Prise en main de JXTA

Cédric Tedeschi

31 août 2003

Table des matières

Chapitre 1

Les concepts du pair à pair (P2P)

1.1 Introduction

1.1.1 L'architecture client serveur

Aujourd'hui, la plupart des architectures fournissant des services sur le Net sont de type client/serveur : les clients se connectent au serveur et communiquent avec lui en utilisant un protocole de communciation donné, dans le but d'accéder à une ressource ou aux services qu'il fournit. Beaucoup d'applications basées sur l'Internet utilisent cette architecture (WWW, FTP, Telnet, SMTP, ...) Ce type d'architecture décharge le client au détriment du serveur sur lequel sont effectués la plupart des travaux ayant à voir avec la délivrance de services.

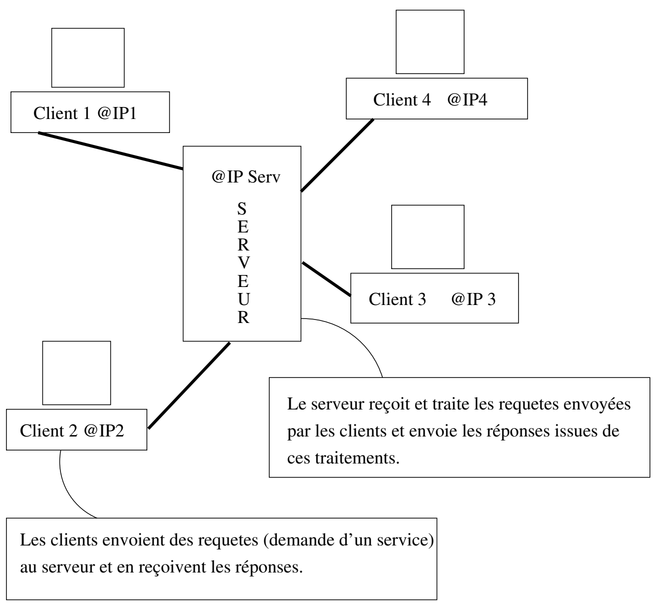

Fig. 1.1 – Exemple d'architecture client/serveur

L'inconvénient majeur de l'architecture client/serveur est le gouleau d'étranglement du serveur : Plus il y a de clients connectés au serveur, plus la demande en bande passante augmente... Donc, l'efficacité d'une telle architecture est limitée par les caractéristiques physiques du serveur. Un autre problème est apparu avec l'extension de l'Internet et l'apparition des adresses IP dynamiques (Les fournisseurs d'accès à Internet attribuent à chaque connexion d'une machine sur le réseau une adresse IP différente.) En effet, un serveur ne peut être accédé qu'en connaissant son adresse IP. Un autre problème est le développement des réseaux privés sécurisés par un firewall, qui filtre toute communciation initialisée depuis l'extérieur du réseau. Ces inconvénients peuvent être contournés dans un réseau en pair-à-pair.

1.1.2 L'architecture pair-à-pair

Alors que l'approche client/serveur propose une architecture fortement centralisée et un rôle bien défini pour chaque machine, le pair-à-pair (de l'anglais peer-to-peer(P2P)) offre une architecture fortement interconnectée et distribuée. Dans un réseau pair-à-pair, la distribution de services est répartie sur l'ensemble des "pairs" constituant le réseau, ce qui évite la disparition d'un service lors de l'apparition d'une anomalie sur un pair. De plus, les itinéraires de routage sont dynamiques et multiples, ce qui évite la congestion de certains noeuds du réseau.

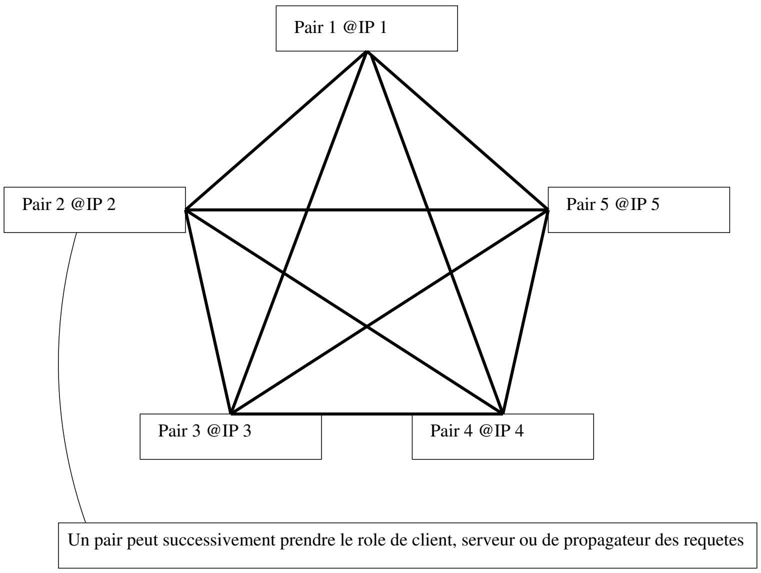

FIG. 1.2 – Exemple d'architecture pair-à-pair

Tout réseau développé en pair-à-pair doit contenir les éléments de bases : les **pairs** (de l'anglais peers.) Ces pairs communiquent entre eux via un certains nombre de protocoles, peuvent se regrouper au sein de **groupes de pairs** (peergroups) à l'intérieur desquels ils s'offrent mutuellement un certain nombre de **services**.

