Développement d'une infrastructure de supervision pour la plateforme JXTA

Julien Braure

Développement d'une infrastructure de supervision pour la plateforme JXTA

Rapport de stage

Julien BRAURE

LORIA
Campus Scientifique
615, rue du jardin Botanique
54506 Vandoeuvre-les-Nancy Cedex
France

RÉSUMÉ. *L'architecture pair à pair propose dorénavant une alternative à l'architecture client/serveur et offre de nombreux avantages, comme la répartition du trafic et de la charge ou la résistance aux fautes. Pour pouvoir être déployés dans des contextes sensibles, où les niveaux de services doivent être garantis, les services pair à pair requièrent une infrastructure de gestion. Cette intégration commence par la réalisation d'un modèle de l'information permettant de décrire un réseau pair à pair dans son intégrité. Un tel modèle a été réalisé en utilisant le Common Information Model (CIM). Ce rapport explique comment ce modèle, déployé sur un réseau pair à pair de type JXTA, une plateforme générique en Java, permet de superviser et gérer efficacement ce dernier. De plus, j'ai réalisé une application de gestion permettant d'interagir avec l'infrastructure de gestion. Je la présente dans ce rapport. Ce travail s'intégre dans le cadre du projet SAFARI et sera utilisé par des partenaires industriels tels que Alcatel ou France Telecom.*

ABSTRACT. *Nowadays, networks are increasingly present in our life time. The peer to peer architecture is a real alternative offering several advantages against the client/server one, like load repartition or fault resistance. In order to be used in sensitive contexts, where QoS needs to be ensured, peer to peer services require a management infrastructure. This integration starts with the creation of a information model which allows the abstraction of an entire peer to peer plateform. This model has been designed as an extension of the Common Information Model (CIM). This report explains how this model, instantiated over the JXTA peer to peer framework, a generic Java platform, allows us to effectively manage it. Moreover, I designed a management application which interacts with the management infrastructure. I present it in this report. This work takes place in the SAFARI project and will be used by industrials partners as Alcatel or France Telecom.*

MOTS-CLÉS : *Pair à pair, Jxta, modèle de l'information, CIM, gestion de réseau*
KEYWORDS: *Peer to peer, Jxta, information model, CIM, network management*

1. Introduction

Dans les années 1950, l'informatique n'en était qu'à ses débuts. A force d'évolution dans les divers domaines que recouvre l'informatique (électronique, automatisme, ...) les réseaux sont de plus en plus performants et présents dans notre vie de tous les jours. L'accès à des configurations et connexions personnelles puissantes étant désormais possible, le concept pair à pair (P2P), jusque là peu répandu, concurrence maintenant efficacement le modèle client/serveur. Peu utilisé durant une longue période, la gestion de ce genre de réseau n'a pas été une priorité. De ce fait, les applications utilisant cette technologie sont dépourvues de systèmes de gestion et travaillent, pour la plupart, en ne garantissant pas de qualité de service (QoS).

Les industriels sont désormais fortement intéressés et demandeurs des nouvelles fonctionnalités que peut apporter le modèle P2P comme l'équilibre du trafic et de la charge du service exécuté, la tolérance aux fautes mais aussi le partage des coûts mis en oeuvre. Ces derniers ne peuvent décemment pas se passer de gestion sur leurs réseaux afin de garantir la qualité de service attendue.

Actuellement, les modèles de gestion existants s'adaptent mal aux propriétés des réseaux utilisant le modèle pair à pair tels que la dynamicité ou le passage à l'échelle. Il est donc impossible de réutiliser directement les travaux effectuées dans le domaine et de les appliquer sur un réseau pair à pair.

Un modèle d'information qui regroupe toutes les caractéristiques fonctionnelles, structurelles et topologiques d'un réseau P2P, a été réalisé par l'équipe MADYNES du LORIA [DOY 04a]. Ce modèle a été réalisé en se basant sur le modèle d'information CIM[1]. Afin de valider ce modèle, une instanciation de celui-ci a été réalisée [DOY 04b] sur Chord [STO 01], une infrastructure P2P de localisation de ressources. L'objectif de mon stage a été de valider ce modèle sur un réseau pair à pair de type JXTA et de réaliser une application de gestion s'y rapportant.

Ce rapport s'organise de la facon suivante : pour commencer, j'introduis les besoins de gestion qui existent dans le domaine des réseaux. Ensuite, je présente l'architecture pair à pair et les travaux déjà réalisés afin de proposer une infrastructure de gestion pour ce type de réseau. Je poursuis avec ma contribution, à savoir le déploiement d'une infrastructure de gestion sur la plateforme pair à pair JXTA. Je serais amené à détailler les différentes technologies utilisées. Par la suite, j'explique les mécanismes mis en oeuvre afin d'obtenir une vue virtuelle de la plateforme JXTA. Finalement je présenterais les fonctionnalités de mon application graphique permettant de gérer la plateforme. Je termine par une conclusion sur la travail que j'ai réalisé durant ce stage.

1. Common Information Model

2. Cadre de travail

Dans cette partie, je présente le LORIA, qui m'a accueilli pendant la durée de mon stage, ainsi que l'équipe de recherche que j'ai intégrée.

2.1. *Le LORIA*

Le LORIA[2] est une Unité Mixte de Recherche[3] commune à plusieurs établissements :

– le Centre National de Recherche Scientifique (CNRS) ;

– l'Institut National Polytechnique de Lorraine (INPL) ;

– l'Institut National de Recherche en Informatique et en Automatique (INRIA) ;

– l'Université Henri Poincaré, Nancy 1 (UHP) ;

– et l'Université Nancy 2.

La création de cette unité a été officialisée le 19 décembre 1997 par la signature du contrat quadriennal avec le Ministère de l'Education Nationale, de la Recherche et de la Technologie et par une convention entre les cinq partenaires. Cette unité, renouvelée en 2001, succède ainsi au CRIN[4], et associe les équipes communes entre celui-ci et l'Unité de Recherche INRIA Lorraine.

Le LORIA est un Laboratoire de plus de 450 personnes composées d'un tiers de chercheurs et enseignants-chercheurs, un tiers de doctorants et post doctorants, le reste étant composé d'ingénieurs, de techniciens et de personnels administratifs. Ce personnel est organisé en équipes de recherche et services de soutien à la recherche. En outre, chaque année, une trentaine de chercheurs étrangers sont invités, des coopérations internationales avec des pays des cinq continents sont organisées et une quarantaine de contrats industriels sont signés.

