JXTA : Un Framework Peer-to-Peer Open Source

Quentin Dallons
qdallons@info.fundp.ac.be

Institut d'informatique FUNDP
Namur, Belgique

Résumé Les technologies Peer-to-Peer sont aujourd'hui de plus en plus utilisées car elles constituent une méthode efficace de partage de ressources entre membres d'une même communauté. Par ailleurs, en plus des utilisations grand public les plus connues (partage de fichiers et messagerie instantanée), ces technologies peuvent résoudre un ensemble de problèmes liés à l'utilisation optimale des ressources disponibles dans les réseaux et les entités constituant ceux-ci. Cependant, aujourd'hui, les implémentations d'architectures Peer-to-Peer sont non-interopérables et en général limitées à une plateforme bien déterminée. C'est dans l'optique de supprimer ces défauts que Sun Microsystem a proposé le framework Peer-to-Peer Open Source JXTA[1].

1 Introduction

Le vocable *Peer-to-Peer* (P2P) est aujourd'hui un des mots les plus à la mode. En effet, il n'est plus nécessaire de présenter Gnutella [1], eDonkey [2], Kazaa [3], Jabber [4], MSN Messenger [5], ICQ [6], SETI@Home [7], ... tellement ces systèmes sont utilisés et connus du grand public. Hormis les problèmes juridiques liés à l'utilisation de certaines de ces applications et à un certain effet de mode, ces méthodes d'utilisation de l'informatique distribuée apportent d'énormes avantages. En effet, l'utilisation des techniques *Peer-to-Peer* permet de résoudre un ensemble de problèmes fréquemment rencontrés de part l'explosion des contenus disponibles sur l'Internet et dans le cadre d'une utilisation optimale des ressources informatiques.

L'Internet se compose principalement d'informations, de bande passante, de capacité de calcul et de stockage. Ces diverses "matières premières" sont rarement exploitées optimalement. Aujourd'hui, aucun moteur de recherche ou portail d'information basé sur des méthodes de recherches traditionnelles[2] ne peut localiser ou cataloguer en temps réel la quantité, sans cesse croissante, d'informations présentes sur le Web. En effet, cette quantité étant colossale, il est impossible de tout répertorier a priori. De plus, beaucoup d'informations

[1] JXTA se prononce "juxta".

[2] Les moteurs de recherches actuels sont basés sur le principe d'une recherche en base de données. Cette base de données a été constituée a priori au moyen de Webcrawlers (robots) recherchant les meta-data des pages html.

présentes sur l'Internet sont de nature temporaire. Cette information *transient* apparaît à un moment donné, reste disponible pendant un laps de temps plus court que le temps nécessaire à son indexation par les moteurs de recherches traditionnels. Ensuite, cette information disparaît parfois même définitivement.

En utilisant les outils traditionnels d'indexation et de recherche, nous passons donc à côté d'une grande quantité d'informations.

Parallèlement, l'utilisation de la bande passante (qui double tous les 16 mois environ) disponible sur l'Internet n'est pas forcément utilisée de manière optimale car de nombreux utilisateurs se rendent souvent sur les mêmes sites (recherche, vente en ligne, information, ...). Ceci provoque des points de congestion inévitables et récurrents car liés aux habitudes des utilisateurs.

Le même effet de sous-exploitation des ressources potentielles de calcul et de stockage des différentes entités constituant l'Internet est lui aussi évident.

C'est pour ces raisons que divers projets P2P ont tenté de remédier à cette sous-exploitation des ressources disponibles sur l'Internet, en offrant des services de recherches distribuées temps réel (ex : Distributed Hash Table, Query Flooding, ...), des services de mise de partage temporaire d'information (ex : Gnutella, eDonkey, Kazaa, ...), des services de calculs distribués (ex : SETI@home, ...), des services de messagerie instantanée (ex : Jabber, MSN Messenger, ICQ, ...), ou des services de distribution du stockage de fichiers (CFS [8], PAST [9], OceanStore [10], Ivy [11], Freenet [12], ...). Ces services sont conçus de manière à utiliser au mieux les ressources disponibles sur l'Internet telles que la bande passante, le temps de calcul et l'espace de stockage.

Malgré leurs bonnes intentions, ces solutions pèchent par leur hétérogénéité et leur non-interopérabilité. Pour cette raison, Sun Microsystem a proposé un framework P2P pour tenter de résoudre ce problème d'hétérogénéité et de non-interopérabilité lié aux diverses implémentations P2P : le framework JXTA[3]. Au commencement, ce framework était fermé mais, par la suite, Sun a décidé de le rendre Open Source de manière à le rendre plus populaire auprès des développeurs et utilisateurs en vue d'en assurer le succès.

Cet article a pour objectif de présenter un "état de l'art" du framework JXTA et ses diverses composantes sur base des articles et documentations disponibles. Pour y parvenir, nous passerons tout d'abord en revue les objectifs essentiels du framework. Ensuite, nous nous attarderons sur l'architecture proposée par Sun ainsi que sur les concepts et différents protocoles entrant dans sa composition.

2 Les objectifs du framework JXTA

Cette section, basée sur les articles [13] et [14], a pour but de présenter les trois objectifs du framework JXTA :

- L'interopérabilité
- L'indépendance vis-à-vis des plateformes logicielles et matérielles
- L'ubiquité

[3] JXTA se prononce "juxta"

2.1 Interopérabilité

L'interopérabilité constitue le premier objectif du framework JXTA et vise essentiellement à simplifier l'interconnexion, la localisation et la communication entre les peers d'un système P2P. L'idée sous-jacente est de permettre à ces peers d'offrir de manière la plus transparente possible des services et de participer plus aisément aux activités proposées par les différentes communautés P2P. Ceci n'étant pas possible aujourd'hui car beaucoup de systèmes P2P ont été développés dans l'optique de ne fournir qu'un seul type de service. Par exemple, Napster se chargeait du partage de fichiers musicaux, Gnutella, de son côté, offre un partage générique de fichiers et les logiciels de messagerie instantanée fournissent souvent en plus de la messagerie proprement dite des services de téléconférence et de transfert de fichiers (Jabber, ICQ, MSN Messenger, ...). Etant donné les caractéristiques très variées de ces services et le manque d'une architecture sous-jacente unifiée, chaque concepteur de logiciel P2P crée son propre système ; système incompatible avec les autres (Par exemple, e-donkey est incompatible avec Gnutella et MSN Messenger avec ICQ). Cette non-interopérabilité des systèmes implique que chaque concepteur crée sa propre communauté d'utilisateurs et fournit certainement un effort de conception déjà réalisé par un autre. Cette explosion du nombre de communautés distinctes dont les systèmes sont "non-interopérables" a pour inconvénient de morceler l'ensemble des peers fournisseurs et consommateurs de services et leur impose de supporter chacune de leurs spécifications si ces peers désirent profiter du maximum de services offerts.