1.2 Les éléments de base du pair-à-pair

1.2.1 Les pairs

Un pair est l'entité (logique) de traitement fondamentale d'un réseau pair-à-pair. Un pair peut être réparti sur plusieurs machines physique, et une machine physique peut contenir plusieurs pairs. Un pair est capable de fournir des services (réaliser un travail) et de communiquer le résultat issu de ce travail directement ou indirectement. Un pair peut prendre plusieurs profils successivement ou simultanément :

- Pair simple,
- Pair de rendez-vous,
- Pair de routage.

Un **pair simple** (simple peer) permet à un utilisateur de fournir des services et de profiter des services disponibles sur les pairs qu'il connait (dont il a l'**avertissement**.) Souvent un tel pair se trouve derrière un firewall, il ne sera alors pas capable de communiquer directement avec d'autres pairs.

Les pairs envoient une requête de découverte (discoveryQuery) au **pair de rendez-vous** qui renvoit alors les informations au sujet des pairs qu'il connait. Il peut stocker ces informations pour un usage ultérieur et ainsi augmenter ses performances. Un pair de rendez-vous n'est pertinant qu'à l'extérieur d'un réseau protégé par un firewall, sinon, il doit pouvoir le traverser, soit en utilisant un protocole autorisé par le firewall (HTTP), soit en se connectant à un pair de routage à l'extérieur de ce réseau.

Un **pair de routage** fournit un mécanisme permettant aux pairs derrière un firewall (ou derrière une technologie NAT) de communiquer avec d'autres pairs. Pour envoyer un message à un pair via un pair de routage, un pair doit d'abord déterminer quel pair de routage utiliser. Le pair-à-pair propose un mécanisme permettant à un équipempent ayant une adresse IP dynamique de communiquer avec le réseau. Ce service associe dynamiquement une adresse IP et un port à un identificateur unique, propre à chaque pair.

1.2.2 Les groupes de pairs

Un **groupe de pairs** est un ensemble de pairs regroupés autour d'un intérêt commun. Ces pairs peuvent restreindre l'accès à leurs services aux membres du groupe. Les raisons de ces regroupements sont diverses : Besoin de travailler sur des données à accès restreints, raisons de sécurité, gestion et/ou surveillance d'un ensemble de pairs.

Les groupes de pairs fournissent plusieurs fois la même application (elle est présente sur plusieurs pairs. Le but de la surveillance des pairs est de s'assurer que le service est disponible en permanence sur au moins un pair.

1.2.3 Le transport

Dans tout réseau pair-à-pair, on trouve un ensemble de mécanismes permettant la transmission des données sur le réseau, c'est la couche transport. Cette couche rend les paquets de données conformes pour permettre à chaque pair de les comprendre (ajoût d'entête des protocoles, ...) Le transport comporte trois éléments fondamentaux : les points finaux, les canaux (de communication), les messages.

- Un **point final** (endpoint) est la source et la destination de toute donnée transmise sur le réseau. Un point final est une interface d'accès au réseau pour la transmission et la réception de données. Chaque pair peut avoir plusieurs point finaux.
- Les **canaux de communication** (pipes) sont unidirectionnels et asynchrones (de par le routage notamment, les paquets de données n'arrivent pas forcément dans l'ordre de leur transmission.) Ces canaux sont des liens virtuels entre deux machines distantes (entre deux points finaux créés au préalable : un d'entrée (input pipe) et un de sortie (output pipe) et ne sont qu'une abstraction de la couche transport, responsable de l'acheminement des données de bout en bout. Il est possible de créer des canaux bidirectionnels : il faut créer deux canaux unidirectionnels entre deux pairs.
- Les **messages** sont des données mises en forme selon un protocole donné et sont transmis par un canal de communication.

Lors d'une communication via un canal, le pair connecté au canal via le point final de sortie crée un message contenant les données et l'envoie, le pair connecté au canal via le point final d'entrée reçoit le message et en extrait les données.

1.3 Les services

Les services sont des ensembles de fonctionnalités qui permettent de travailler pour d'autres pairs. Des exemples courants sont : le transfert de fichiers, fournir des informations d'états d'autres pairs, realiser un calcul, etc... Sans service, rien n'est possible dans un réseau en pair-à-pair, tout est service depuis le routage jusqu'à la résolution de problèmes. Il existe deux types de services : les services de pairs et les services de groupes de pairs. La plupart des services sont offerts aux seins de groupes. Exemple : le service de routage est fourni par le "World Peer Group" (premier groupe auquel appartient tout pair nouvellement créé.)

- Les **services de pairs** sont des fonctionnalités isolées offertes par un pair donné, ce service est bien sur disponible que lorsque ce pair est connecté au réseau.
- Les **services de groupes de pairs** sont des fonctionnalités offertes par le groupe à l'ensemble des pairs membres de ce groupe. Ces fonctionnalités sont souvent présentes sur plusieurs pairs du groupe. Le service existe tant qu'au moins un des pairs offrant le service est connecté au réseau.