Le LORIA effectue plusieurs missions. Les principales sont la recherche fondamentale et appliquée au niveau international dans le domaine des Sciences et Technologies de l'Information et de la Communication, la formation par la recherche en partenariat avec les Universités lorraines ainsi que le transfert technologique par le biais de partenariats industriels et par l'aide à la création d'entreprises.

2. Laboratoire Lorrain de Recherche en Informatique et ses Applications
3. UMR 7503
4. Centre de Recherche en Informatique de Nancy

2.2. *L'équipe MADYNES et la gestion de réseaux*

L'équipe de recherche MADYNES[5] vise la conception, la validation et la mise en oeuvre de nouveaux paradigmes et architectures de supervision et de contrôle capables de maîtriser la dynamicité croissante des services et résistantes au facteur d'échelle induit par l'Internet ubiquitaire. En d'autres termes, son activité de recherche concerne principalement la gestion des réseaux et services, domaine que je présente ci-après.

Le gestion d'un réseau regroupe l'ensemble des mécanismes et des activités qui visent maintenir un réseau ainsi que les services qu'il fournit dans un état de fonctionnement satisfaisant.

Le premier but d'une infrastructure de gestion est de connaître l'état du réseau à un moment donné. Cet état est donné par la connaissance des différents acteurs ainsi que les relations qu'ils entretiennent. Une connaissance statique et ponctuelle n'est pas suffisante afin de prendre en compte tous les éléments qui composent un réseau actif. En effet, il est intéréssant de connaître l'existence d'une relation entre deux éléments d'un réseau, mais il l'est encore plus de savoir combien de messages sont échangés grâce à cette liaison. Pour ce faire, une des activités de la gestion de réseau concerne la surveillance, celle-ci permettant ensuite d'adapter les ressources qu'utilise le réseau. les performances du réseau peuvent diminuer si celles-ci sont mal réparties.

Une infrastructure de gestion permet également de detecter l'apparition de problèmes sur un réseau. Leurs résolutions se traduisent alors par des actions de gestion à caractère automatique ou manuel.

La conception d'une infrastructure de gestion est décomposée par la défintion de quatre modèles :

– un modèle de l'information : qui permet de modéliser l'architecture réseau afin de rendre compte de tous les éléments qui composent le réseau ainsi que leurs interactions ;

– un modèle d'organisation : qui définit les différentes entités qui composent l'infrastructure de gestion et la manière dont elles s'organisent. Une organisation est très répandue dans le domaine de la gestion de réseau : le modèle d'organisation gestionnaire/agents. Ce modèle est utilisé, par exemple, par SNMP[6][CAS 90] ;

– un modèle de communication : définissant le manière dont les différents acteurs de l'infrastructure de gestion communiquent entre eux ;

– et un modèle fonctionnel : donnant le but de l'infrastructure de gestion. Généralement on cherche à offrir un système de surveillance et de supervision pouvant inclure différentes aires fonctionnelles de gestion qui sont la gestion

5. MAnagement of DYnamic NEtworks and Services
6. Simple Network Management Protocol

des fautes, de la configuration, de la facturation, de la performance et de la sécurité.

3. Réseaux pair à pair et gestion

Les différents systèmes informatiques sont classés suivant la perspective de la figure 1 [MIL 02] :

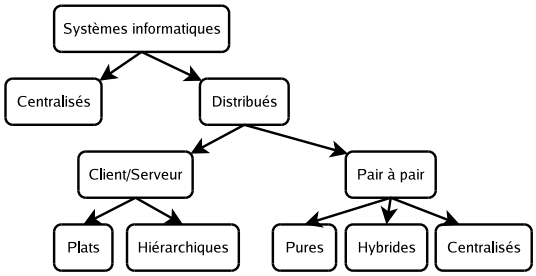

Figure 1. *Taxonomie des systèmes informatiques*

Nous remarquons que les systèmes distribués sont divisés en deux grandes familles :

– Les réseaux ayant une architecture Client/Serveur ;

– Les réseaux suivant l'architecture pair à pair.

L'architecture client/serveur étant très connue, voyons dans cette partie les différences apportées par l'architecture pair à pair.

3.1. *Présentation des réseaux pair à pair*

Le terme "pair à pair", ou "peer-to-peer", fait référence à une classe de systèmes et d'applications qui utilisent des ressources distribuées afin de réaliser une fonction d'une manière décentralisée. Son principal objectif est d'optimiser au maximum le partage des ressources sur le réseau. Pour atteindre ce but, le pair à pair ne fait plus de distinction entre les éléments qui publient des ressources sur le réseau et ceux qui les consomment. Comme le montre la figure 2, plus de client ni de serveurs : rien que des pairs [MIL 02].

3.1.1. *Client/serveur contre pair à pair*

Le modèle client/serveur ne parvient pas à tirer le maximum de potentiel du réseau comme l'entend le concept du pair à pair, car les seules ressources publiées sur le réseau sont celles des serveurs, ceux-ci étant équipés de services

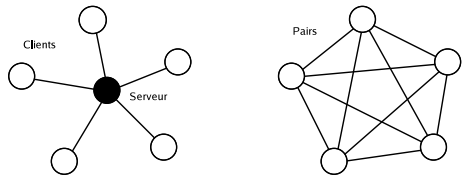

Figure 2. *Différence entre les architectures client/serveur et pair à pair*

d'infrastructure nécessaires. Dans ce genre de réseau, le trafic est centralisé ce qui implique des goulots d'étranglement au niveau du serveur. Aussi, cette architecture est fragile en terme de résistance aux fautes puisque si le serveur tombe en panne, le service offert est inutilisable. Enoutre, l'accès à des configurations puissantes étant de plus en plus facile, on commencer à envisager une utilisation des ressources que le client n'utilise pas. L'architecture pair à pair permet de répondre à ses nouveaux besoins en matière d'échange d'informations.

