'envoi d'un message par ce plugin.
Paramètres :

id	identifiant associé à l'envoi du message d'origine
ok	indique si le message a pu être envoyé ou non
msg	éventuel message d'erreur

`public int` **`getType`**`()`

Le plugin doit indiquer son type (`plugin.TYPE_xxx`).
Retourne :
le type du plugin

`public ImageIcon` **`getIcon`**`()`

Le plugin doit retourner l'icône à utiliser pour signaler un événement lui étant associé.
Remarque :
peut retourner '`null`', auquel cas une icône par défaut est utilisée
Retourne :
l'icône associée à ce plugin

`public AudioClip` **`getSound`**`()`

Le plugin doit retourner le son à utiliser pour signaler un événement lui étant associé. Remarque : peut retourner '`null`', auquel cas un son par défaut est utilisé Retourne : le son associé à ce plugin
`public java.lang.String` **`getNameContact`**`()` Le plugin doit retourner son nom (affiché dans le menu contextuel des contacts). Retourne : le nom à afficher
`public java.lang.String` **`getNameSystem`**`()` Le plugin doit retourner son nom (affiché dans le menu des plugins) . Retourne : le nom à afficher
`public java.lang.String` **`getNameConfigure`**`()` Le plugin doit retourner son nom (affiché dans le menu de configuration des plugins). Retourne : le nom à afficher

5.4.4. Configuration

Le comportement de PIM est paramétré par le fichier `pimDefault.cfg` qui se trouve dans `yb/p2pim/` et par les options de l'utilisateur. Les fichiers de configuration utilisateur de PIM sont enregistrés dans $home/.pim/ et ses sous-répertoires (ou $home désigne le répertoire de base du compte d'utilisateur).

Le répertoire $home/.pim/ contient un fichier `lastid.pim` qui indique la dernière identité utilisée par l'utilisateur, ainsi qu'un répertoire par identité. Ces répertoires contiennent :

- un fichier `contacts.pim` qui contient la liste des contacts.
- un fichier `events.pim` qui contient les événements en attente (reçus mais non encore "vus" par l'utilisateur).
- un fichier `config.pim` qui permet de modifier la configuration de base de PIM, fichier dans lequel les plugins peuvent également enregistrer des données.

Les options chargées et reconnues à partir de `pimDefault.cfg` ou `config.pim` sont les suivantes (les quatre dernières options sont utilisées par `gPim` et non pas par la classe `pim` elle-même, ces options n'existent pas dans la configuration par défaut et sont créées à l'exécution):

Tableau 12. Options de configuration de PIM

Option	Description	Valeur par défaut
SOUND	activer ou non les sons	true
ACCEPT-UNKNOWN	accepter ou non les événements provenant de contacts hors liste	true
DEBUG	0 = rien 1 = affichage sur stdout 2 = fichier 3 = fichier + stdout	0 2 et 3 non implémentés
sizeX	taille en X de la fenêtre PIM	-
sizeY	taille en Y de la fenêtre PIM	-
posX	position X de la fenêtre PIM	-
posY	position Y de la fenêtre PIM	-

Remarque : PIM charge et met à disposition des plugins toutes les options trouvées dans ces fichiers, même si ce ne sont pas des options directement reconnues par PIM. Le plugin "PIM Messenger" présenté ci-après enregistre et récupère par exemple ses options au travers des méthodes `setProperty()` et `getProperty()` de PIM.

5.5. Plugins pour PIM

Les plugins permettent d'étendre le fonctionnement de base de PIM. Ce sont les plugins qui offrent les réelles fonctionnalités intéressantes pour l'utilisateur final.

5.5.1. Chargement

Le chargement des plugins par PIM se déroule comme suit :

- PIM ouvre le répertoire `"../../plugins"` (répertoire `plugins` au même niveau que le répertoire `yb` de base des paquetages). Remarque : si les paquetages sont regroupés dans un fichier `.jar` il faut indiquer "Class-Path: ./" dans le fichier manifeste (sans oublier le retour à la ligne !) afin que PIM puisse sortir du `.jar` et trouver le dossier `plugins`.
- PIM parcourt les sous-répertoires de `plugins` (chaque plugin doit se trouver dans un sous-répertoire séparé, de nom quelconque) et tente de charger un fichier `plugin.pim` dans chacun de ces répertoires. Ce fichier doit contenir le nom de la classe à charger pour construire le plugin.
- PIM construit un objet plugin en chargeant la classe indiquée.
- PIM initialise le plugin en appelant sa méthode `init()`.

5.5.2. Modes de fonctionnement

Un plugin doit indiquer deux choses quant à la manière dont il désire fonctionner :

- son mode de gestion des événements entrants
- la façon dont il désire être utilisable par l'utilisateur

5.5.2.1. Gestion des événements entrants

Lorsqu'un plugin est initialisé, il peut s'abonner auprès de PIM pour les catégories de messages qui l'intéresse. Pour chaque abonnement, il est possible de choisir le mode de remise des messages parmi trois options :

- `plugin.EVENTS_USER` : l'événement est signalé dans la liste des contacts, l'utilisateur doit 'cliquer' l'événement avant que le plugin ne le reçoive.
- `plugin.EVENTS_DIRECT` : le plugin reçoit l'événement directement, sans que l'utilisateur en soit informé au préalable.
- `plugin.EVENTS_BOTH` : le plugin reçoit un avertissement comme quoi un événement est disponible, mais l'événement ne lui est pas transmis. L'événement est également signalé à l'utilisateur. Il sera consommé et transmis au plugin lorsque l'utilisateur le 'cliquera' ou lorsque le plugin en fera la demande au travers de la méthode `yb.p2pim.pim.getNextEvent()` (ce qui arrivera en premier).