1.3.1 Les avertissements

Un avertissement (advertisement) est une représentation structurée d'une entité (pair, groupe, canal, point final, service, ressources, ...) rendu disponible par un pair et faisant parti du réseau pair-à-pair. Toute entité du réseau pair-à-pair a un avertissement lui correspondant contenant les informations importantes la concernant (type, identificateur, ...)

1.3.2 Les protocoles

Tout échange de données au sein d'un réseau pair-à-pair est dicté par un protocole de communication qui donne les règles de l'échange (qui, quoi, quand, comment) Les pairs acteurs de l'échange se sont au préalable mis d'accord pour suivre un protocole donné. les protocoles du pair-à-pair sont les suivants :

- découverte de pairs
- découverte de services
- obtention d'information d'états concernant des pairs
- demande de résolution d'un problème par un service
- gestion des groupes de pairs (création, adhésion, départ)
- création des éléments de connexion (canaux, points finaux)
- routage des messages

1.3.3 Mise en oeuvre de la communication pair-à-pair

Pour réaliser un réseau en pair-à-pair, il faut que tout pair soit capable de découvrir les pairs et services constituant le réseau, pour cela, il doit :

- acquérir les avertissements concernant ces pairs et services,
- pouvoir traverser les firewall (et les équipements de type NAT)

La **découverte des avertissements** se fait de deux façons :

- La **découverte directe** : Tout pair doit être capable de découvrir les autres pairs sur son réseau local (sans passer par un pair de rendez-vous. Une telle découverte requiert l'utilisation de la diffusion (broadcast) ou du multicast. Une fois que les pairs d'un même réseau physique se sont découverts, ils peuvent communiquer directement. Ce type de découverte est cependant limitée aux réseaux locaux.
- La **découverte indirecte** : Cette découverte se fait en envoyant une requête à un pair de rendez-vous. Les pairs de rendez-vous ont plusieurs façons de localiser des avertissements (de pairs et autre) :
  - La propagation : Le pair de rendez-vous recevant une requête de découverte la propage à tous les pairs qu'il connait (pairs de rendez-vous inclus)
  - Le stockage des avertissements : Chaque pair peut stocker les avertissements qu'il connait, dans le but de réduire le trafic sur le réseau. Ainsi, le pair de rendez-vous peut directement répondre au pair initiateur de la requête avec ses avertissements stockés.

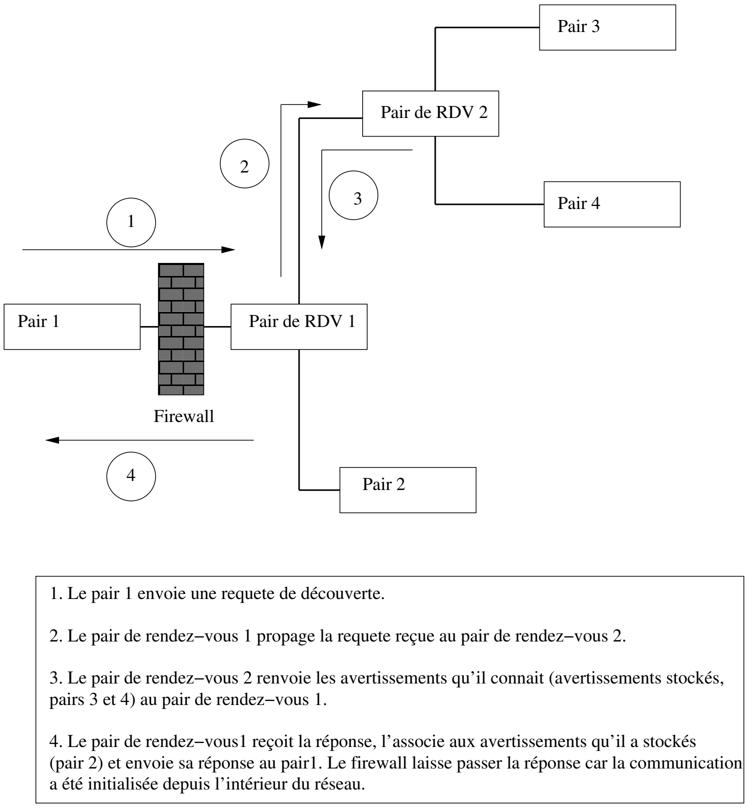

Fig. 1.3 – Découverte indirecte

Souvent, la solution retenue est la superposition des deux méthodes pour offrir un service le plus performant possible. Un inconvénient est qu'il est possible de faire boucler une requête de découverte, et ainsi d'engendrer un trafic inutilement élevé sur le réseau. Les requêtes de découvertes sont équipés d'une TTL (Time To Leave) qui est décrémentée à chaque passage dans un pair de rendez-vous. Lorsqu'elle atteint 0, la requête est détruite.