Dans un réseau pair à pair, il n'y a plus aucune différence entre clients et serveurs, ce sont tous des pairs, chaque pair publiant et consommant des ressources. Pourtant la famille des réseaux pair à pair peut se décomposer en trois sous-ensemble, comme le montre la figure 1 :

– Les réseaux pair à pair purs : aucune différenciation n'est faite entre les pairs, ils sont tous égaux. Gnutella [KAN 01], Chord [STO 01] et Pastry [ROW 01] sont des exemples de réseau pair à pair purs ;

– Les réseaux pair à pair hybrides : dans cette architecture, certains pairs présentent des fonctionnalités particulières, mais restent avant tout des pairs. Kazaa [7] et Jxta [WIL 02] sont des réseaux pair à pair hybrides ;

– Les réseaux pair à pair centralisés : dans lesquels la localisation des ressources est d'abord faite par requête à un serveur. Ensuite l'échange est réalisé de manière décentralisée et autonome entre les pairs. Napster [SHI 01] est un exemple de réseau pair à pair centralisé.

3.1.2. *Propriétés d'un réseau pair à pair*

De nature décentralisée, l'architecture pair à pair présente des propriétés intéressantes détaillées ici [MIL 02] :

– l'équilibre du trafic et de la charge : c'est peut être la principale propriété d'un réseau pair à pair. Comme il n'existe plus de serveur central, on évite ainsi les goulots d'étranglement ;

7. http ://www.zeropaid.com

– la résistance aux fautes : le fait de ne plus avoir de serveur rend le système plus résistant aux pannes que pourrait subir le serveur et qui rendent le service offert inutilisable ;

– le passage à l'échelle : grâce à la nature décentralisée du modèle pair à pair, le réseau devient plus robuste lors du passage à l'échelle ;

– l'autonomie : puisqu'il est possible de consommer des services sans pour autant s'enregistrer auprès d'un serveur centralisé ;

– et la répartition et réduction des coûts d'équipement : les applications pair à pair étant souvent deployées à l'échelle de l'Internet, les utilisateurs appartiennent souvent à des domaines différents (entreprises, administrations, particuliers, ...). Puisque la fonction de serveur est distribuée sur l'ensemble des pairs, il n'y a plus besoin de serveur puissant pour faire face aux requêtes.

3.2. *Modéliser le réseau pour le gérer*

Les applications actuelles utilisant l'architecture pair à pair sont dépourvues de système de gestion, alors que des applications à garantie de service, par exemple dans le cas d'une contrainte temps réel, requièrent une surveillance et une supervision. Les modèles architecturaux de gestion existants s'adaptnt mal au modèle pair à pair [DOY 05]. De nature centralisée, ils ne peuvent pas s'interfacer avec le caractère dynamique de ce modèle. Dans ce but, MADYNES travaille sur la proposition d'un modèle de gestion adapté au pair à pair. Dans cet section, je présente le formalisme utilisé pour créer un modèle de l'information de gestion ainsi que le modèle créé par l'équipe MADYNES.

3.2.1. *Le formalisme CIM*

CIM[8] est un modèle de l'information de gestion proposé par le DMTF[9]. Il fournit des définitions et une structure de données utilisant le paradigme orienté-objet. L'approche est basée sur l'élaboration d'un cadre conceptuel pour la description de l'environnement géré que l'on peut ensuite étendre à un contexte spécifique. Ce cadre se compose de trois couches [FES 03, BUM 00] :

– Un modèle noyau[10] [Dis 00] : composé des concepts applicables à tous les domaines de gestion. Il comprend un ensemble de classes, d'associations et de propriétés qui fournissent un vocabulaire de base pour définir des éléments gérés. Les éléments gérés sont décrits dans une composante logique ou une composante physique.

– Un modèle commun[11] [Dis 03] : regroupant les concepts généraux de chacun des domaines de gestion et servant de base extensible : Systèmes, Appli-

8. Common Information Model, http ://www.dmtf.org/standards/cim
9. Dynamic Management Task Force, http ://www.dmtf.org
10. Core Model
11. Common Models

cations, Dispositifs, Utilisateurs, Réseaux, etc. Le modèle de l'information est assez spécifique afin de fournir les bases pour le développement d'applications de gestion ;

– Des d'extensions[12] : qui représentent les extensions spécifiques à des environnements, comme par exemple les systèmes d'exploitation.

3.2.2. *Le modèle P2P créé par MADYNES*

Voyons maintenant le modèle qui permet d'obtenir une représentation virtuelle d'une plateforme pair à pair. Le formalise CIM a été utilisé afin de créer ce modèle qui en est une extension. Ce modèle, proposé par l'équipe MADYNES, est générique à toute application pair à pair et est composé de cinq sous-modèle [DOY 04a]. Les interactions de ces sous-modèles sont représentées figure 3.

Les sous-modèles sont :

– le modèle *Peer and Community* : qui formalise la notion de pairs, communauté, et de topologie virtuelle ;

– le modèle *Communication* : qui fournit les informations sur la manière dont les pairs communiquent ;

– le modèle *Resources* : qui contient les informations relatives aux ressources qu'un pair partage avec les autres dans le cadre d'un service ;

– le modèle *P2P Services* : qui représente un service fourni à la communauté sur le réseau pair à pair ;

– et le modèle *Routing and Forwarding* : qui englobe les services et les tables de routage contenus dans les pairs.

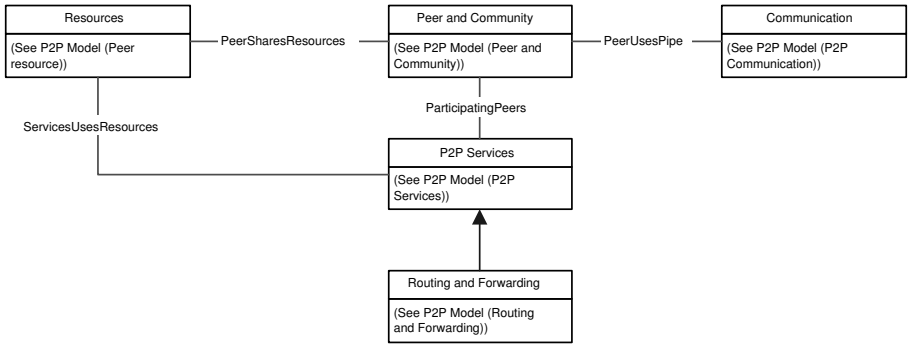

Figure 3. *Modèle P2P de l'équipe MADYNES*

Je vais détailler le sous-modèle *Peer and Community*. La figure 4 le présente.