5.5.2.2. Utilisation du plugin

Lorsque l'interface est construite, chaque plugin doit indiquer comment il désire être accessible à l'utilisateur (appel de `plugin.getType()` par `gPim`). Les options reconnues sont les suivantes:

- `plugin.TYPE_CONTACT` : le plugin est accessible depuis le menu contextuel de la liste des contacts.
- `plugin.TYPE_SYSTEM` : le plugin est accessible depuis le menu des plugins.
- `plugin.TYPE_BOTH` : le plugin est accessible depuis la liste des contacts et depuis le menu des plugins.

A noter que quel que soit le mode de fonctionnement du plugin, ce dernier sera accessible dans le sous-menu `Configuration` du menu des plugins.

5.5.3. Accès au réseau

Un plugin a accès à toutes les méthodes publiques de la classe `yb.p2pim.pim` telles que présentées plus haut. Cela signifie en particulier qu'un plugin n'as PAS accès directement au réseau représenté par PDP, cependant les informations utiles pour contacter un peer sont disponibles (adresse IP notamment).

Un plugin a donc deux possibilités pour communiquer avec un autre peer :

- passer par l'intermédiaire de PIM
    - avantages :
        - utilisation très simple
        - gestion des relais
        - accusés de réception
    - inconvénients :
        - TCP + couche PDP/PIM, peut ne pas être assez rapide

- établir une communication directe avec son correspondant en utilisant un protocole propre au plugin

    - avantages :
        - meilleur contrôle de la transmission

    - inconvénients :
        - pas de gestion des relais
        - plus compliqué à mettre en place

Le choix dépendra donc du type de communication désirée. Un plugin envoyant relativement peu de messages aura tout intérêt à passer par PIM, tandis qu'un plugin désirant transmettre beaucoup de données rapidement (vidéo, fichiers) pourrait avoir intérêt à utiliser son propre mécanisme de communication.

5.5.4. Configuration

Les plugins peuvent se comporter comme de petites applications indépendantes de PIM, ou alors ils peuvent profiter des possibilités offertes par PIM. Les plugins peuvent enregistrer des valeurs à trois endroits :

- dans le fichier de configuration du réseau PDP, par l'intermédiaire de la méthode `setNetworkProperty(key, value)` de PIM.

- dans le fichier de configuration de PIM, par la méthode setProperty(key, value).

- directement dans les propriétés additionnelles d'un contact, par la méthode `setProperty(key, value)` de la classe `contact`.

5.5.5. Exemple

Le plugin de messagerie instantanée "PIM Messenger" est rapidement présenté ici. Ce plugin se trouve dans le répertoire `plugins/messenger/` qui contient un fichier `plugin.pim` indiquant la classe `"plugins.messenger.messenger"` comme classe à charger.

Ce plugin implémente un service de messagerie instantanée, qui se présente sous la forme illustrée ci-dessous.

Le plugin fonctionne en mode `TYPE_CONTACT`, ce qui signifie qu'il est accessible dans le menu contextuel de la liste des contacts, et en mode `EVENTS_BOTH`, ce qui signifie que les événements lui sont signalés immédiatement, mais transmis uniquement lorsque l'utilisateur les sélectionne ou lorsque le plugin le demande.

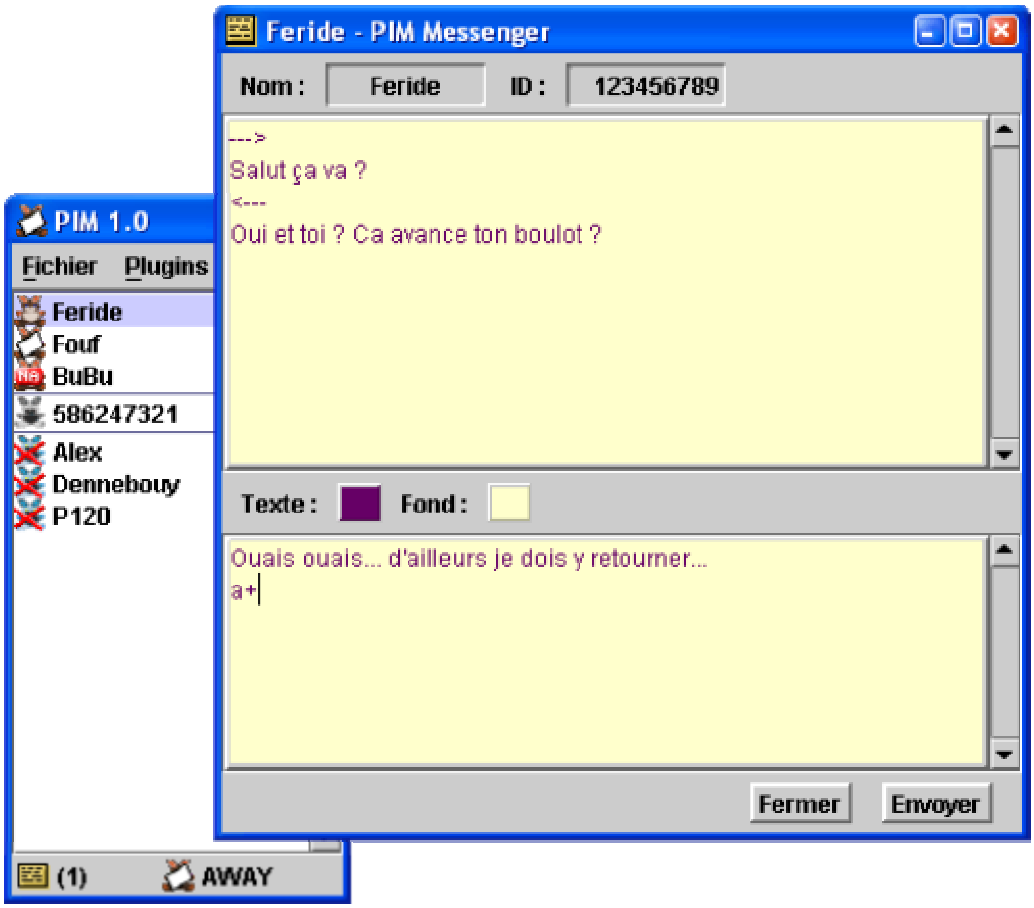


Figure 28. Plugin de messagerie instantanée pour PIM

Le plugin autorise une seule fenêtre de discussion par contact. Si une fenêtre de discussion existe déjà avec un contact donné, elle est réutilisée pour l'envoi de messages vers ce même contact (si la fenêtre est réduite ou masquée elle est tout d'abord re-affichée).