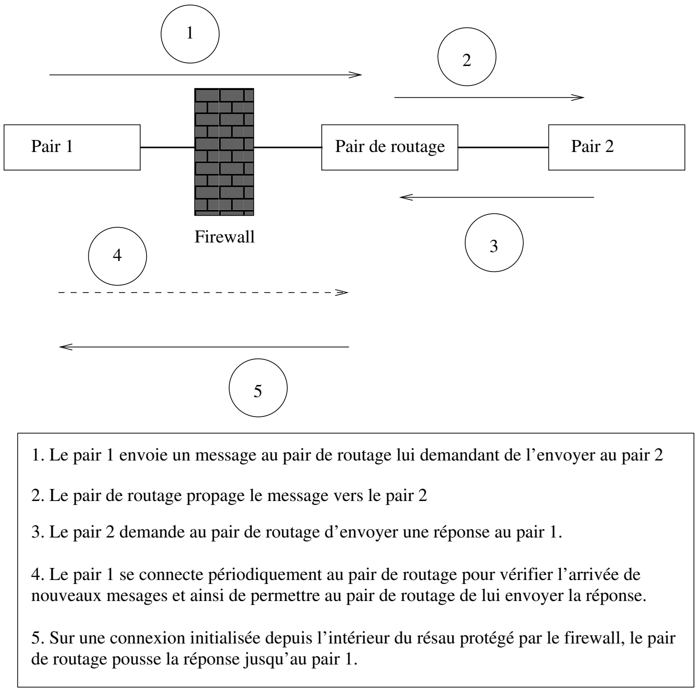

FIG. 1.4 – Communication traversant un firewall

Le passage des firwalls et des technologies NAT (Network Adress Translation), dispositifs n'autororisant que les communications initialisées depuis l'intérieur du réseau protégé, se fait par le protocole HTTP qui est un protocole qui, à chaque requête attend une réponse : Pour communiquer avec un pair derrière un firewall, il faut que ce pair se connecte périodiquement à un pair de routage (les pairs derrière le firewall ne sont jamais connectés directement à un pair simple à l'extérieur du réseau) pour permettre au pair à l'extérieur du réseau protégé d'établir une connexion via le pair de routage, qui sera acceptée par le firwall, puisque initialisée par le pair à l'intérieur du réseau.

Chapitre 2

Services et protocoles de JXTA

Dans la suite de ce manuel, nous utiliserons les noms anglais des protocoles et des composants JXTA pour faciliter la compréhension et le rapprochement des concepts et de la programmation.

2.1 Manipulation des protocoles, utilisation des services

Chaque protocole est mis en oeuvre de chaque côté de la communication par un service, qui offre les primitives nécessaires à la réalisation de ce protocole.

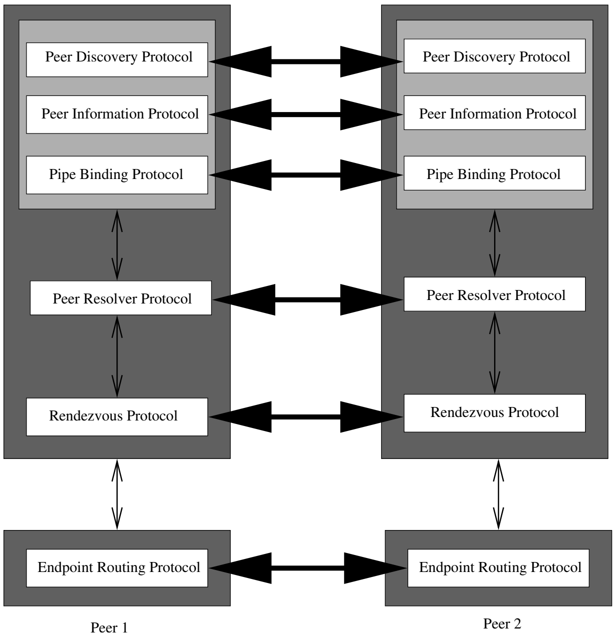

FIG. 2.1 – Pile de protocoles JXTA

JXTA est composé de six protocoles qui permettent la réalisation des différentes fonctionnalités inhérentes à tout réseau P2P. Ces protocoles sont basées sur l'échanges de messages XML. Comme dans toute pile de protocoles, il y a différents niveaux de protocoles : un niveau Transport (qui permet l'abstraction du routage et un niveau application, qui peut se découper en sous-niveaux : un niveau permettant l'abstraction de l'utilisation des rendezvous peers, un niveau donnant les bases de toute communication de niveau application, et enfin, trois protocoles qui répondent à des fonctionnalités précises du P2P, en se basant sur le protocole précédent.

Chaque protocole de la pile JXTA encapsule les protocoles de niveaux inférieurs et permettent de s'affranchir des contraintes physiques de la machine lors de l'utilisation du protocole de niveau transport, de s'affranchir du transport lors de l'utilisation des protocoles de niveaux applicatifs. En effet, les trois protocoles applicatifs de plus haut niveau s'appuient sur le Peer Resolver Protocol, qui s'appuie ainsi que le Rendezvous Protocol sur le Endpoint Routing Protocol qui lui-même s'appuie sur le protocole d'échange physique du réseau sur lequel le peer se trouve.