Cette situation ressemble à l'Internet avant l'apparition des browsers Web où un utilisateur était bloqué dans une communauté car pour avoir accès à l'information, il devait souscrire à un fournisseur d'accès particulier (Compuserve, AOL,...) qui était responsable de la technique de diffusion du contenu.

Pour conclure, on peut résumer cet objectif d'interopérabilité par la citation suivante issue de [13] : "Le projet JXTA a pour but d'offrir au monde du P2P ce que le browser a apporté à l'Internet."

2.2 Indépendance de la plateforme

Le second objectif du framework JXTA est d'être indépendant des langages de programmation (C, C++, Java, Ruby, Python, Perl, ...), des systèmes d'exploitation (MS Windows et Unix) et des technologies réseaux (TCP/IP, UDP, Bluetooth, WiFi, ...).

La motivation principale poussant à cette indépendance provient du fait que beaucoup de systèmes P2P offrent leurs services au moyen d'un ensemble d'API écrites dans un langage particulier, au-dessus d'un système d'exploitation déterminé et sur base d'un protocole réseau précis ; cette forte détermination en limite l'utilisation et la diffusion. Ainsi, on peut trouver des exemples de systèmes P2P offrant un "jeu" d'API C++ sous Windows au-dessus de TCP/IP, ou d'autres systèmes offrant des API C sous Unix au-dessus de TCP/IP et HTTP. Cette multitude d'API dépendantes de la plateforme logicielle ou matérielle ne facilite aucunement la tâche au développeur. Celui-ci, s'il veut offrir un même service

à deux communautés, devra implémenter son service deux fois en fonction des "jeux" d'API utilisés ou éventuellement créer un pont entre les deux plateformes. Reconnaissons-le, cette méthode est peu réaliste au vu du nombre de plateformes P2P existantes.

2.3 Ubiquité

Le dernier objectif est l'ubiquité. Il matérialise le désir que le framework JXTA soit implémentable sur n'importe quel périphérique doté d'un coeur digital ("digital heartbeat"[13]). Cette notion est très générale puisqu'elle comprend les senseurs, l'électronique grand public, les PDA, les GSM, les routeurs réseaux, les ordinateurs de bureau, les serveurs de données ou de traitements, ...

Cet objectif a été motivé par la constatation suivante : beaucoup de systèmes P2P récents ont choisi Microsoft Windows comme unique cible de déploiement. Parmi les raisons souvent invoquées pour justifier ce choix, on trouve l'argument que la plateforme "Wintel" constitue la base la plus largement installée. Cependant, ce choix, souvent basé sur une vision à court terme, a l'effet néfaste suivant : l'incorporation de caractéristiques propres à la plateforme "Wintel" dans la conception du système et une augmentation de la dépendance vis-à-vis de cette plateforme. De plus, les applications P2P actuelles sont souvent orientées grand public, ce qui constitue un choix plutôt restrictif par rapport aux possibilités offertes par la technologie P2P. Cette dernière peut s'adresser à l'ensemble des peers disponibles allant des senseurs à de gros systèmes d'entreprise.

3 Architecture

Cette section est basée sur les références suivantes : [15][13][16]

En Janvier 2000, suite à une étude portant sur un grand nombre d'architectures P2P existantes et visant à identifier les fonctionnalités et mécanismes indispensables au support d'applications P2P multi-plateformes offrant des fonctionnalités de haut niveau, Sun Microsystems Inc. a proposé l'architecture du framework JXTA.

Au coeur de cette architecture, on peut trouver des fonctions essentielles telles que la gestion de groupes et des communications entre membres de la communauté de `Peers`[4] JXTA. A un plus haut niveau, ces fonctionnalités de "base" peuvent servir à créer des services d'indexation, de recherche, de partage, ... Et pour finir, au-dessus de ces fonctions essentielles et services, les développeurs pourront développer des applications à valeur ajoutée.

Les différentes couches de l'architecture JXTA sont représentées sur la figure 1.

[4] Voir définition de `Peer` JXTA à la section 4.1

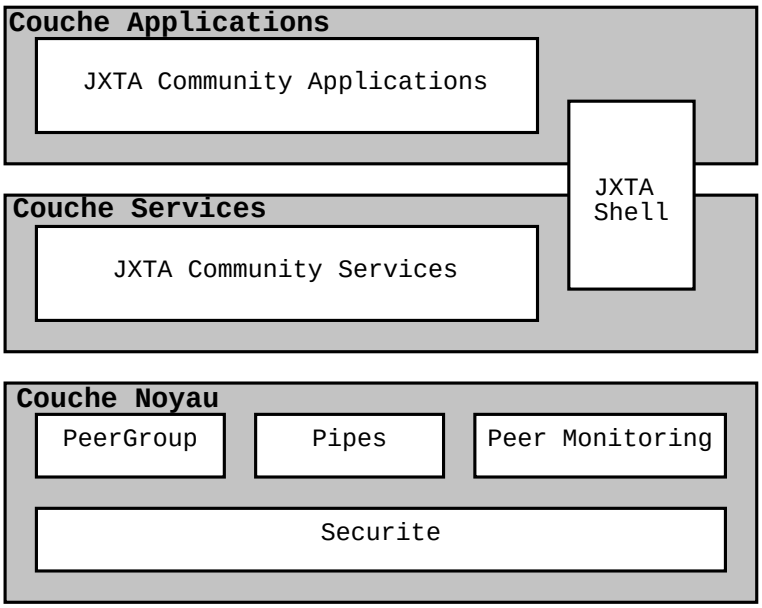

Fig. 1. Architecture logique du framework JXTA

3.1 La Couche Noyau

La `Couche Noyau` de l'architecture JXTA offre aux développeurs un ensemble de mécanismes de base tels que les `PeerGroups` (voir section 4.2), les `Pipes` (voir section 4.5) et le `Peer Monitoring` (voir section 4.3). Ces mécanismes permettent de bâtir des services et des applications dans un environnement d'exécution sûr et multi-plateformes.