Le mode de fonctionnement `EVENTS_BOTH` permet au plugin de trier les événements entrants (messages reçus) :

- les événements en provenance d'un contact pour lequel une fenêtre de discussion est déjà ouverte sont immédiatement consommés et affichés par le plugin (s'il existe une fenêtre, c'est que l'utilisateur est en pleine discussion ; il est donc logique de lui afficher les messages sans qu'il doive les 'cliquer').
- les autres événements ne sont pas consommés (ils sont signalés à l'utilisateur par PIM et seront consommés lorsque l'utilisateur les sélectionnera).

Pour ce qui est de sa configuration (couleurs choisies par l'utilisateur), ce plugin fait usage des méthodes de PIM pour enregistrer ses options dans le fichier de configuration de PIM.

Tableau 13. Options de configuration du plugin PIM Messenger

Option	Description	Valeur par défaut
messenger-fg	Couleur du texte	-
messenger-bg	Couleur de fond	-

Tableau 14. Protocole du plugin PIM Messenger

Catégorie	Message	Remarque
pim-messenger	Contenu du message	Un message

5.6. Perspectives

La version actuelle de PDP/PIM est tout à fait utilisable, mais du travail pourrait encore être réalisé sur ce projet.

5.6.1. PDP

Le module PDP est le plus compliqué, et c'est aussi celui qui pourrait encore demander le plus de travail.

5.6.1.1. Format des messages

J'avais prévu d'utiliser un protocole de messages faisant usage de XML, mais le temps a finalement manqué pour remplacer le format texte simple de départ. L'utilisation de XML pourrait apporter une plus grande flexibilité et une plus grande facilité dans l'analyse des messages.

Un message d'envoi d'une liste de peers pourrait se présenter comme suit :

```
<pdp version="4.0">
    <type>exploration</type>
    <sous-type>envois-liste</sous-type>
    <id>232</id>
    <dest-id>45547653242</dest-id>
    <donnees>
        <peer>
            <id>67676819</id>
            <ip>212.43.2.73</ip
            <port>4043</port>
            <peer>  <!-- le relais -->
                <id>12305671</id>
                <ip>11.5.3.56</ip
                <port>4043</port>
            </peer>
        </peer>
        ...
    </donnees>
</pdp>
```

5.6.1.2. Réglage des bonus / malus

Il serait utile d'analyser et tester plus à fond le fonctionnement du système de bonus/malus attribués aux peers, afin d'obtenir un système qui tende vers un état stable (un nombre de peers constant).

5.6.1.3. Problèmes d'accès au réseau

Les peers "cachés" derrière des firewalls ou dans des réseaux NAT posent des problèmes : il faudrait investir plus de temps dans la recherche et le développement d'une solution à ces problèmes spécifiques.

La découverte automatique de peers non joignables par multicast pose aussi quelques problèmes qu'il faudrait pouvoir étudier plus à fond.

5.6.1.4. Structure du réseau

Idéalement, il faudrait que le réseau soit totalement auto-organisationnel, c'est-à-dire que les peers s'arrangent automatiquement entre eux pour que la connectivité soit optimale.

Actuellement, la structure du réseau est seulement semi auto-organisationnelle, dans le sens où les peers peuvent se transformer en "serveur relais" ou "client relais" (auto-organisation d'une hiérarchie), mais ils ne cherchent pas à structurer leurs interconnexions.

5.6.1.5. Cryptage

J'avais dès le départ prévu un petit module (ou sous-module de PDP) dédié au cryptage, ce qui explique la présence un peu partout dans PDP et dans PIM d'identifiants publiques et d'identifiants privés.

Suite à un léger glissement du planning (dû entre autre au système de plugins, et au fait que plus de temps a été alloué au rapport), le module de cryptage n'a pas pu être réalisé. Cependant, pour le lecteur intéressé, le code source mentionne les endroits où j'avais prévu d'utiliser ce module.

Il était prévu d'utiliser le cryptage principalement dans deux buts :

- sécuriser la phase d'identification
- sécuriser les messages transmis

Identification

L'identification pourrait être renforcée par l'utilisation d'un protocole tel que celui-ci (appliqué juste après qu'une connexion ait été établie entre A et B, et juste avant que la transmission ne débute) :

- A annonce son ID auprès de B.
- B crypte une chaîne aléatoire à l'aide de la clé publique de A (son ID justement) et transmet le résultat à A.
- A décrypte cette chaîne et la renvoie à B.

Avec ce petit protocole, B est assuré que le peer s'annonçant en tant que A est effectivement A (puisqu'il est en possession de la clé privée de A ayant permis de décrypter la chaîne aléatoire).

Messages

Les messages les plus intéressants à crypter seraient ceux du type `"message"` du protocole PDP. En effet, ce sont les seuls à transporter une réelle information potentiellement confidentielle. Le cryptage pourrait fonctionner comme décrit ci-dessous (modèle à clés privées / clés publiques où la clé publique est l'identifiant du peer) :

- lors d'un envoi de A vers B :
  - A signe le message à l'aide de sa clé privée (A crypte le hash du message, voir le message entier).
  - A crypte le message signé à l'aide de la clé publique de B.
- lorsque B reçoit le message de A :
  - B décrypte le message à l'aide de sa clé privée.
  - B vérifie la signature de A en décryptant cette dernière avec la clé publique de A (comparaison du hash réalisé sur le message par B, avec le hash décrypté provenant de A).