Chacun de ces protocoles est mis en oeuvre de chaque côté de la communication par les services associés. On a ainsi six services dans le noyau JXTA, à chacun de ces services correspond une interface fournit par JXTA :
- le **Discovery Service**,
- le **Pipe Service**,
- le **Peer Info Service**,
- le **Resolver Service**,
- le **Rendezvous Service**,
- le **Endpoint Service**.

2.1.1 Le Endpoint Routing Protocol (ERP)

Le Endpoint Routing Protocol s'appuie sur la couche transport du réseau sur lequel le peer se trouve. Ce protocole permet d'envoyer des messages entre deux peers sur des réseaux physiques différents. JXTA fournit deux interfaces (`EndpointListener` et `Messenger`) permettant la mise en oeuvre du Endpoint Service, lui-même permettant l'échange de messages via le Endpoint Routing Protocol.

Lorsque l'on veut mettre en oeuvre ce service sur un peer, il faut écrire une implémentation de l'interface EndpointListener (permettant la réception et le traitement des messages arrivant par le Endpoint Routing Protocol) et utiliser la méthode `propagate` de l'interface `EndpointService`.
- Dans l'implémentation de l'EndpointListener destinataire du message, il faut réécrire la méthode `processIncomingMessage` (qui traite les messages arrivant par le Endpoint Routing Protocol.) Lors du lancement du peer implémentant cette interface, il faut :

  - Instancier notre classe implémentant l'interface `EndpointListener`,
  - récupérer le `EndpointService` du peer Group dans lequel on travaille (par défaut, le WorldPeerGroup),
  - ajouter notre EndpointListener au groupe créé.

  Notre peer est maintenant capable de recevoir et de traiter les messages envoyés par le Endpoint Routing Protocol.
- Dans l'implémentation de l'EndpointListener expéditeur du message, il faut :
  - récupérer le `EndpointService` de notre peer group,
  - créer le message à envoyer,
  - l'envoyer en appelant la méthode `propagate` (envoi à tous les endpoints connus) de l'objet`EndpointService` précédemment créé.

  Le `EndpointListener` de notre peer destinataire recevra et traitera ce message avec la méthode `processIncomingMessage`.

Nous avons vu comment deux peers peuvent communiquer via le Endpoint Routing Protocol connaissant le chemin d'un peer à l'autre. l'ERP permet à un message d'être envoyé à un peer distant en utilisant une route découverte au préalable. Pour déterminer cette route, JXTA fournit deux messages XML :
- Le **Route Query Message** est envoyé par un peer quand il veut connaître les peers qu'il devra traverser pour aller jusqu'à un Endpoint donné. Ce message contient l'advertisement du peer lançant la requête et l'adresse de l'endpoint qu'il veut atteindre.
- Lors de la réception d'un tel message, un peer renvoit une réponse : un **Route Response Message** décrivant l'ensemble des routeurs à passer, dans le champ XML `<GatewayForward>`.

Une fois une réponse reçue et donc une route définie, le peer peut créer un **Endpoint Router Message** qu'il accolera au message qu'il veut envoyer, et qui contient les informations nécessaires à l'ERP sur l'itinéraire à suivre pour atteindre l'endpoint désiré.

2.1.2 Le Rendezvous Protocol (RVP)

Le RendezVous Protocol s'appuie sur le Endpoint Routing Protocol et fait donc abstraction du routage. Il permet la connexion d'un peer à un rendezvous peer dans le but de propager des messages via ce dernier. Ce protocole s'appuie sur trois messages :
- Le **Lease Request Message** (LRM) : message utilisé par un peer pour demander le droit d'utiliser un rendezvous peer pour la propagation de ses messages, ce message XML contient l'advertisement du peer demandant une connexion.
- Le **Lease Granted Message**(LGM) : message utilisé par un rendezvous peer pour signifier à un peer son accord de l'utiliser pour propager des messages, ce message XML contient l'ID du rendezvous peer accordant une connexion, ainsi qu'une chaine de caractère spécifiant le temps de la connexion accordée, en millisecondes.
- Le **Lease Cancel Message**(LCM) : message utilisé par un peer pour informer un rendezvous peer qu'il se déconnecte. Il ne l'utilise plus pour propager ses messages.

Ce message contient l'advertisement du peer annulant sa connexion.