Bien que fournissant les fonctions de base à la gestion de groupes et aux communications, la `Couche Noyau` permet également aux développeurs de choisir des politiques particulières et optionnelles telles que l'anonymat et le cryptage.

3.2 La Couche Services

La `Couche Services` du framework JXTA permet quant à elle d'étendre les mécanismes de base offerts par la `Couche Noyau` et ainsi faciliter le développement d'applications de plus haut niveau. Les principaux services proposés dans cette couche sont des mécanismes de recherche et d'indexation (JXTA Search [17][18][19]), de partage de ressources (JXTA cms et JXTA gisp [17]), de stockage (jxtaspaces [17]), de traduction de protocoles (JXTAxmlrpc, jxta-rmi, jxtasoap, ... [17]), etc.

Au moyen de ces services, effectuer des recherches simples (dans un repository) ou des recherches plus complexes portant sur des contenus générés dynamiquement serait tout à fait possible autant qu'imaginer l'implémentation d'un

"pont" permettant l'incorporation de résultats de recherches issus de l'Internet au sein d'un `PeerGroup` JXTA. [15]

Egalement, en fonction des choix de politique opérés par le développeur au niveau de la `Couche Noyau`, celui-ci pourra implémenter de nouveaux services comme par exemple un service "évolué" d'administration des `PeerGroups` se basant sur des fonctions offertes par la couche noyau. [15]

3.3 Le Shell JXTA

Le `Shell JXTA` est un outil situé à cheval sur les couches `Services` et `Applications` qui permet aux développeurs et utilisateurs expérimentés de tester leurs applications ainsi que de paramètrer plus finement l'environnement de leur `Peer` JXTA. Ce shell offre une interface en ligne de commande semblable aux shells Unix et des commandes facilitant l'accès aux fonctions du noyau JXTA telles que lister les `Peers` accessibles dans le `PeerGroup` du `Peer`, accéder aux fonctions de découverte de `Peers`, joindre ou quitter un `PeerGroup`, envoyer des messages à d'autres `Peers`, ...

3.4 La Couche Applications

Au sein de la `Couche Applications`, les applications seront développées sur base des services disponibles dans la `Couche Services` accédant aux mécanismes de base fournis dans la `Couche Noyau`.

Il est toutefois important de remarquer que le but du projet JXTA n'est pas le développement d'applications mais la fourniture des mécanismes fondamentaux indispensable au développement d'applications reposant sur la technologie P2P. La conception des applications est laissée au soin de la communauté d'utilisateurs et développeurs JXTA.

Par ailleurs, en plus des applications "classiques" de partage de fichiers, les auteurs de l'article [15] proposent une série d'exemples d'applications implémentables facilement au moyen du framework.

On peut imaginer ainsi des applications de partage de ressources de calcul telles que SETI@Home. Ces applications seraient, grâce à JXTA, plus facilement implémentables et déployables au sein d'un réseau d'hôtes hétérogènes.

Des applications de messagerie instantanée, de mailing et de gestion d'agendas entre membres d'un même `PeerGroup` peuvent offrir une aide à la collaboration des individus au sein d'un groupe et une plus grande indépendance vis-à-vis de fournisseurs de services tiers. On peut également envisager un système de messagerie anonyme avec gestion de la persistance des messages bâti sur les services offerts par la `Couche Services`.

Un dernier exemple consisterait en une extension de l'intranet d'une entreprise vers son extranet (partenaires, consultants et fournisseurs) en utilisant les possibilités offertes par le cryptage des communications entre `Peers` et par la gestion des appartenances aux `PeerGroups`.

4 Concepts

Cette section constitue la synthèse des références suivantes : [14][13][16] [20][15].

Cette section a pour objectif de présenter les concepts essentiels du framework JXTA tels que les `Peers`, les `PeerGroups`, le `Peer Monitoring`, les `Peer EndPoints`, les `Pipes`, les `Peer Routers`, les `Identifiants`, les `Messages`, les `Avertissements`, les `Credentials` et les `Modules`.

4.1 Peer

Un `Peer` JXTA est une entité (un processus, un processeur, un hôte ou un utilisateur) implémentant les protocoles indispensables du framework[5] et optionnellement les autres [6]. Cependant, les protocoles JXTA travaillant en *best effort* n'ont aucune obligation de fournir une réponse et un `Peer` qui ne les implémente pas tous peut quand même participer à la communauté JXTA mais de manière limitée en fonction des protocoles qu'il connait.

4.2 PeerGroup

L'architecture JXTA offre la possibilité à ses `Peers` de s'organiser en groupes. Un `PeerGroup` JXTA consiste en un groupe logique et dynamique, constitué de `Peers` ayant un intérêt commun (généralement la thématique du contenu ou des services partagés) et un ensemble de politiques communes (les conditions d'appartenance au groupe, les communications, la sécurité, ...).

Les avantages, souvent avancés ([20][16]), résidant dans l'utilisation de ces `PeerGroups` sont les suivants :

- permettre la création de domaines sécurisés pour l'échange de contenu par définition de politiques de sécurité particulières au sein d'un groupe.
- permettre la création de groupes partageant un même intérêt commun a pour effet de réduire la portée des requêtes et des découvertes des autres `Peers` en se limitant aux seuls `Peers` du groupe et non plus à toute la communauté.
- permettre la création d'environnements de monitorage du `PeerGroup` (voir section 4.3).