12. Extensions shemes

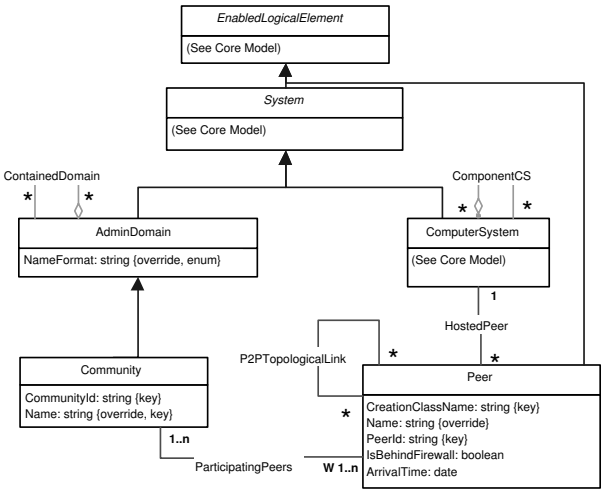

Figure 4. *Sous-modèle Peer and Community*

Il existe trois classes et trois associations dans ce modèle : *Community*, *Peer*, et *ComputerSystem* pour les classes, *ParticipatingPeer*, *HostedPeer*, et *TopologicalLink* pour les associations. Qu'elles soient classes ou associations, elles héritent toutes de classes fournies par les modèles noyau ou commun du formalise CIM.

Ces trois classes et ces trois assocations sont suffisantes pour formaliser la population de pairs, leur appartenance à une communauté et la manière dont une communauté s'organise. Les associations permettent de relier les instances des différentes classes entre elles afin de savoir quel pair participe à quelle communauté et qui l'héberge. L'association *TopologicalLink* permet de représenter une liaison topologique entre deux pairs. Cette laison ne possède aucune sémantique particulière, elle dépend de l'implantation.

Ensuite chaque classe contient des attributs spécifiques. Dans le cas de la classe *Community* par exemple, le nom de la comunauté est contenu dans l'attribut *Name*, et son numéro d'identification dans l'attribut *CommunityId*. On remarque la présente d'un modificateur *key* sur l'attribut *CommunityId*. Cela signifie que la concaténation de tous les attribut clés doit être unique dans tout le modèle, ceci afin d'assurer l'unicité de l'instance représentant un objet, ici une communauté. Le modificateur *override*, présent sur l'attribut *Name*, permet de surcharger l'attribut homonyme hérité d'une classe mère. Ici l'attribut *Name*, hérité de la classe *ManagedSystemElement* provenant du modèle noyau de CIM, est redéfini pour maintenant correspondre au nom de la communauté.

Afin de décrire le modèle P2P, l'équipe MADYNES a utilisé le langage de description MOF[13], spécifique au formalisme CIM. Ce langage permet de rendre compte de tous les détails apportés par le formalisme CIM sous forme de fichier texte. Le listing 1 propose un extrait du fichier MOF, qui concerne la classe *Community*. On retrouve bien le nom de la classe *Community* ainsi que la classe quelle étend, *AdminDomain*. S'ajoutent en entête de la classe, la version de la classe ainsi qu'une description de celle-ci. Les deux attributs *CommunityId* et *Name* sont présents ainsi que leur type, et une description pour chaque attribut permet de détailler leur sémantique. Les modificateurs *Key* et *Override* sont placés en entête des attributs. On remarque la présence du modificateur *MaxLen (256)* dans les entêtes, ce qui permet de donner une contrainte sur la longueur de la chaîne de caractère.

Listing 1 – La classe Community du modèle Peer and Community décrite en langage MOF

```
[Version ("1.0.0"), Description (
    "This class represents a community in a P2P application.") ]
class P2P_Community : CIM_AdminDomain {

    [Key, MaxLen (256), Description (
        "The identifier of the considered community in the context of "
        "the executing application.") ]
    string CommunityId;

    [Override ("Name"), MaxLen (256), Description (
        "The Name property uniquely identifies the ServiceAccessPoint "
        "and provides an indication of the functionality that is "
        "managed.  This functionality is described in more detail in "
        "the object's Description property.") ]
    string Name;
};
```

3.2.3. *Validation du modèle*

Afin de valider ce modèle qui formalise les réseaux pair à pair, le déploiement de celui-ci a été envisagé sur plusieurs plateformes pair à pair. Le première instanciation a été réalisée dans l'équipe sur une plateflorme de type Pastry[ROW 01]. Le même travail m'a été demandé sur la plateforme JXTA.

3.3. *Présentation d'une plateforme pair à pair en Java : JXTA*

JXTA[14] [OAK 02], de l'anglais « juxtapose », est un projet *open source* lancé par Sun Microsystems en avril 2001. Le but de JXTA est de pouvoir interconnecter n'importe quel système sur n'importe quel réseau. Par définition, il pourra interconnecter un ordinateur avec un PDA, ou un téléphone portable, etc. En mars 2004, le projet JXTA regroupait plus de 17 000 développeurs, et plus d'une centaine de projets adoptaient cette nouvelle technologie. Sun

13. Managed Object Format
14. http ://www.jxta.org

cherche à l'imposer comme un standard du pair à pair. De plus en plus de sociétés utilisent JXTA pour leurs projets.

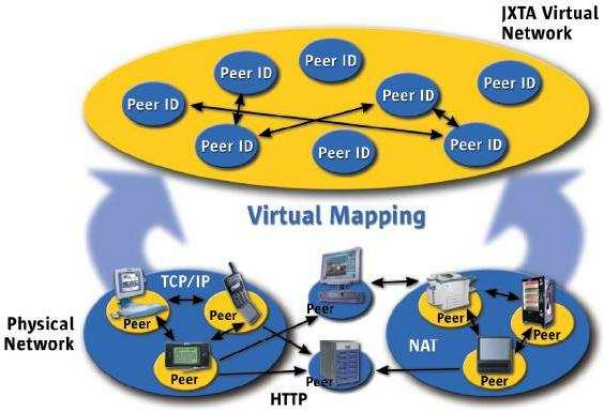

Figure 5. *Le réseau virtuel JXTA*

Comme d'autres architectures pair à pair, JXTA crée un réseau virtuel, appelé *overlay*, au dessus des autres réseaux issus du monde IP ou non, permettant aux pairs d'interagir et de s'organiser indépendamment de leur mode de connexion réseau. Un exemple d'*overlay* est présenté figure 5.