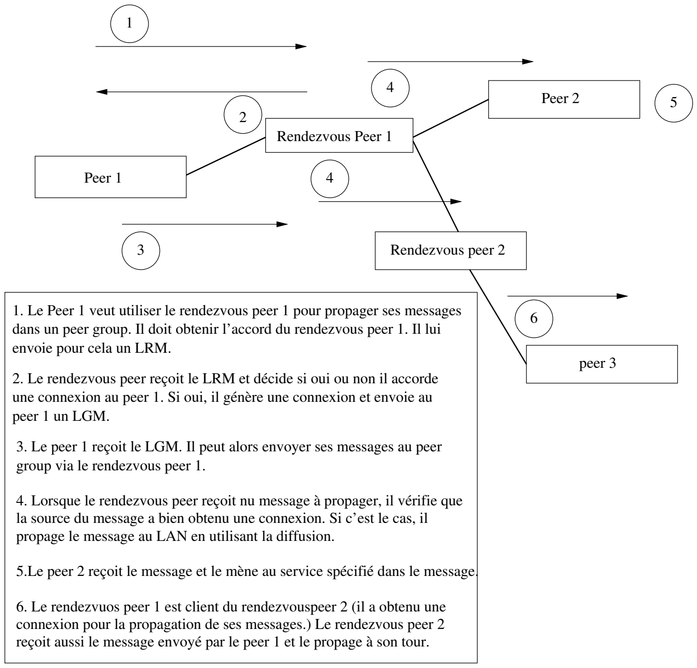

FIG. 2.2 – Echange de messages dans le Rendezvous Protocol

Avant de pouvoir envoyer un LRM à un rendezvous peer, il faut bien sur que le peer le connaisse, c'est-à-dire ait son advertisement. Il l'obtient en envoyant un **Discovery Query Message** (voir le Discovery Protocol.) Une fois le rendezvous peer connu, le peer peut envoyer un LRM avec le Endpoint Service (voir le Endpoint Routing Protocol.)

Une fois la connexion avec un rendezvous peer accordée, le peer pourra propager ses messages : des **Rendezvous Propagate Messages** (XML) qui contiennent :
- Un Id identifiant le message (exprimé par le temps en milliseconde écoulé depuis la propagation initiale du message.)
- Le service demandé par le peer
- les paramètres du service demandé
- une TTL pour éviter les loopbacks sur le réseau.

Un tel message est envoyé grâce à la méthode `propagate` de l'interface `RendezvousService`, récupéré de la même façon que précédemment le `EndpointService`. Toujours de la même façon que précédemment avec le Endpoint Routing Protocol, pour permettre à un peer de recevoir et traiter ces messages, il faut implémenter un `Endpointlistener` et le lancer en utilisant la méthode`addPropagateListener` du `RendezVousService`.

L'interface`RendezVousService` fournit en plus des interfaces permettant de gérer les connexions sur un rendezvous peers.

2.1.3 Le Peer Resolver Protocol (PRP)

Ici, le routage et la propagation via les rendezvous peers sont abstraits. Les messages utilisés dans le Peer Resolver Protocol seront automatiquement encapsulés dans des messages des protocoles précédents. Le Peer Resolver Protocol définit des règles pour envoyer une requête générique à un *handler* (équivalent du listener dans les protocoles précédents) qui traitera les requêtes et les réponses envoyées via le PRP. Le Resolver service implémente ce protocole, il s'appuie sur deux types de messages :
- Le **Resolver Query Message** : Un message XML pour envoyer des requêtes,
- Le **Resolver Response Message** : Un message XML pour envoyer des réponses à des requêtes.

Ces messages sont envoyés aux peers connus et propagés par les rendezvous peers. Si le handler d'un peer est capable de traiter la requête contenue dans le message, alors, il la traitera et enverra une réponse via la méthode `processQuery` de l'interface à implémenter `QueryHandler`. La requête (Resolver Query Message) est structuré en XML et doit contenir les éléments suivants :
- `<HandlerName>` : Nom unique spécifiant le nom du handler que doit invoquer le peer distant pour traiter la requête.
- `<SrcPeerId>` : Id du peer envoyant la requête (dans le format JXTA URN.)
- `<Query>` : La requête elle-même, sous la forme d'une chaîne de caractères. Il est de la responsabilité du Handler désigné de comprendre et d'extraire les composants de cette chaîne de caractères.

Une telle requête est implémentée par la classe JXTA `ResolverQuery`. La réponse doit contenir les éléments suivants :
- `<HandlerName>` : Nom unique spécifiant le nom du Handler que doit invoquer le peer distant pour traiter la réponse.
- `<Response>` : La réponse elle-même, sous la forme d'une chaîne de caractères. Il est de la responsabilité du Handler de savoir comment extraire les composants de cette chaîne de caractères.

Une telle réponse est implémentée par la classe JXTA `ResolverResponse`. Pour réaliser ce protocole, il faut :
- Définir un format de requête qui soit compréhensible par le Handler invoqué sur le peer distant pour résoudre la requête.
- Ecrire une classe implémentant la requête (à mettre dans le champs `<Query>` du Resolver Query Message) sous la forme d'une chaîne de caractères au format XML.
- Définir un format de réponse (à mettre dans le champs `<Response>` du Resolver Response Message) qui soit compréhensible par le Handler invoqué sur le peer distant pour traiter la réponse.

- Ecrire une classe implémentant la réponse sous la forme d'une chaîne de caractères au format XML.
- Implémenter l'interface JXTA `QueryHandler` en réécrivant les deux méthodes de cette interface : `processQuery` qui :
  - extrait la requête du Resolver Query Message reçue,
  - traite la requête,
  - génère une réponse qu'il encapsule dans un Resolver Response Message
  - envoie un Resolver Response Message en réponse.

  et `ProcessResponse` qui :
  - extrait la réponse du Resolver Response Message reçue,
  - traite la réponse
- Lors du lancement de la plate-forme, on lance deux entités de peers instanciant ces trois classes. Sur les deux peers, on doit :
  - créer une instance d'un `ResolverService` (en le récupérant depuis le peer group)
  - instancier notre handler (implémentant l'interface `QueryHandler`,
  - lancer notre handler en l'ajoutant à notre objet `ResolverService`.