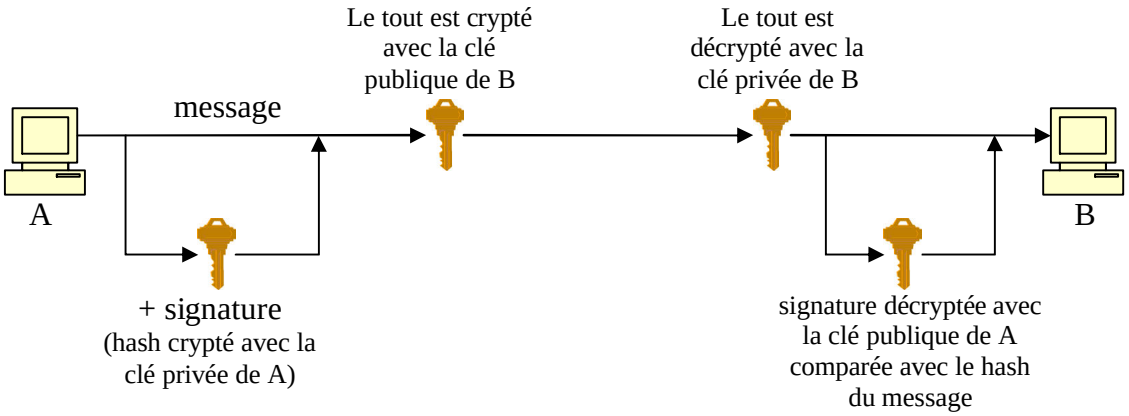


Figure 29. Schéma des encryptions

En appliquant ce protocole, la sécurité (authenticité, confidentialité et intégrité) des échanges de messages serait garantie.

5.6.2. PIM

Le module PIM ayant provoqué moins de problèmes imprévus, il n'y a pas de réel défaut à corriger, mais on pourrait y apporter quelques modifications ou ajouts, comme par exemple offrir la possibilité d'utiliser un mot de passe local pour protéger l'accès à une identité.

5.6.3. Plugins

La partie plugin serait la partie la plus intéressante à développer, puisque c'est cette partie qui permet de réaliser des services utilisant les possibilités de PDP/PIM. Il serait par exemple envisageable de développer des plugins tels que ceux listés ci-dessous :

- un chat (discussion à plus de 2 utilisateurs simultanés)
- un plugin de partage de fichiers
- un système de tableau noir virtuel
- un agent mobile (par exemple pour 'indexer' les peers participants)

6. Conclusion

Le module PDP, responsable de la mise en place du réseau de peers et du mécanisme de présence, atteint ses objectifs, mais il reste de la place pour des améliorations (comme exposé au chapitre 'Perspectives'). Ce sujet pourrait constituer le thème d'un projet ou d'un diplôme à part entière.

L'application PIM réalisée, responsable de la messagerie instantanée, atteint quant à elle tout à fait ses objectifs et les dépasse même. En effet, PIM ne se contente pas d'être une application de messagerie instantanée, mais offre un support à toutes sortes de fonctionnalités peer-to-peer au travers de son système d'extension par plugins.

L'objectif de portabilité enfin est bien atteint, le développement ayant été entièrement réalisé en Java.

La figure présentée ci-dessous illustre mon "taux de satisfaction" quant à la 'réussite' des quatre objectifs :

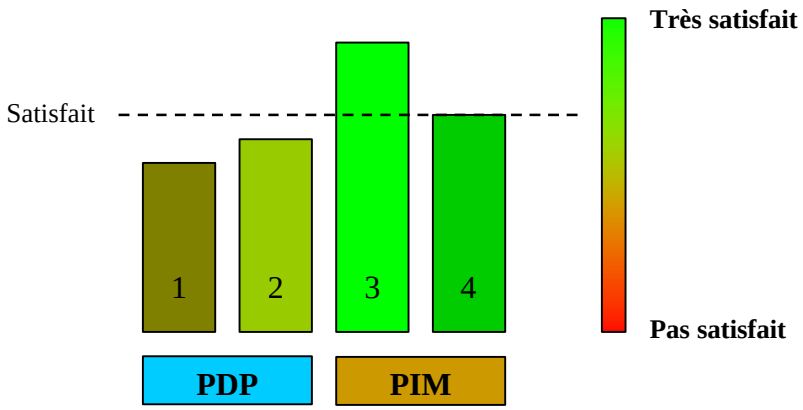


Figure 30. Degré de satisfaction

1. Définir et mettre en place un réseau peer-to-peer.
2. Définir un mécanisme de présence sans serveur s'appuyant sur ce réseau.
3. Développer un système de messagerie instantanée peer-to-peer.
4. Intégrer ces points au sein d'une même application, en privilégiant la portabilité (compatibilité inter-plateforme).

Comme le montre la figure, les objectifs ont globalement été atteints. Le système dans son état actuel convient bien à une utilisation en réseau local, mais nécessiterait encore quelques améliorations pour une utilisation élargie (Internet) efficace.