Pour les atteindre, le framework définit uniquement les mécanismes essentiels à la gestion de ces groupes. Ainsi, un ensemble de services de base est défini pour la gestion des `PeerGroups` tels que la découverte (voir section 5.4), la résolution (voir section 5.3), les `Pipes` (voir section 5.6), les informations sur les `Peers` (voir section 5.5), les `Rendezvous` (voir section 5.2) et le routage (voir section 5.1). Chacun de ces services répond à son propre protocole. Cependant, si l'un d'entre

[5] Le Endpoint Routing Protocol, le Rendezvous Protocol, le Peer Discovery Protocol, le Peer Resolver Protocol et le Pipe Binding Protocol

[6] Le Peer Information Protocol

eux (ou plusieurs) s'avère ne pas correspondre aux attentes d'un `PeerGroup`, il pourra être remplacé par un protocole plus adaptés et commun au groupe.

Il est important de remarquer qu'un `Peer` peut appartenir à plusieurs `PeerGroups` et que chaque `Peer` de la communauté JXTA appartient à deux "Meta PeerGroup" : le `NetPeerGroup` et `World PeerGroup` offrant les services de base du framework et leurs protocoles.

4.3 Peer Monitoring

Le `Peer monitoring` permet à un `Peer` de contrôler le comportement et l'activité des autres membres de son `PeerGroup`. Le `Peer monitoring` peut servir, par exemple, à implémenter un *Peer Manager* offrant des fonctions de contrôle des accès, de priorité de certains `Peers`, de mesure du trafic et de répartition de la bande passante disponible. [15]

4.4 Peer EndPoint

Les `Peer EndPoints` sont utilisés par les `Peers` JXTA pour établir entre eux des connexions directes. Chaque `Peer` peut annoncer les différentes interfaces de communications (GPRS, WiFi, Ethernet, BroadBand, ...) par lesquelles il peut être contacté. Chacune de ces interfaces constituera un `Peer EndPoint` et sera identifiée de manière unique au moyen d'une URI[7].

4.5 Pipes

Les `Pipes` sont des canaux de communication logiques établis dynamiquement entre les `Peer EndPoints` d'un ou plusieurs `Peers`. Ils permettent aux services et aux applications des différents `Peers` de communiquer. Ils consistent en une file de messages supportant un ensemble d'opérations tel que la création, l'ouverture/résolution, la fermeture, l'effacement, l'envoi et la réception.

Ils ont été conçus pour être asynchrones et unidirectionnels. Ainsi, un `Pipe` a deux fins : une fin réceptrice et une fin émettrice. Ceci implique, que généralement, ils vont par paire (un `Pipe` d'entrée et un `Pipe` de sortie) pour donner l'illusion d'une communication bi-directionnelle entre `Peers`. Ce choix a été motivé par la volonté d'offrir des canaux de communication supportant le transfert de données brut et le transfert de contenu[8], tout en limitant au maximum la dépendance vis-à-vis des couches transports sous-jacentes potentiellement utilisables actuellement ou à l'avenir.

Les `Pipes` peuvent être de deux types :

- Les `Point-to-point Pipes` permettent de connecter unidirectionnellement un `Peer` à un autre `Peer`. Deux `Pipes` sont alors nécessaires pour assurer une communication bi-directionnelle entre deux `Peers`.

[7] Voir [16]

[8] Un contenu peut être un fichier texte, un document structuré (un PDF ou un fichier XML) ou un contenu exécutable (Java .jar, une librairie, du code, un exécutable, ...)

– Les `Propagate pipes` permettent de connecter un `Pipe` de sortie à un ou plusieurs `Pipe(s)` d'entrée. Ce type de `Pipe` permet d'envoyer en une seule fois un message à plusieurs destinataires dont les `Pipes` d'entrée sont connectés au `Pipe` de sortie de l'émetteur du message.

Il est important de remarquer que le fonctionnement interne d'un `Pipe` n'est pas défini, laissant ainsi au développeur le choix du protocole de transport. Les exigences liées au protocole de transport sont assez faibles (asynchrone, unidirectionnel, en mode datagramme et non fiable) ce qui laissera au développeur un large choix possible de protocoles de transport. Ainsi, un grand nombre de protocoles "unicast" et "multicast" pourront être utilisés pour implémenter les `Pipes` et permettront différentes qualités de `Pipes` (asynchrone et non-fiable, synchrone et fiable, ...).

4.6 Peer Router

Les `Peer Routers` ont pour mission principale de trouver la ou les route(s)[9] permettant d'acheminer un message de son émetteur à son destinataire.

Cependant, dans un réseau aussi varié que l'Internet quant à sa topologie physique et aux divers protocoles employés, il est très fréquent que des `Peers` ne puissent pas communiquer directement. Ce cas de figure peut avoir plusieurs causes telles que les connexions physiques (dial-up de fournisseur Internet et leurs adresses ip dynamiques) ou des configurations de réseau particulières (NAT, Firewall, Proxies,...). Pour tenter de contourner ces problèmes de topologie physique de réseau et ce indépendamment de la couche transport choisie pour l'implémentation (TCP/IP à port fixe ou non, XML-RPC, SOAP, ...), les `Peers Routers` JXTA peuvent jouer le rôle de boîte aux lettres "spoolant" les messages à destination d'un `Peer` injoignable directement. Celui-ci en relève les messages qui lui sont destinés dès qu'il en a la possibilité.

Les auteurs du framework et la communauté JXTA sont conscients que cette solution n'est pas optimale et sont toujours en quête d'une solution plus efficace.

4.7 Identifiant

Au sein de l'architecture JXTA, chaque entité (`Peer`, `PeerGroup`, contenu, `Avertissement`, ...) est identifiée par un identifiant unique (UUID). Les UUID JXTA sont présentés sous forme d'Uniform Ressource Name (URN)[10]. Ces URNs sont des identifiants uniques, persistants et indépendants de la localisation des ressources qu'ils identifient.