  Maintenant, nos peers sont prêt à utiliser le Peer Resolver Protocol. Sur le peer initiateur de la requête :
  - créer une requête,
  - l'envoyer dans un Resolver Query Message, par la méthode `send` de l'objet`ResolverService`.

  Sur le peer destinataire :
  - recevra le Resolver Query Message,
  - en extraira la requête (champs `<Query>`),
  - traitera cette requête, génerera une réponse, l'encapsulera dans un Resolver Response Message et l'enverra, le tout par la méthode `processQuery` de son handler.

  Enfin, le peer initiateur de la requête :
  - recevra le Resolver Response Message,
  - extraira la réponse (champs `<Response>` et la traitera, le tout par la méthode `processResponse` de son Handler.

Les trois derniers protocoles sont des protocoles qui s'appuient sur le **Peer Resolver Protocol**, ils fonctionnent comme des instances de ce dernier, le service demandé est fixé.

2.1.4 Le Peer Discovery Protocol (PDP)

Le Peer Discovery Protocol définit les règles de communication pour la découverte d'advertisements des peers distant et pour la réponse à de telles demandes venues d'un peer distant. Ce protocole fonctionne comme une instance du Peer Resolver Protocol. Il s'appuie sur deux types de messages :

- Le **Discovery Query Message**, au format XML qui contient le type de l'advertisement à découvrir (0 = peer, 1 = peer group, 2 = autre.) D'autres éléments sont possibles comme le nombre d'éléments maximum, ou un ensemble de couples

Attribut/valeur donnant des critères que doivent respecter les éléments recherchés. Il est implémenté par la classe JXTA `DiscoveryQuery`.
- Le **Discovery Response Message**, au format XML qui contient les éléments du Discovery Query Message auquel il répond, en ajoutant l'éléments `<Response>` qui contient les advertisements correspondants aux critères demandés. Il est implémenté par la classe JXTA `DiscoveryResponse`.

Le mise en oeuvre de la communication est similaire à celle du Peer Resolver Protocol, mais :
- le Discovery Query Message remplace le Resolver Query Message
- le Discovery Response Message remplace le Resolver Response Message
- l'instance d'un DiscoveryService remplace celle du ResolverService
- l'implémentation de la méthode `discoveryEvent` de l'interface `DiscoveryListener` remplace notre implémentation des méthodes `processQuery` et`processResponse` du `QueryHandler`. La méthose `discoveryEvent` prend en paramètre un objet `DiscoveryEvent` qui encapsule un Discovery Query Message ou un Discovery Response Message. On les extrait en utilisant les méthodes `getQuery` et `getResponse`.

2.1.5 Le Pipe Binding Protocol (PBP)

Le Pipe Binding Protocol permet de créer un lien entre deux peers souhaitant utiliser un pipe pour communiquer. Il se repose sur deux types de messages :
- le Pipe Binding Query Message, pour la recherche d'un peer distant connecté à un pipe donné.
- le Pipe Binding Answer Message, pour indiquer si un peer a été trouvé ou non.

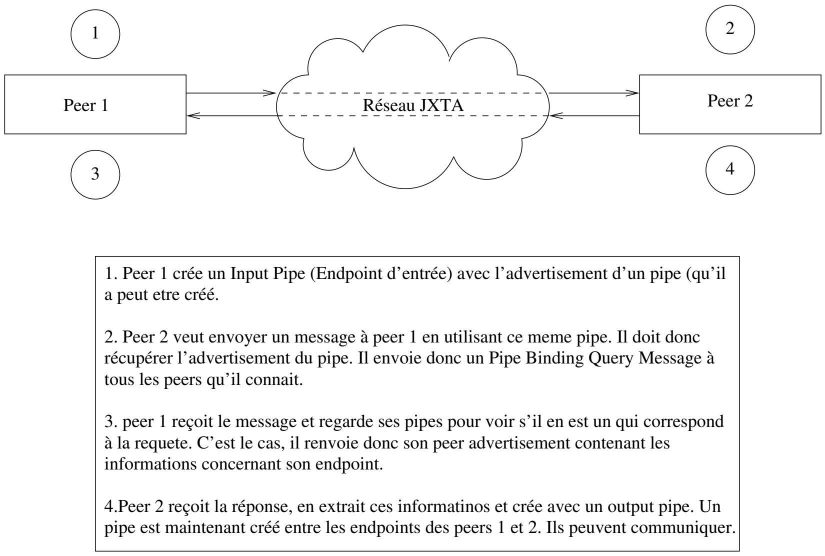

FIG. 2.3 – Etapes du Pipe Binding Protocol

JXTA met à dispositions des outils similaires à ceux proposées pour les protocoles précédents. Reprenons les deux peers du schéma ci-dessus.