7. Liste des symboles et abréviations

AIM	"AOL Instant Messaging". Un logiciel de messagerie instantanée.
API	"Application Programmation Interface", ensemble des éléments d'un objet utilisables par un autre programme, documentation de ces éléments.
FIFO	"First In First Out", premier entré premier sorti, mode de gestion de queue (liste, file d'attente).
ICQ	"I Seek You", "je te cherche". Un logiciel de messagerie instantanée.
ID	"Identity", identité, identifiant.
IDE	"Integrated Development Environment", environnement de développement.
IM	"Instant Messaging", messagerie instantanée.
IP	"Internet Protocol", protocole pour Internet, adresses utilisées par ce protocole.
ms	Millisecondes.
NAT	"Network Address Translation", traduction d'adresse. Procédé permettant l'accès au réseau à un nombre d'hôtes plus important que le nombre d'adresses réellement disponibles.
P2P	"Peer to Peer".
PAT	"Port Address Translation", traduction d'adresses basée sur les numéros de port. Procédé permettant l'accès au réseau à un nombre d'hôtes plus important que le nombre d'adresses réellement disponibles.
PDP	"Peers Discovery Protocol" ou "Protocole de Découverte de Peers", module de base de mon développement.
PIM	"Peers Information Manager". Signifiait au départ "Peers Instant Messaging", version contractée de "Peer to Peer Instant Messaging" (p2pim). Nom de mon application développée au-dessus du module PDP.
SMS	"Short Message Service", service d'envoi de messages sur téléphones portables.
TCP	"Transmission Control Protocol", protocole orienté connexion, sûr, qui permet le contrôle de flux et de congestion.
TTL	"Time To Live", durée de vie, compteur de distance de transmission
UDP	"User Datagram Protocol", protocole orienté 'sans connexion', sans contrôle.
UML	"Unified Modeling Language", langage de modélisation (spécification, visualisation, documentation).
VS	"Versus", contre, opposé à.
XML	"Extensible Markup Language", langage de représentation de documents.

Firewall	Pare-feu. Dispositif destiné à sécuriser un réseau en bloquant / contrôlant les connexions.
Hash	Hachage. Chaîne réduite obtenue à partir d'une chaîne originale, par un algorithme de hachage.
Peer	Membre du réseau, ordinateur de ce réseau.

8. Liste des figures

9. Liste des tableaux

10. Bibliographie

10.1. Peer-to-peer

- Dépliants EPFL portes ouvertes 2003
- Instant Messaging and Presence Protocol (impp) Charter (http://www.ietf.org/html.charters/impp-charter.html)
- IMPP Information HomePage (http://www.imppwg.org)

10.2. Systèmes de messagerie instantanée

- ICQ (http://www.icq.com)
- AIM (http://www.aim.aol.com)
- Yahoo! Messenger (http://messenger.yahoo.com)
- MSN Messenger (http://messenger.microsoft.com)
- Protocols – gaim (http://gaim.sourceforge.net/protocol.php)
- The ICQ hacking page (http://www.algonet.se/~henisak/icq/)
- FAIM-AIM-Oscar Protocol (http://aimdoc.sourceforge.net/faim/protocol/)
- libyahoo2 - A C library for Yahoo! Messenger (http://libyahoo2.sourceforge.net)
- MSN Messenger protocol (http://www.venkydude.com/articles/msn.htm)
- Jabber, Inc. (http://www.jabber.com)
- Jabber Software Foundation (http://www.jabber.org)
- Bantu Instant Messaging and Presence Solutions for Enterprises (http://corp.bantu.com)
- Lotus Sametime (http://www.lotus.com/products/lotussametime.nsf/wdocs/homepage)
- Trillian (http://www.ceruleanstudios.com/trillian)
- EveryBuddy! (http://www.everybuddy.com/en/index.php)
- PIM (http://change.to/pim)

10.3. Logiciels d'échange de fichiers

- Gnutella (http://www.gnutella.com)
- Kazaa (http://www.kazaa.com)
- eDonkey2000 (http://www.edonkey2000.com)
- eMule (http://www.emule-project.net/)
- BitTorrent (http://bitconjurer.org/BitTorrent/)
- peer-to-peer networks (http://www.goldenpi.no-ip.org/drm/p2p.shtml)
- Dépliants EPFL portes ouvertes 2003

10.4. Développement

- Documentation des APIs Java 1.4.2 (http://java.sun.com/j2se/1.4.2/docs/api/)
- The Java Tutorial (http://java.sun.com/docs/books/tutorial/)

- Java Tech Tips (http://developer.java.sun.com/developer/TechTips/index.html)
- NetBeans IDE (http://www.netbeans.org/)
- Poseidon for UML (http://www.gentleware.com)
- Inno Setup (http://www.innosetup.com)
- NSIS : Nullsoft Scriptable Install System (http://nsis.sourceforge.net/site/index.php)

10.5. Autres

- Project JXTA (http://www.jxta.org)
- P-Grid (http://www.p-grid.org)
- PIM (http://change.to/pim)

11. Annexe A : Planning

11.1. Calendrier

Les éléments en *italique* sont des remarques par rapport au déroulement prévu.

Semaine 1 29.09.2003 – 03.10.2003	- planning global - installation - plan de découpage en modules *JBuilder a été abandonné au profit de NetBeans IDE*	démarrage du projet
Semaine 2 06.10.2003 - 10.10.2003	- spécifications du module PDP *spécifications passablement modifiées en cours de route, pas idéal*	Module PDP (réseau de peers) *4 semaines*
Semaine 3 13.10.2003 - 17.10.2003	- communications en JAVA - PC connu à PC connu - PC → broadcast / multicast - PC → broadcast / multicast vers un autre réseau *les programmes de test réalisés se trouvent sur le CD-ROM annexé, les résultats sont présentés en annexe. Recherche d'Info sur JXTA : sympa, malheureusement il faut configurer manuellement un « rendez-vous peer » et un « relay-peer » et la liste de ces peers est centralisée sur le serveur du site JXTA.*	
Semaine 4 20.10.2003 - 24.10.2003	- implémentation de PDP *version 1, puis 2, du protocole PDP* *1ers tests de fonctionnement de PDP*	
Semaine 5 27.10.2003 - 31.10.2003	- implémentation de PDP, tests *ajout tardif du système de relais, donc passage à la version 3 du protocole PDP.* *correction de bugs*	
Semaine 6 03.11.2003 - 07.11.2003	- spécifications du module PIM *ajout des fichiers de configuration externes à PDP avec utilisation d'objet 'Properties' (lu ceci par hasard dans la doc Java).* *PIM est mieux planifié que PDP, et donc plus "agréable" à implémenter ensuite*	Module PIM (utilisation des peers) *4 semaines*
Semaine 7	- interface	