De manière à garantir, au sein de la communauté JXTA tout entière, l'unicité d'un UUID, une partie de celui-ci est fabriquée sur base d'un nombre de 64 bytes

[9] Une route est constituée d'un ensemble de hops (Peers) reliant l'émetteur d'un message et son destinataire.

[10] Voir rfc [21]

généré par un algorithme assurant une très haute probabilité d'unicité dans le temps et l'espace de la communauté .

Exemple d'un UUID de `Peer` :

```
urn:jxta:uuid-59616261646162614A7874615032503304BD268FA4764960AB93A53D7F15044503
```

Actuellement, seulement cinq types d'identifiants ont été définis : les identifiants de `Peers`, `PeerGroups`, `Pipes`, `Codats` [11] et `Modules`.

4.8 Avertissements

Les `Avertissements` décrivent des ressources disponibles dans la communauté JXTA. Ils sont échangés entre `Peers` au moyen des protocoles JXTA et sont définis au moyen d'un document XML contenant des informations sous forme de metadata.

Le framework définit les `Avertissements` de base suivants : `Peer advertisement`, `PeerGroup advertisement`, `Pipe advertisement`, `Module advertisement`, `PeerInfo advertisement`, `Content advertisement` et `PeerEndpoint advertisement`. Ces `Avertissements` peuvent cependant être enrichis au moyen de "XML schema" pour former des `Avertissements` plus complets en informations et metadata.

Les `Peers` mettent en cache, publient et échangent des `Avertissements` pour publier ou trouver des ressources disponibles. Tous ces avertissements sont émis avec un temps de vie indiquant la durée de vie de la ressource dans le système. Par ailleurs, un avertissement peut être republié pour étendre la durée de vie de la ressource qu'il désigne avant que l'avertissement précédent publié n'expire.

Par exemple, un `PeerGroup Advertisement` décrit un `PeerGroup` au moyen de son identifiant unique (GID), l'identifiant de son module de spécification (MSID) correspondant à l'ID de l'avertissement du module de spécification dans le `PeerGroup`, le nom du `PeerGroup` et sa description.

Les `Pipe advertisements` contiennent quant à eux un identifiant unique du `Pipe` (PipeId), son type (Unicast ou Propagate) et un nom symbolique optionnel.

Exemple d'un `Pipe Advertisement` [16] :