JXTA appartient à la famille des réseaux pair à pair hybride. En effet, certains pairs sont spécialisés afin d'offrir un service de découverte à la communauté à laquelle ils appartiennent, ce sont les pairs de *Rendezvous*. D'autres jouent un rôle de routage, ce sont les pairs relais.

Plusieurs implantations de JXTA existent, la plus avancée est celle qui est programmée en langage Java, donc multi-plateforme. En outre, elle utilise XML[15] pour l'échange des données ce qui permet une grande interopérabilité.

3.3.1. *Les concepts de JXTA*

JXTA repose sur plusieurs concepts que je vais détailler ici [Sun05]. Ces différents concepts sont :

- les pairs ;
- les communautés ;
- les annonces ;
- les *pipes* ;
- et les *handlers*.

15. eXtensible Markup Language

Les pairs sont les éléments de base de JXTA. Il en existe trois types différents :

– les pairs simples[16] : qui sont les plus basiques. Ils permettent de consommer et de proposer des services à la communauté pair à pair. Ils sont dépourvus de responsabilité envers le réseau ;

– les pairs de rendezvous[17] : proposent, en plus du rôle de pair simple, un service de localisation qui permet aux autres pairs du réseau de découvrir les pairs et autres ressources du réseau, comme par exemple un service. Les performances des pairs de rendezvous est accrue grâce à la mise en mémoire tampon des informations, mais aussi en partageant ses connaissances avec d'autres pairs de rendezvous ;

– et les pairs routers[18] : proposent, en plus du rôle de pair simple, un service de communication afin d'atteindre des pairs se trouvant, par exemple, derrière un pare feu ou un serveur NAT. L'utilisation de plusieurs pairs routers peut s'avérer utile afin de permettre la communication entre deux pairs se trouvant sur des réseaux incompatibles en terme de transport.

Chaque pair sur une plateforme JXTA peut agir comme l'un ou plusieurs des types précédents, chacun définissant un ensemble de responsabilités afin d'entretenir le réseau pair à pair auquel il appartient. Notons qu'un pair peut endosser le triple rôle de pair simple, pair de rendezvous et pair router.

La notion de communauté est prise en charge par JXTA qui la formalise sous forme de *PeerGroup*. Un *PeerGroup* permet donc de regrouper plusieurs pairs ayant un centre d'intérêt commun ou proposant un service à ses membres qui ne serait pas accessible par les autres pairs du réseau. Un pair peut appartenir à plusieurs *PeerGroups* à la fois et peut constituer un *PeerGroup* à lui tout seul, mais lors de sa création, un pair rejoint le groupe *NetPeerGroup* qui agit comme un groupe par défaut.

Sur un réseau JXTA, toutes les ressources, incluant les pairs, les communautés, les services, etc., sont représentés grâce à une annonce appelée *advertisement*. Ces annonces sont structurés grâce à XML, et sont normalisées pour toutes les ressources faisant partie du noyau JXTA. A chaque ressource est attribué un identifiant unique contenu dans son annonce. Ce numéro d'identification, ou *ID*, rend la ressource indépendante de sa localisation sur le réseau.

Les *pipes* sont un des moyens de communication de JXTA. Ce sont des canaux de communication virtuels utilisés pour échanger des messages entre les pairs. Ces messages étant utilisés par les services ou les applications présents sur le pair. Les *pipes* proposent une communication unidirectionnelle et sont composés de deux extrémités. D'un côté le *Input Pipe* permet de recevoir des

16. *Edge peers*
17. *Rendezvous peers*
18. *Relay peers*

données, et de l'autre, le *Output pipe* permet de les envoyer. Il existe deux types de *pipes*, illustrés sur la figure 6 :

– le *pipe* point à point : qui relie un *Output pipe* à un *Input Pipe* ;

– et le *pipe* propagé : qui relie un *Output pipe* à plusieurs *Input Pipes*.

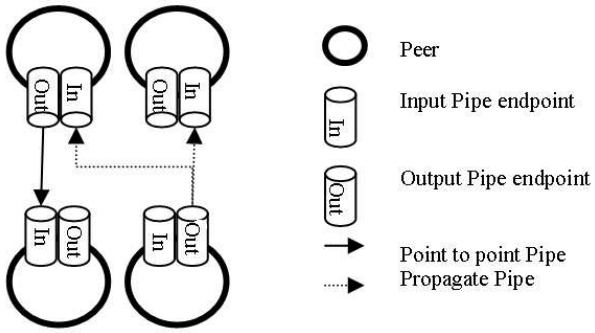

Figure 6. *Extrait de [OAK 02]. Exemples de pipes*

Enfin, les *handlers* permettent une communication plus large, au niveau de la communauté. En effet, tout pair ayant chargé un *handler* peut émettre des messages qui seront propagés dans toute la communauté. Seuls les pairs ayant le même *handler* prendront en charge ces messages et y répondront si besoin. Ceux-ci sont largement utilisés dans les protocoles internes de JXTA. Il est néanmoins possible de définir ses propres *handlers*.

3.3.2. *Les mécanismes internes de JXTA*

Les concepts précédents sont mis en oeuvre à travers six protocoles qui s'organisent selon la pile illustrée sur la figure 7. Ces protocoles sont divisés en deux familles [WIL 02] :

– Les protocoles de spécification du noyau :

- Endpoint Routing Protocol (ERP) : permet à un pair de découvrir une route vers un autre pair afin de lui envoyer un message ;

- Peer Resolver Protocol (PRP) : définit comment un pair peut émettre une requête générique et recevoir les réponses.

– Les protocoles de services standards :

- Rendezvous Protocol (RVP) : propose un service de propagation. Dans une communauté, chaque pair doit se connecter à un pair de rendezvous. Ainsi, quand un pair veut emettre un message, celui le transmet à son pair de rendez-vous auquel il est connecté. Celui-ci s'occupe ensuite de propager le message à ses divers abonnés ainsi qu'aux autres pairs de rendezvous qu'il connait ;

- Peer Discovery Protocol (PDP) : est utilisé par les pairs afin de publier et de découvrir les différentes annonces de la communauté. Ce protocole fait appel à PRP afin d'envoyer et propager les requêtes de découverte ;

- Peer Information Protocol (PIP) : est un protocole orienté gestion qui permet d'obtenir des informations d'autres pairs, comme l'état, la charge du trafic, etc. Ce protocole fait appel au PRP afin d'envoyer et propager les requêtes d'informations. L'implantation de ce protocole n'est pas encore terminée ;

- Pipe Binding Protocol (PBP) : permettant à un pair d'établir un canal de transmission virtuel entre plusieurs pairs. Il est utilisé pour relier deux extrémités d'un *pipe* (ou plus dans le cas d'un *pipe* propagé) à des *endpoints* physiques, par exemple un port TCP[19].