10.11.2003 - 14.11.2003	*ajout des accusés de réception au protocole de PDP*	
Semaine 8 17.11.2003 - 21.11.2003	- gestion liste des contacts - utilisation du réseau *corrigé un gros problème d'interblocage dans PDP, malgré tout l'utilisation UC affiche un déprimant 99% ☹.* *trouvé et corrigé le problème de sommeil (wait) de PDP, UC baissée à < 5% ☺*	
Semaine 9 24.11.2003 - 28.11.2003	- gestion messages / événements - plugin de messagerie *25.11.2003 : crash de ma machine juste après mise à jour par WindowsUpdate (plus d'accès réseau, démarrage extrêmement lent), comme quoi : "mise à jour" != "amélioration"*	
~~Semaine 10~~	~~- réalisation du module de cryptage~~	~~Cryptage / Sécurité~~
Semaine 10 01.12.2003 - 05.12.2003	- structuration du rapport - rédaction *finitions sur le plugin PIM Messenger*	Rapport ~~*2 semaines*~~ *~3 semaines*
Semaine 11 08.12.2003 - 12.12.2003	- rendu rapport préliminaire (début de semaine) - rédaction du rapport final *rendu poster de présentation. développement du site web de PIM (qui n'était pas prévu).*	
Semaine 12 15.12.2003 - 18.12.2003	- correction, relecture et impression du rapport final - préparation et gravure des CDs - rendu des copies (jeudi 18.12.2003) *corrections de dernière minute…* *Bonnes Fêtes ☺*	

11.2. Remarques

Le module de cryptage était considéré comme moins important, car ce n'était pas le sujet principal de ce travail, c'est pourquoi il ne lui était alloué qu'une semaine en fin de projet. Il est évident que pour une utilisation « réelle », ce module devrait être considéré comme très important.

Suite à un léger glissement du planning (dû au système de plugins qui n'avait pas été prévu dans la planification globale initiale, et au fait que plus de temps a été alloué au rapport), le module de cryptage n'a pas pu être réalisé, mais le code source mentionne les endroits où j'avais prévu d'utiliser ce module.

12. Annexe B : Diagrammes de classe

12.1. yb.pdp (module PDP)

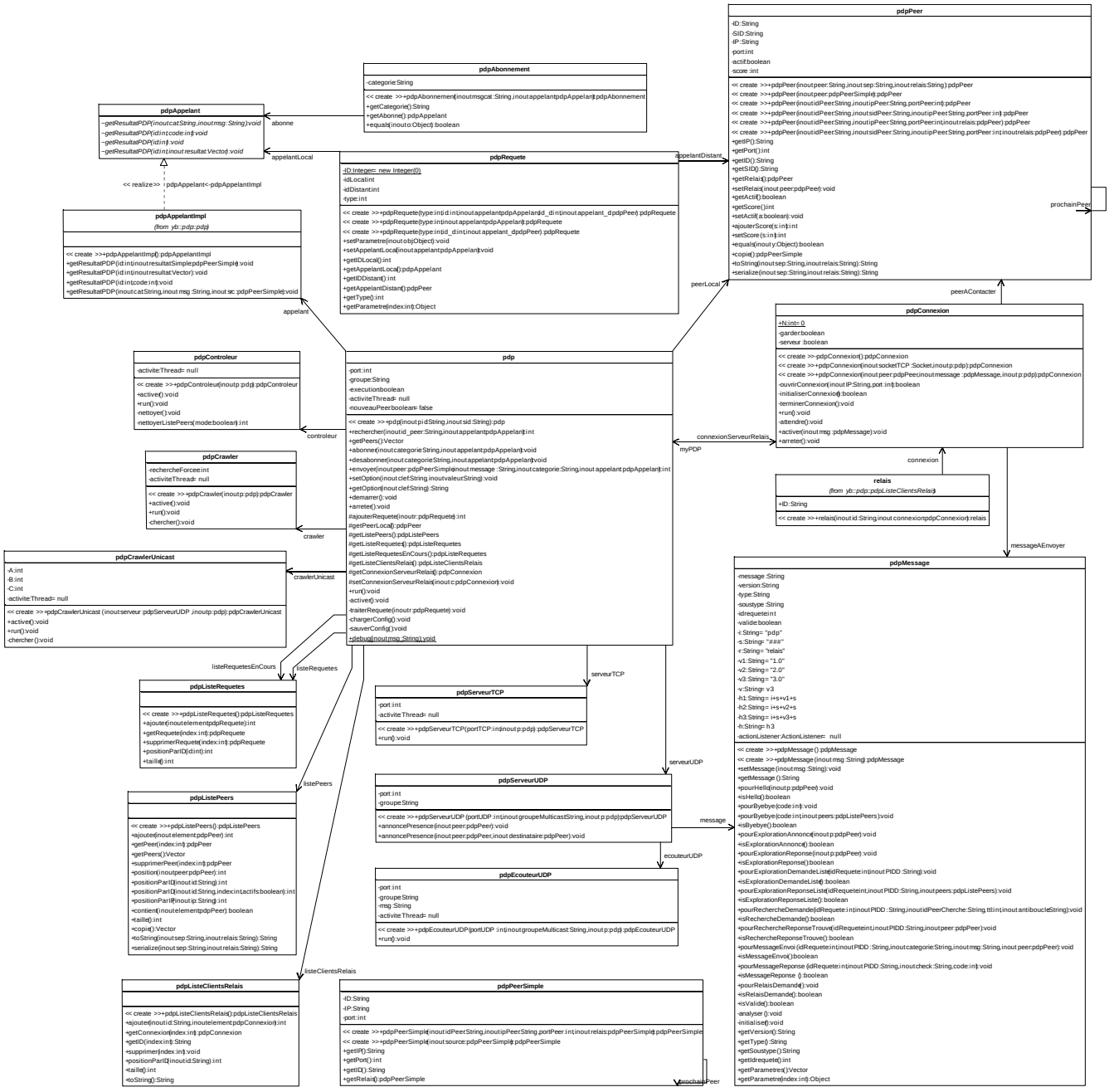


Figure 31. Diagramme des classes du paquetage yb.pdp (PDP)