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE jxta:PipeAdvertisement>

<jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">
  <Id>urn:jxta:uuid-094AB61B99C14A ... A26BB362B34D1F104</Id>
  <Type>JxtaUnicast</Type>
  <Name>Talk to Me!</Name>
</jxta:PipeAdvertisement>
```

[11] Un codat est l'association d'un contenu (texte,code,...) et un UUID

4.9 Messages

Les `Peers` JXTA interagissent en s'échangeant des `Messages` encodés selon le `JXTA Message Wire Representations` et permettent deux représentations possibles : binaire ou XML[12]. Le choix de la représentation dépendra du type de couche transport utilisée : TCP/IP entraînera le choix du format binaire tandis que RPC-XML ou SOAP entraîneront XML.

Un `Message` est une séquence d'éléments ordonnés[13]. Chaque élément est composé d'un `NAMESPACE`, d'un `NAME`, d'un `TYPE`[14] et d'un `CONTENT`. Souvent, le `CONTENT` sera au format XML car il est utilisé pour l'échange d'avertissements entre les couches des `Peers` JXTA.

Ainsi, lorsqu'un message descend la pile de protocoles JXTA (voir fig. 2), ceux-ci pourront ajouter un ou plusieurs élément(s) au message. Inversément, lorsqu'un message remonte la pile, chaque protocole concerné par un élément doit l'extraire du message. Juste avant son envoi par la couche transport, le message sera "enveloppé" d'un `HEADER` et des `Peer EndPoints` de la source et de la destination du `Message`.

Le choix d'écrire les messages en XML a été motivé par les arguments suivants [13] :

- Simplifier la re-définition et le re-codage de JXTA en cas de changement de norme.
- L'utilisation de XML n'implique pas que tous les `Peers` soient capables de parser et créer tous les documents XML possibles. Un `Peer` peut se limiter à la reconnaissance de certains types de messages XML.

4.10 Credentials

Afin d'accéder aux diverses ressources de la communauté sur base d'un modèles de confiance reposant sur les rôles, le format des messages JXTA permet aux applications d'ajouter des informations arbitraires sous forme de "metadata". Ces informations peuvent être des certificats, des clés publiques, des résumés du contenu et des justificatifs d'identités (credentials).

L'ajout d'un justificatif d'identité permet, lorsqu'il est présent dans le corps d'un message, d'identifier l'expéditeur et de vérifier ainsi qu'il a le droit d'envoyer le message à un `Peer EndPoint` déterminé. Les résumés, quant à eux, garantissent l'intégrité des messages. Les certificats et clés de cryptage permettent d'encrypter et signer les messages pour augmenter leur confidentialité et empêcher le "replaying".

---

[12] Les types MIME sont utilisés pour l'encodage de données autres que du texte.
[13] Le dernier élément ajouté apparaît à la fin.
[14] Le TYPE d'un CONTENT d'un élément est un type MIME tel que défini dans la RFC2046.

4.11 Modules

Un `Module` est une définition du code exécutable d'un service ou d'une application proposée par un `PeerGroup` dont l'API est propre à chaque environnement d'exécution.

Pour cette raison, un module sera décrit par trois types d'avertissements :

- Un `Module Class Advertisement` décrit une classe de modules offrant le même service.
- Un `Module Specification Advertisement` décrit les spécifications d'un module et la méthode pour l'invoquer et l'utiliser. Un module peut être utilisé à travers son API, en localisant son implémentation, puis en le chargeant et ensuite en le démarrant ou bien en l'utilisant à travers un `Pipe` ou à travers un *proxy* du `Module`.
- Un `Module Implementation Advertisement` décrit une implémentation d'une spécification de module au moyen de son type d'environnement d'exécution, le nom du module et le contenu de ses paramètres.

JXTA ne fait que définir les moyens des publier et découvrir les Modules de services. Il laisse le choix du mode d'invocation. Ainsi, des techniques d'invocations telles que les Services Web, RMI, ... pourront être utilisées.

5 Protocoles

Cette section constitue la synthèse des références suivantes : [14][16]

Cette section a pour objectif de présenter les 6 protocoles JXTA de base et leurs principes de fonctionnement.

La figure 2 illustre la pile de protocoles JXTA et leurs dépendances pour l'envoi et la réception de messages.

5.1 Endpoint Routing Protocol

Le `Endpoint Routing Protocol` (ERP) permet à un `Peer` de trouver les routes disponibles pour l'envoi d'un message à un autre `Peer`. Pour obtenir ces routes, le `Peer` émetteur interroge un `Peer Router` et ce dernier lui fournit une liste de `Peer Routers` menant jusqu'au `Peer` de destination. Cependant, il est fréquent que, lorsque deux `Peers` communiquent ensemble, ils ne soient pas directement connectés par le même protocole de transport ou qu'ils soient connectés de manière "transient" (dial-up) ou soient séparés par un NAT ou un Firewall. Dans ces situations, un `Peer Router` jouera le rôle d'intermédiaire pour le `Peer` inaccessible de l'extérieur de son réseau interne (voir section 4.6).

Un `Peer` a le choix de devenir un `Peer Router` en choisissant d'implémenter le protocole ERP.

ERP définit un ensemble de messages de requêtes et leur traitement permettant aux `Peers` de router les messages jusqu'à destination.

Lorsqu'un `Peer` doit envoyer un message jusqu'à un `Peer EndPoint` déterminé, il vérifie tout d'abord dans sa cache locale s'il connaît déjà une route

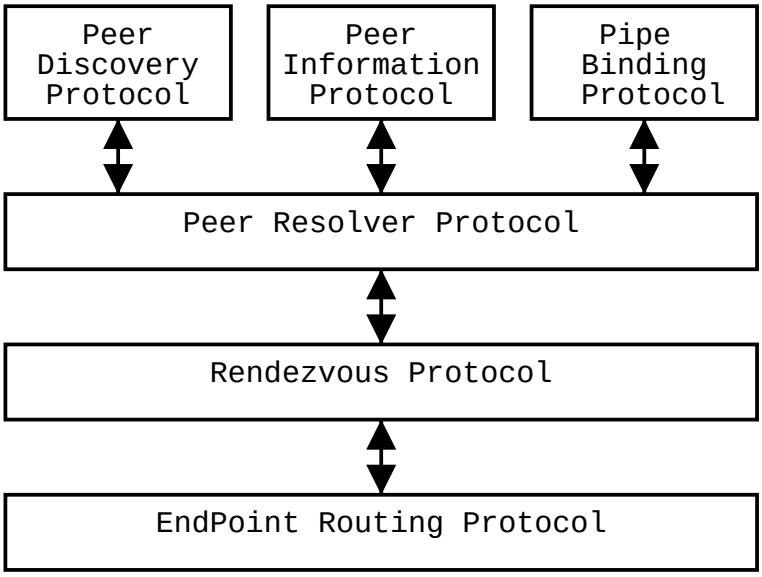

Fig. 2. Pile des protocoles JXTA [20]

menant à ce `Peer EndPoint`. Si ce n'est pas le cas, il envoie un message contenant une requête de résolution de route aux `Peer Routers` qu'il connaît[15].

Suite à la réception d'une requête de résolution de route, si un `Peer Router` connaît une route menant à la destination, il répond à la requête en retournant l'information de route (un liste ordonnée du `Peer Router` à la destination finale) qui sera utilisée par l'émetteur du message et les `Peers` intermédiaires. S'il ne connaît pas de route, il effectue une recherche auprès des `Peer Routers` auxquels il est connecté. Ce mécanisme est récursif car à chaque intermédiaire, si la route d'un message est obsolète elle nécessitera une nouvelle recherche de route.

Pour éviter les boucles, chaque `Peer EndPoint` intermédiaire ajoutera de l'information aux messages envoyés par un `Peer`. Cette trace sera utilisée pour ne plus propager un message qui boucle, un message dupliqué ou pour enregistrer une nouvelle information de route par un `Peer Router`.