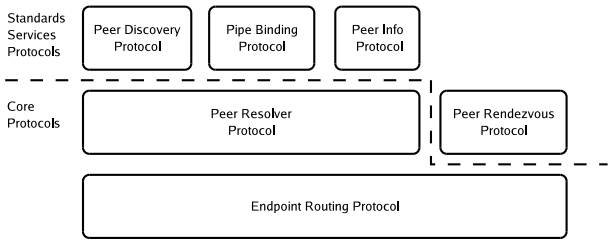

Figure 7. *Modèle architectural des protocoles JXTA*

4. Gestion de la plateforme JXTA

Le but recherché dans la mise en place de l'infrastructure de gestion prséntée dans la figure 8. Chaque pair héberge un agent. Un gestionnaire agrége les données de gestion proposées par les différents agents. Enfin, l'application de gestion utilise le gestionnaire pour obtenir les informations agrégées.

Cette infrastructure de gestion va permettre de voir le réseau, de le surveiller et d'interagir avec ce dernier. Il faut donc pouvoir visualiser la topologie globale, pouvoir récupérer des données de gestion, et aussi arrêter ou démarrer un service à partir du modèle. Le modèle fonctionnel de mon infrastructure de gestion étant définit, je vais maintenant aborder les trois autres points que composent une infrastructure de gestion : le modèle de l'information, le modèle d'organisation et le modèle de communication.

19. Transmission Control Protocol

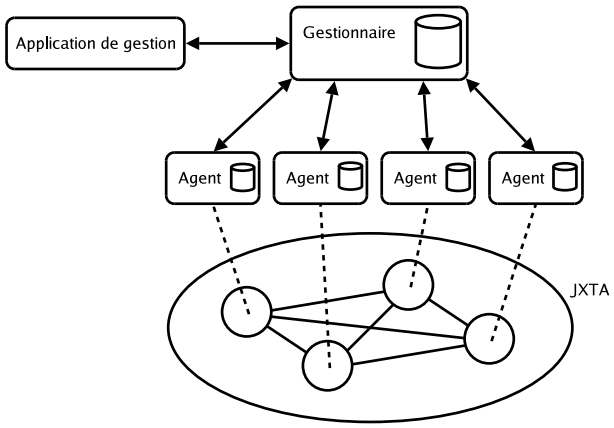

Figure 8. *Vue d'ensemble de l'infrastructure de gestion*

4.1. *Modèle de l'information : Modèle P2P de l'équipe MADYNES*

Afin de modéliser la plateforme JXTA, je me base sur le modèle pair à pair réalisé par l'équipe MADYNES du LORIA.

4.1.1. *Extension du modèle pour JXTA*

Le modèle pair à pair de l'équipe MADYNES formalise uniquement les caractéristiques génériques des réseaux pair à pair. Il ne suffit donc pas à formaliser tous les concepts spécifiques à la plateforme JXTA. Afin de rendre compte de tous les détails d'une telle plateforme, il s'avère obligatoire d'étendre le modèle actuel. Ce travail a été amorcé par Rizi Mohanty, stagiaire au sein de l'équipe [MOH 04].

L'extension pour JXTA permet de préciser le modèle générique. Comme le montre la figure 9, la classe *Peer* est étendue en une classe *JxtaPeer* qui contient un attribut supplémentaire *IsManageable*. Cet attribut permettra de renseigner la présence, ou non, d'un agent sur le pair. D'un autre coté, JXTA appartenant aux réseaux pair à pair hybrides, certains de ses pairs peuvent être de type *Rendezvous* ou *Relay*. Cette précision est contenue dans la classe *JxtaParticipatingPeer* qui étend la classe *ParticipatingPeers*. Pour finir, l'association *RendezvousConnection* étend *TopologicalLink*. J'utilise celle-ci pour modèliser l'abonnement d'un pair à un pair de rendezvous.

4.1.2. *Instanciation du modèle*

L'utilisation d'une librairie Java permettant d'interagir avec des instances d'objets, correspondants au classes CIM, est requise. Pour cela j'utilise la spé-

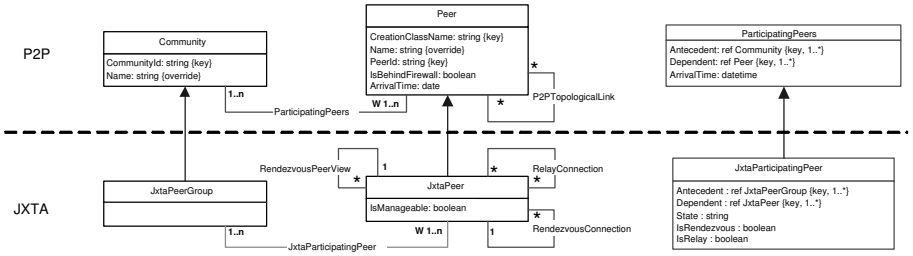

Figure 9. *Extension du sous-modèle Peer and Community pour le spécialiser pour la plateforme JXTA*

cification JMX[20] [JAS 02], proposée par Sun Microsystems dans un but de gestion des applications Java. Le choix de JMX permet de réaliser une infrastructure de gestion intégrée, ce qui veut dire implantée à partir d'un seul langage de programmation.