12.2. yb.p2pim (module PIM)

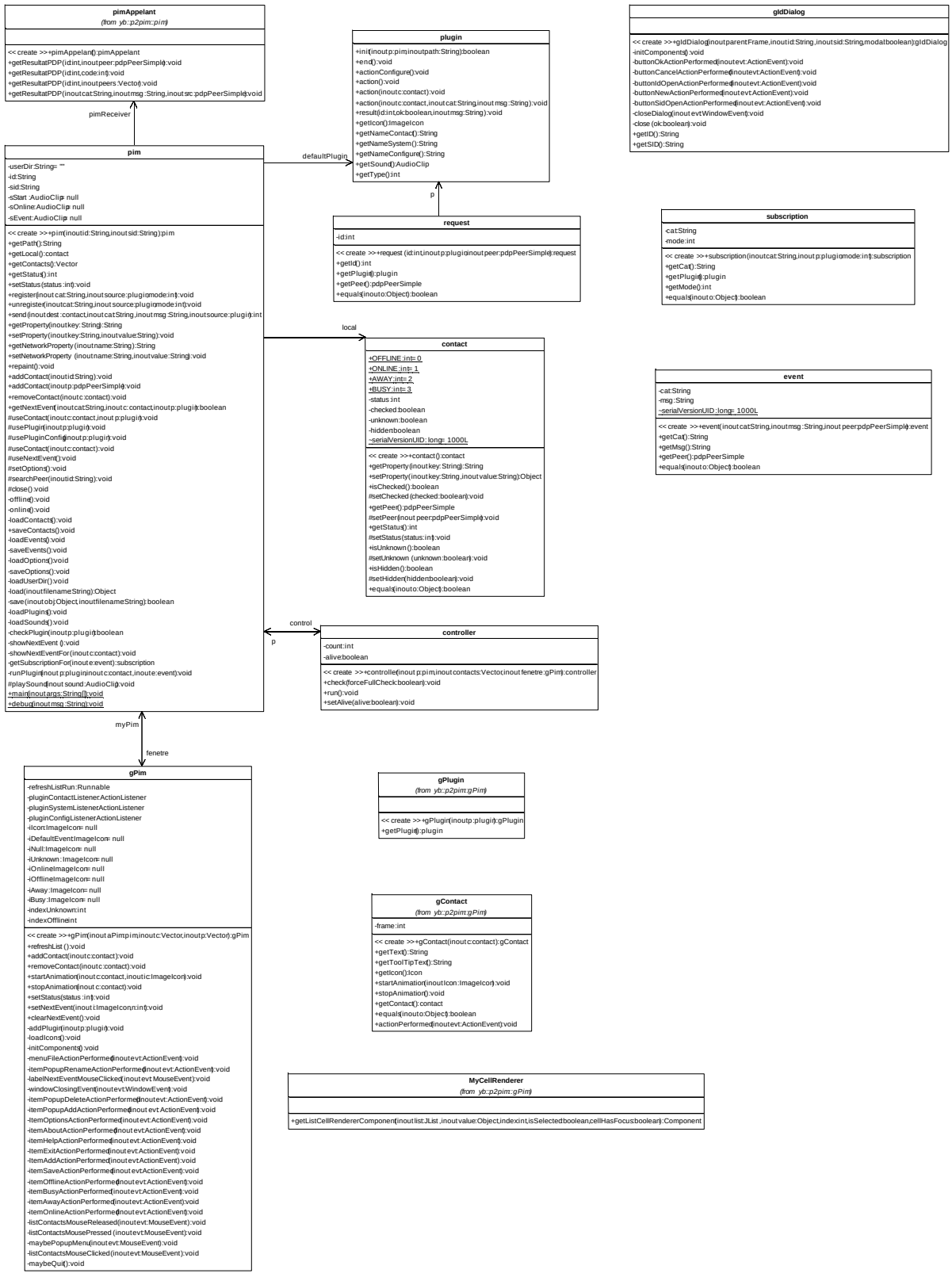


Figure 32. Diagramme des classes du paquetage yb.p2pim (PIM)

13. Annexe C : tests de communication

Cette annexe résume les tests de communication réalisés en début de développement. La configuration de test était la suivante : trois machines, A, B et C. A et B se trouvent dans le même réseau local (LAN) qui est protégé par un firewall, mais ce dernier a été désactivé pour l'occasion, il peut donc être ignoré. C se trouve dans un réseau séparé, protégé par un firewall (aucun port ouvert en entrée). Les machines A, B et C tournaient respectivement sous Windows 98, Windows XP et Linux.

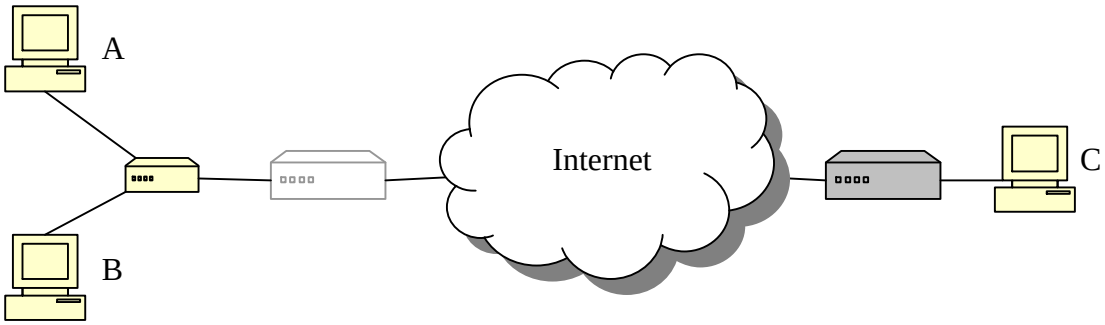


Figure 33. Tests de communication

1) TCP : unicast, c'est-à-dire connexion vers un hôte précis. TCP est orienté connexion, il y'a donc établissement d'une connexion avant l'échange de messages.