Comme pour les autres protocoles JXTA, ERP fournit les mécanismes essentiels au routage des messages (format des routes et des messages pour les échanger) et laisse la possibilité aux développeurs d'implémenter des services de routage plus efficaces et plus intelligents optimisant les routes choisies, connues et publiées.

[15] Les `Peer Routers` peuvent être connus d'un `Peer` par pré-configuration ou découverts dynamiquement au moyen d'une recherche de `Peer`.

5.2 Rendezvous Protocol

Le `Rendezvous Protocol` (RVP) permet la propagation des `Messages` et son contrôle (TTL, détection de boucle, ...) au sein d'un `PeerGroup` . Comme ERP introduit ses `Peer Routers`, RVP quant à lui introduit des `Peers` jouant un rôle spécial appelés `Rendezvous Peers`. Ces `Rendezvous Peers` re-propagent les messages qu'ils ont reçus. Un `Peer` peut devenir dynamiquement un `Rendezvous Peer` et/ou peut négocier dynamiquement un bail avec un `Rendezvous Peer` au moyen du `EndPoint Routing Protocol`.

L'ensemble de messages définis par RVP pour établir un bail entre un `Peer` et un `Rendezvous Peer` sont :

- LeaseRequest : ce message est envoyé par un `Peer` désirant se connecter à un `Rendezvous Peer`. Un `Rendezvous Peer` qui accepte de fournir un bail retourne un LeaseGrant. Un bail peut à tout moment être annulé par les deux parties.
- LeaseGranted : ce message est retourné par un `Rendezvous Peer` s'il a accepté la requête de bail d'un `Peer`. La durée du bail est fixée par le `Rendezvous Peer` et est fournie dans le message.
  LeaseCancel : ce message est envoyé par un `Peer` pour annuler un bail existant.

Par ailleurs, RVP assure aussi le contrôle de la propagation des messages au moyen d'un élément Time-To-Live et d'un cache des identifiants de `Messages`. RVP ne propage pas un message dans les situations suivantes :

- En cas de duplication d'un message ou de boucle : chaque message propagé est identifié de manière unique. Quand un message est dupliqué c'est-à-dire qu'il a déjà été reçu par le peer, il sera jeté.
- En cas d'expiration du Time To Live : les messages propagés sont associés à un Time To Live (TTL) décrémenté à chaque passage par un `Peer`. Lorsque le TTL d'un message atteint zéro, le message est jeté.

5.3 Peer Resolver Protocol

Le `Peer Resolver Protocol` (PRP) permet, au sein d'un `PeerGroup`, l'envoi de requêtes génériques adressées à un ou plusieurs contacts (handlers) et l'identification des réponses correspondantes. PRP fournit les mécanismes minimaux pour la gestion des requêtes et leurs réponses mais laisse la liberté au développeur de concevoir des services de résolution ayant une meilleure connaissance de la topologie du groupe.

Chaque requête est adressée à un nom de contact spécifique. Ce nom de contact définit la sémantique particulière de la requête et de sa réponse, mais n'est pas associée à un `Peer` spécifique. Une requête peut être reçue par n'importe quel nombre de `Peers` d'un `PeerGroup`, voire tous, et sera traitée par un `Peer` en fonction du nom du contact s'il est défini sur ce `Peer`.

Ce nom de contact permet de savoir à quel service de haut niveau est destiné le message pour être traité. Par exemple, les protocoles Peer Discovery Protocol (voir section 5.4), Peer Information Protocol (voir section 5.5) et Pipe Binding Protocol (voir section 5.6) s'inscrivent comme service auprès de PRP au moyen d'un nom de contact en vue de recevoir les messages qui leur sont destinés.

Dans les deux implémentations de référence (Java et C) du framework, les instances de services génèrent un nom de contact unique sur le `Peer` mais identique pour les instances situées sur d'autres `Peers`. Par convention, on concatène le nom du service et l'identifiant du groupe pour rendre ce nom de contact commun au groupe. A cela, on ajoute des paramètres additionnels pour assurer l'unicité au sein d'un `Peer`.

Les peers peuvent également participer à un index distribué des ressources partagées (Shared Resource Distributed Index (SRDI)). Cet index fournit un mécanisme générique que les services JXTA peuvent utiliser pour diriger les requêtes dans une direction où elle a plus de chance d'obtenir une réponse ou pour propager des messages vers des `Peers` susceptibles d'être intéressés par le type de messages à propagé.

Cependant, PRP n'offre aucune garantie quant à la réception des requêtes ou des réponses correspondantes. Il utilise comme PDP le principe du "best effort" pour envoyer les requêtes ; ce principe maximise la chance d'obtenir une réponse si elle peut être obtenue. De même, le protocole n'est pas tenu de répondre à une requête. Il n'y a pas non plus de garantie qu'un message SRDI soit honoré. PRP ne suppose pas non plus la présence d'un transport fiable des messages.

La propagation des requêtes à l'ensemble suivant de `Peers` est laissé à RVP (voir section 5.2). RVP est responsable de déterminer l'ensemble de `Peers` qui doivent recevoir le message à propager. Cependant, il ne propage pas automatiquement un message, il laisse le service (le contact de la requête) se charger de déterminer si le message doit être propagé ou non.

La politique de PRP est la suivante : si le contact de la requête ne donne pas pour instruction de "jeter" la requête et si son `Peer` est un `Rendezvous`, alors la requête est repropagée (en fonction de la gestion du TTL et de la détection de boucle de RVP). Si le contact de la requête le désire, une requête identique peut être réémise avec le `Peer` comme émetteur.

5.4 Peer Discovery Protocol

Le `Peer Discovery Protocol` (PDP) permet de découvrir les ressources publiées par les `Peers` d'un `PeerGroup` au moyen d'avertissements (voir section 4.8). PDP aide les services des couches supérieures en prenant en charge les détails de gestion des `Avertissements` (messages, cache, expiration, ...). Il constitue le protocole de découverte par défaut du WorldPeerGroup. C'est pour cette raison, qu'il fournit les mécanismes de base permettant la découverte des `Avertissements` tout en servant de support à la conception éventuelle de services de découverte spécifiques au groupe et ayant une meilleure connaissance de la topologie du groupe. PDP utilise le `Peer Resolver Protocol` (voir section 5.3) pour envoyer ses requêtes et recevoir les réponses.

Il est important de souligner que PDP ne garantit pas la réponse d'un `Peer` à une requête et le principe du "best effort" est appliqué pour la concordance entre les résultats des requêtes et le contenu de la cache d'avertissement.

5.5 Peer Information Protocol