Cette spécification propose l'accès à un serveur de MBeans. Ces MBeans, implantés sous forme d'inferaces Java, permettent l'interaction avec des instances d'objets Java contenu dans la machine virtuelle. A l'intérieur de ce serveur chaque objet, MBean, est identifié grâce à un nom unique. Ce nom, appelé *ObjectName*, est composé d'un nom de domaine ainsi qu'une suite de champs composés d'une clé et d'une valeur, par exemple :

```
domaine:clé1=valeur1,clé2=valeur2,...
```

La spécification JMX définit quatre types de Mbeans :

– les MBeans standard (*Standards MBeans*) : exposent leurs opérations et attributs administrables en implémentant une interface Java contenant des méthodes qui respectent les modèle de nommage JavaBeans standard pour les méthodes de réglage (*setter*) et d'obtention (*getter*) ;

– les MBeans dynamiques (*Dynamics MBeans*) :exposent à l'éxecution les opérations et attributs administrables ;

– les MBeans modèles (*Models MBeans*) :étendent les MBeans dynamiques et font office de proxy pour les objets réels à gérer ;

– les MBeans ouverts (*Opens MBeans*) : étendent les MBeans dynamiques pour afficher leurs attributs et opérations à l'aide de métadonnées constituées d'un ensemble prédéfini de types Java standard.

Chaque type correspond à un usage spécifique et donc dépend de la manière dont on veut effectuer la gestion. Dans le cadre du projet abordé durant mon

20. Java Management eXtensions

stage, j'utiliserais les MBeans de type standard, ceux-ci étant suffisant et correspondant aux opérations que je souhaite effectuer sur les objets que je suis amené à gèrer.

Notons que comme JMX n'est qu'une spécification, il existe donc plusieurs implantations :

– la RI[21] proposée par Sun Microsystems ;

– MX4J[22], une autre implémentation. Un de ces principaux avantages repose sur l'utilisation de mémoires tampons.

Dans mon projet, j'ai utilisé la RI de fournie par Sun Microsystems.

La spécification JMX correspond très bien à mes besoins puisque chaque objet du modèle P2P peut être représenté par un MBean, ce dernier étant rendu identifiable grâce à son *ObjectName*. Nous utiliserons donc la suite de couple clé/valeur afin de rendre différenciable les instances des classes du modèle P2P, les clés de l'*ObjectName* portant le nom des clés issues de la classe CIM que le MBean représente, et les valeurs correspondantes de l'*ObjectName* contenant la valeur de l'attribut de classe CIM correspondante.

L'instanciation du modèle P2P soulève un premier problème : la plateforme JXTA est implémentée en Java, CIM n'est pas un formalise Java. J'ai donc eu besoin d'un outil pour implanter le formalise CIM en Java sous forme de MBeans. Ceci fut réalisé grâce à l'outil MOF2MBean. Cet outil permet de générer le squelette des classes Java permettant de représenter l'objet sous fourme de MBean. En effet comme mentionné précédemment, le modèle de l'information CIM est décrit grâce au langage MOF. Il faut noter que l'outil MOF2MBean, n'étant pas entièrement conforme à la spécification MOF, ne permet pas de rendre compte de toutes les subtilités du formalisme CIM.

Un extrait de la classe *Community* générée par MOF2MBean est présenté dans le listing 2. Cette classe abstraite Java implante les caractéristiques de la classes CIM et est conforme à la spécification JMX. On remarque, lignes 11 et 20, les deux méthodes permettant la lecture des attributs de l'instance. La lecture des attributs hérités étant pris en charges par les classes mères. Ces méthodes sous-traitent le travail à des méthodes abstraites, lignes 31 et 33, puis vérifient si le résultat est conforme aux contraintes listées dans le fichier MOF. Dans le cas contraire, une *Exception* Java est levée. La méthode *checkKeys*, ligne 23, permet de vérifier la présence des attributs clés CIM dans la composition de l'*ObjectName*. Cette méthode est invoquée lors de l'enregistrement de l'instance dans le serveur de MBean.

21. Reference Implementation, http ://www.jav.sun.com/products/JavaManagement/
22. Management eXtensions for Java, http ://mx4j.sourceforge.net/

Il suffit ensuite de créer une nouvelle classe qui étend cette classe abstraite afin de définir le constructeur, et les méthodes abstraites correspondants aux *setter* et/ou ,comme ici, aux *getter* de l'objet.

Listing 2 – Extrait de la classe *Community* générée à partir du fichier MOF grâce à l'outil MOF2MBean

```
1  import <...>;
2
3  public abstract class P2P_Community extends CIM_AdminDomainImpl
4          implements P2P_CommunityMBean, MBeanRegistration {
5
6  public P2P_Community() throws InvalidAttributeValueException {
7          super();
8          className = new String("P2P_Community");
9  }
10
11 public String getCommunityId() throws ImplementationException,
12         InvalidAttributeValueException {
13         String result = _do_getCommunityId();
14         if (result != null)
15                 if (result.length() > 256)
16                         throw new MaxLenConstraintViolationException(result.length(),256);
17         return result;
18 }
19
20 public String getName() throws ImplementationException,
21                 InvalidAttributeValueException {<...>}
22
23 protected void checkKeys(Vector keyList) throws MalformedObjectNameException {
24         String[] cimKeyList = {"CommunityId"};
25         try {
26                 removeMyKeys(keyList,cimKeyList);
27                 super.checkKeys(keyList);
28         } catch (MalformedObjectNameException e) {throw e;}
29 }
30
31 protected abstract String _do_getCommunityId() throws ImplementationException,
32         InvalidAttributeValueException;
33 protected abstract String _do_getName() throws ImplementationException,
34         InvalidAttributeValueException;
35 }
```

Je vais donc utiliser l'architecture JMX dans mon infrastructure de gestion. Les ressources gérées seront les instances des classes du modèle CIM accessibles grâce aux MBeans générés avec MOF2MBean. Le serveur de MBean permet de les présenter dans un but de gestion. On pourra venir les consulter grâce à un client RMI au travers du connecteur RMI. J'utiliserais également un adaptateur HTML me servant uniquement de vue de mon serveur de MBeans dans un but de débogage.

4.2. *Modèle d'organisation : Gestionnaire / Agent*

J'ai choisi d'organiser mon infrastructure de gestion suivant le modèle gestionnaire/agent. Voyons dans cette partie pourquoi et comment je la déploie suivant ce modèle.

4.2.1. *Intégration d'un agent dans un pair JXTA*

Afin de rendre compte de l'état actuel du pair qui l'héberge, un agent collecte et présente des informations spécifiques à cleui-ci. Suite à cette introspection, l'agent crée les instances adéquates du modèle de l'information de gestion pour le pair à pair dans le but de proposer une vue locale. Ces instances sont présentées dans un serveur de MBean et peuvent être consultées à partir d'un connecteur RMI. La figure 10 illustre un pair, l'agent que celui-ci héberge, ainsi que la vue locale proposée par l'agent.