2.1.6 Le Peer Information Protocol (PIP)

Le peer Information protocol permet d'obtenir des informations sur les peers distants (découverts au préalable.) dans le but d'améliorer les performances du réseau P2P. Les informations obtenues touchent essentiellement au trafic du réseau autour du peer distant, son niveau d'activité et d'utilisation de ses ressources. Comme précédement, le PIP se base sur deux messages :

- le **Peer Info Query Message**, message XML pour envoyer des requêtes pour obtenir des informations d'état sur un peer distant. Il doit contenir :
  - le champ `<SourcePid>` : format JXTA URN de l'ID du peer qui génère la requête,
  - le champ `<TargetPid>` : format JXTA URN de l'ID du peer duquel on cherche des informations,
- le **Peer Info Response Message**, message XML pour envoyer une réponse à ces requêtes. Il contient les champs `<SourcePid>` et `<TargetPid>` et contient les informations d'état de la façon suivante, les champs étant optionnels :
  - le champ `<uptime>` contient le temps écoulé en millisecondes depuis l'inscription du peer au peergroup,
  - le champ composé `<traffic>` contient des détails concernantle trafic du réseau sur lequel se trouve le peer. Il est de la forme suivante :

```
<traffic>
     <lastIncomingMessageAt> ... </lastIncomingMessageAt>
     <lastOutgoingMessageAt> ... </lastOutgoingMessageAt>
     <in>
          <transport endptaddr=''...''> ... </transport>
          <transport endptaddr=''...''> ... </transport>
          ...
     </in>
     <out>
          <transport endptaddr=''...''> ... </transport>
          <transport endptaddr=''...''> ... </transport>
          ...
     </out>
</traffic>
```

    les éléments `lastIncomingMessageAt` et `lastOutgoingMessageAt` contiennent respectivement le temps écoulé en millisecondes depuis l'arrivée d'un message sur le peer et le temps écoulé en millisecondes depuis l'envoi d'un message depuis le peer. Les éléments `<in>` et `<out>` contiennent le nombre d'octets actuellement traités par les endpoints spécifiés respectivement en entrée et en sortie.

Lorsqu'un peer envoie un Peer Info Query Message, tous les peers en relation avec lui le reçoivent. Ils testent alors si le champs `<targetPid>` contient l'ID du peer local. Si c'est le cas, ils construisent un Peer Info Response Message et l'envoie au `<sourcePid>`. Les outils proposés (listener et classes implémentant ces messages) sont similaires à ceux proposées pour les autres protocoles applicatifs.

2.1.7 Peergroups et services

Les services mettant en œuvre les protocoles du noyau existent au sein d'un peergroup. Comme on l'a vu, on récupère les services depuis une instance du peergroup dans lequel on souhaite travailler. Les services et les peergroups n'existent pas indépendamment. Par exemple pour mettre en œuvre le Discovery Service sur un peer, on doit écrire :

```
PeerGroup group = peerGroupFactory.newNetPeerGroup() ;
DiscoveryService discoService = group.getDiscoveryService() ;
```

Cela crée une instance du World Peer Group, puis d'un NetPeerGroup, qui contient l'ensemble des services correpondant aux protocoles du noyau de JXTA.

Chapitre 3

Démarrage d'une plate-forme JXTA

Pour lancer une plate-forme JXTA sur une machine, il faut :

- Télécharger une Java 2 standard edition SDK, version 1.3.1 ou supérieure depuis le site `http://java.sun.com/j2se`.
- L'installer en lançant le fichier binaire téléchargé.
- Paramètrer les variables d'environnement (PATH, ...) pour l'accès aux binaires java (java, javac, javah, ...)
- Télécharger les composants JXTA depuis le site `http://download.jxta.org/stablebuilds/index.html`
- Décompresser le fichier ZIP ainsi obtenu.
- Aller dans le répertoire shell ainsi créé.
- Lancer la plate-forme JXTA : lancer le fichier SH sous linux, ou le fichier EXE sous Windows), la fenêtre de configuration du peer apparait.
- Le premier onglet "basic" demande le nom du peer.
- l'onglet "advanced" permet la configuration TCP et HTTP. Pour utiliser le peer en local, remplir les champs adresses par `localhost` et ne pas s'occuper des paramètres HTTP. Si vous souhaitez communiquer avec un réseau local ou le réseau mondial, vous devez remplir tous les champs adresses avec l'adresse IP de votre machine (le port doit être différent pour TCP et HTTP.)
- Si vous souhaitez atteindre des peers en dehors du réseau local, il faut remplir le troisième onglet, cocher "use a relay" et télécharger les adresses des peers mis à disposition par `http://www.jxta.org`.
- Le dernier onglet concerne la sécurité, une fois rempli, un identifiant et un mot de passe seront demandés à chaque lancement de la plate-forme.
- Une fois le peer lancé, pour de nouveau configurer le peer, il faut entrerla commande `peerconfig`.

Bibliographie

[1] JXTA Doc. `http://platform.jxta.org/nonav/java/api/index.html`.

[2] Exemple Shell JXTA. `http://www-106.ibm.com/dfeveloperworks/java/library/j-p2pint2/`.

[3] Sing Li. *JXTA Peer-to-Peer Computing with Java.* Wrox, april 2002.

[4] Brendon J. Wilson. *JXTA*. New Riders, june 2002. Disponible sur le site `http://www.brendonwilson.com/projects/JXTA/index.shtml` ainsi que des exemples illustrant chacun des chapitres.