	envoi	réponse
A → B	✓	✓
B → A	✓	✓
A ou B → C	✗	✗
C → A ou B	✓	✓

2) UDP : unicast, orienté sans connexion (envoie de message sans garantie).

	envoi	réponse
A → B	✓	✓
B → A	✓	✓
A ou B → C	✗	✗
C → A ou B	✓	✗

3) Multicast : envoi vers un « groupe » (IPs 224.0.0.1 à 239.255.255.255) d'hôtes « inscrits » à ce groupe.

	envoi	réponse
A ou B → A et/ou B	✓	-
A ou B → C	✗	-
C → A ou B	✗	-

4) Broadcast : envoie à tout le monde / tous les hôtes d'un réseau (IP net-id + 1..1).

	envoi	réponse
A ou B → A et/ou B	✓	-
A ou B → C	✘	-
C → A ou B	✘	-

Multicast et Broadcast fonctionnent bien en LAN mais fonctionnent très mal (voir pas du tout) sur Internet. En effet, pour que cela fonctionne, il faudrait que tout les routeurs intermédiaires soient configurés pour accepter ce genre de trafic, ce qui est loin d'être le cas.

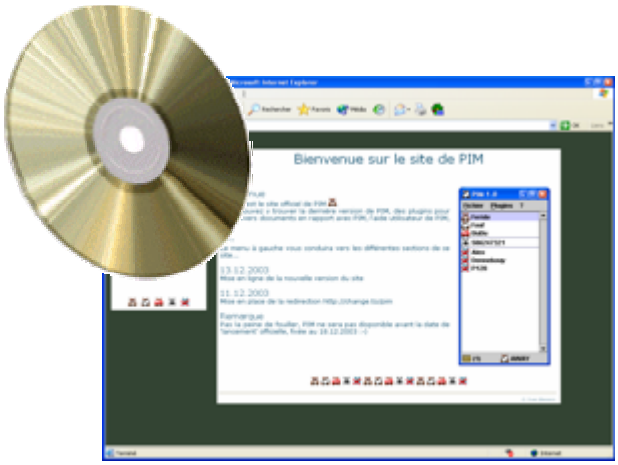
UDP à l'avantage sur TCP d'être plus simple et plus rapide (pas d'établissement de connexion) mais il a des désavantages ; manque de fiabilité, pas de contrôle de flux / congestion et surtout UDP ne permet pas, au contraire de TCP, de communiquer de manière bidirectionnelle avec un hôte derrière un firewall (TCP le permet si l'un des deux hôtes n'est pas bloqué. Exemple : C → A ou B).

Petit test de « scan » : envoi de messages UDP sur toutes les IP possibles, une par une. Résultat : il faut environ 30 minutes pour balayer toutes les IPs.

Petit test : le récepteur UDP multicast reçoit les paquets multicast mais AUSSI les paquets UDP unicast (pour autant que le port corresponde), le récepteur UDP unicast ne reçoit que les paquets unicast. Démarrer les deux récepteurs en parallèle est risqué, le résultat est aléatoire. Il est donc conseillé de n'utiliser que le récepteur multicast.

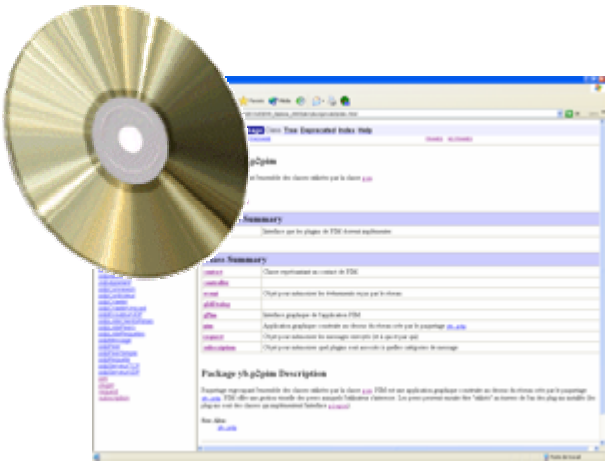
14. Annexe D : Documentation utilisateur

La documentation utilisateur à jour est disponible en ligne, sur le site Web de PIM, à l'adresse suivante : `http://change.to/pim`. Une copie se trouve également sur le CD-ROM annexé.



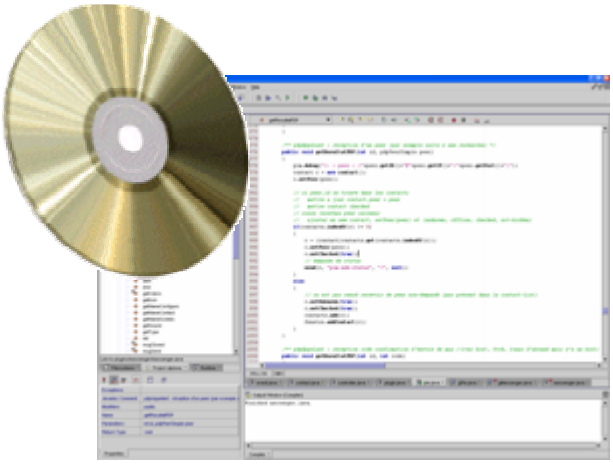
15. Annexe E : Documentation technique des API

La documentation de programmation complète, au format HTML Javadoc, se trouve sur le CD-ROM annexé. La version "publique" est également disponible sur le site de PIM à l'adresse `http://change.to/pim`.



16. Annexe F : Code source

Le code source se trouve sur le CD-ROM annexé.



17. Annexe G : CD-ROM

Le CD-ROM annexé contient ce document, le poster de présentation de PIM, les codes sources du projet, la version compilée du projet, les programmes de test utilisés en début de projet, la documentation Javadoc des API, la documentation utilisateur, l'environnement d'exécution JAVA, etc.