Le `Peer Information Protocol` (PIP) permet d'obtenir, au moyen d'échanges de messages, des informations sur l'état d'un `Peer`. PIP est un protocole optionnel et les `Peers` n'ont pas l'obligation de répondre à ses requêtes. PIP se situe au dessus de PRP (voir section 5.3), et l'identifiant de requêtes PRP est utilisé par les réponses aux requêtes PIP pour permettre leur "matching". Par ailleurs, les messages PIP contiennent un champ dont le format n'est pas spécifié afin de permettre son utilisation par des services de haut niveau.

5.6 Pipe Binding Protocol

Le `Pipe Binding Protocol` (PBP) permet aux `Peers` de lier (bind) des `Peer EndPoints` au moyen de `Pipes` en fonction des avertissements de `Pipe` reçus ou envoyés.

PBP se situe au-dessus de PRP et est toujours nécessaire pour lier les `Peer EndPoints` entre eux et ce quel que soit le type de couche transport du `Peer EndPoint` (HTTP, TCP, UDP, ...).

Tout comme les autres protocoles JXTA, PBP n'impose pas les réponses à une requête de résolution.

6 Conclusion

Au moyen de son framework JXTA, Sun offre aux développeurs et utilisateurs un ensemble de services de haut niveau permettant la création et l'utilisation d'applications reposant sur une communauté basée sur la technologie P2P. Cette communauté se "juxtapose" sur les différents réseaux et communautés existants. Ces services et applications ont l'avantage d'être interopérables, multi-plateformes et leur déploiement est facilité au moyen des modules.

Bien que les spécifications de JXTA aient pour qualité d'être les plus abstraites possibles et les moins empreintes de détails d'implémentation, cela les rend plus difficiles à comprendre. De plus, ce haut niveau d'abstraction laisse en suspens toute une série de questions qui sont reléguées comme étant des détails d'implémentation laissés au choix de l'implémenteur (Le fonctionnement interne des `Pipes` par exemple). Celui-ci devra s'inspirer de solutions connues provenant d'autres protocoles pour les implémenter. Du point de vue des performances, le parsing multiple de message XML par les protocoles, services et applications peut éventuellement nuire à la performance globale d'un `Peer`.

On peut également reprocher à JXTA l'hypothèse de non fiabilité quant à l'émission et à la réception des messages de même que le principe généralisé du

"best effort". En effet, ces principes imposent au développeur d'implémenter lui-même pour son `PeerGroup` un protocole plus fiable si un service qu'il développe nécessite une certaine qualité de service (fiabilité du protocole de transport et pas de "best effort" dans le traitement des requêtes et réponses).

Heureusement, le framework est déjà implémenté dans deux langages : Java et C. D'autres portages sont en cours (Python, Ruby, Perl, J2ME, PocketPC et SmallTalk). Dans les deux implémentations de références (C et Java), plusieurs protocoles de transports sont utilisés et garantissent un certain niveau de fiabilité (TCP/IP, HTTP et Transport Layer Security Version 1 [22]).

Pour terminer, d'autres aspects du framework n'ont pas pu être abordés dans cet article mais mériteraient d'être approfondis : la sécurité et JXTA [23][24], le framework de service de recherche JXTA Search [19][18], l'adaptation du framework pour les périphériques embarqués de faible puissance [25]

Références

1. Community, G. : (Gnutella community website) `http://www.gnutella.com/`.
2. eDonkey : (edonkey 2000) `http://www.edonkey2000.com/`.
3. : (Kazaa media desktop)
4. Foundation, J.S. : (Jabber) `http://www.jabber.org/`.
5. Microsoft : (Msn messenger) `http://messenger.fr.msn.be/`.
6. ICQ : (Icq) `http://web.icq.com/`.
7. Institute, S. : (Seti@home) `http://www.seti.org/science/setiathome.html`.
8. Dabek, F., Kaashoek, M.F., Karger, D., Morris, R., Stoica, I. : Wide-area cooperative storage with CFS. In : Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01), Chateau Lake Louise, Banff, Canada (2001)
9. Rowstron, A.I.T., Druschel, P. : Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In : Symposium on Operating Systems Principles. (2001) 188–201
10. Kubiatowicz, J., Bindel, D., Chen, Y., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., Zhao, B. : Oceanstore : An architecture for global-scale persistent storage. In : Proceedings of ACM ASPLOS. (2000)
11. Muthitacharoen, A., Morris, R., Gil, T.M., Chen, B. : Ivy : A read/write peer-to-peer file system. In : Proceedings of 5th Symposium on Operating Systems Design and Implementation. (2002)
12. Clarke, I., Hong, T.W., Miller, S.G., Sandberg, O., Wiley, B. : Protecting free expression online with freenet. IEEE Internet Computing **6** (2002) 40–49
13. Gong, L. : Project JXTA : A Technology Overview. (Sun Microsystems - White Papers)
14. Gong, L. : Jxta : A network programming environment. IEEE Internet Computing **5** (2001)
15. Microsystems, S. : Project JXTA : An Open, Innovative Collaboration. (Sun Microsystems - White Papers)

16. Microsystems, S. : JXTA v2.0 Protocols Specification. (Sun Microsystems - White Papers)
17. Microsystems, S. : (Jxta project) `http://www.jxta/org`.
18. Waterhouse, S., Doolin, D.M., Kan, G., Faybishenko, Y. : Distributed search in peer-to-peer networks. IEEE Internet Computing **6** (2002)
19. Waterhouse, S. : JXTA Search : Distributed Search for Distributed Networks. (Sun Microsystems - White Papers)
20. Traversat, B., Abdelaziz, M., Duigou, M., Hugly, J.C., Pouyoul, E., Yeager, B. : Project JXTA Virtual Network. (Sun Microsystems - White Papers)
21. Moats, R. : Rfc - urn syntax. Technical report, IETF (1997) `http://www.ietf.org/rfc/rfc2141.txt`.
22. Dierks, T., Allen, C. : The tls protocol version 1.0. Technical report, IETF (1999) `http://ietf.org/rfc/rfc2246.txt`.
23. Microsystems, S. : Security and Project JXTA. (Sun Microsystems - White Papers)
24. Altman, J.E. : PKI Security for JXTA Overlay Networks. (IAM Consulting, Inc.)
25. Arora, A., Haywood, C., Pabla, K.S. : JXTA for J2ME - Extending the Reach of Wireless With JXTA Technology. (Sun Microsystems - White Papers)
26. CM316 : Multimedia Systems coursework, E. : Gnutella vs. jxta. (a trouver plus de détails)
27. Verbeke, J., Nadgir, N., Ruetsch, G., Sharapov, I. : Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment. (Sun Microsystems - White Papers)
28. Microsystems, S. : Project JXTA : Technical Shell Overview. (Sun Microsystems - White Papers)
29. Chen, R., Yeager, W. : Poblano A Distributed Trust Model for Peer-to-Peer Networks. (Sun Microsystems - White Papers)
30. Microsystems, S. : Project JXTA Technology - Creating Connected Communities. (Sun Microsystems - White Papers)