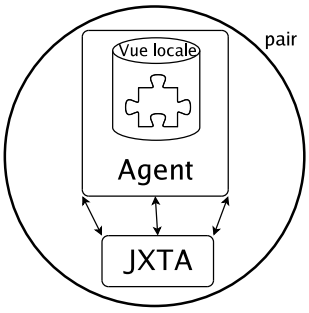

Figure 10. *Un pair, son agent, et la vue locale*

Les informations des gestion sont principalement récupérées grâce au projet MMP[23] qui propose une instrumentation de la plateforme JXTA. Cependant, ce projet n'étant apparement plus en développement pour le moment, j'ai été amené à définir, d'une manière similaire, plusieurs données de gestion dont j'avais besoin. Par exemple, le service de *Rendezvous* est très bien instrumenté par MMP, par contre aucun travail n'a été réalisé en ce qui concerne le service de *Pipes*. J'ai aussi défini des données de gestion permettant de connaître à tout instant à quels *PeerGroups* le pair est abonné. Pour cela, j'ai du assimiler et modifier le code source de la plateforme JXTA. De ce fait l'utilisation d'un agent sur un pair nécessite l'utilisation de la plateforme que j'ai modifiée.

Dans l'état actuel, les informations proposées par l'agent sont rafraichies périodiquement. Cependant le taux de rafraichissement peut être modifié à tout moment, et ce à la volée. Dans tous les cas, une latence est inévitable, ce qui rend la vue locale fausse pendant un certain temps. Malheureusement, il est impossible de trouver un juste milieu entre performance et validité des informations, un pair pouvant être invariant pendant une longue période ou bien être très actif mais ce pendant un court moment. Il faut absolument en privilégier un au détriment de l'autre.

23. Metering and Monitoring Project, http ://meter.jxta.org/

En d'autres termes, l'agent fonctionne par scrutation de l'état du pair. Une amélioration probante serait de le faire fonctionner par notification. C'est à dire que le pair serait à même de prévenir l'agent à chaque fois que celui-ci change d'état. Ceci n'est pas proposé par actuellement par la plateforme JXTA, le choix entre scrutation et notification n'a pas été soulevé.

Le premier travail que j'ai réalisé fut d'intégrer les données de gestion issues du projet MMP dans le modèle pair à pair étendu pour JXTA. En d'autres termes, on ne peut pas traduire directement les objets MMP dans les objets CIM. Certaines classes issues de MMP vont s'intégrer dans différentes classes CIM. Par exemple, les données de gestion relatives au service de *Rendezvous* issues de MMP permettent de renseigner l'attribut *IsRendezvous* dans la classe *JxtaParticipatingPeer*, mais permettent aussi la création d'une instance de la classe *RendezvousConnection* représentant une connexion à un pair de *Rendezvous*.

4.2.2. *Conception d'un gestionnaire centralisé*

Il faut noter que la juxtaposition de plusieurs vues locales n'est pas suffisante à formaliser entièrement la plateforme pair à pair. En effet, par exemple, un *pipe* est par essence virtuel et est partagé entre deux pairs. De chaque coté, chaque pair connait une extrémité de ce *pipe*. Par contre aucun d'entre eux ne formalise ni ne doit formaliser le *pipe* en lui-même, c'est une instance partagée. C'est uniquement lors que l'on possède les deux extrémités du *pipe* que l'on peut en déduire que celui-ci existe.

Pour pouvoir accéder aux instances partagées, une vue globale, composée de la somme de toutes les vues locales plus ces instances partagées, est requise. La figure 11 illustre la création de cette vue globale. J'affecte cette tâche supplémentaire à un pair particulier qui joue le rôle de gestionnaire s'ajoutant à son rôle d'agent.

Le rôle de gestionnaire regroupe plusieurs sous-tâches comme :

– maintenir à jour une liste des agents présents dans la même communauté, sans oublier l'agent travaillant sur le pair qui l'héberge ;

– collecter les vues locales proposées par ses agents et les aggréger. Ceci est réalisé grâce à un client RMI qui va utiliser le connecteur RMI de chaque agent ;

– et finalement réaliser un travail de globalisation en rajoutant les liaisons entre les vues locales, à savoir les instances partagées, afin de rendre compte de toutes les caractéristiques de la plateforme.

Je fais travailler mon gestionnaire par délégation. En d'autres termes, celui-ci ne copie pas les informations récoltées chez ses agents mais uniquement la *coquille* des objets offrants des informations. En effet celui-ci ne créé qu'une redirection vers les informations de chaque agent. Ce mécanisme, illustré par la figure 12, permet d'obtenir toujours les informations les plus à jour possibles.

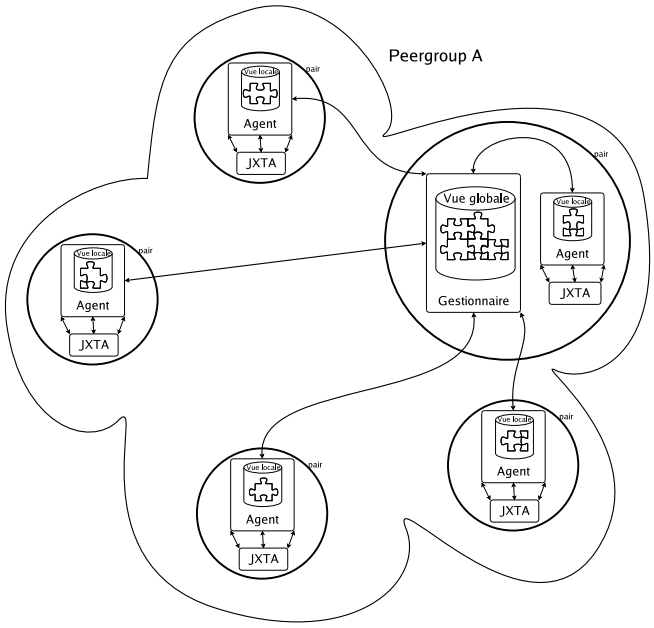

Figure 11. *Le gestionnaire collecte les différentes vues locales et les aggrége.*

Il est vrai que ceci est indispensable dans le cas d'un attribut variable, par exemple un compteur, mais peut être dispensable dans le cas d'une information invariable, par exemple pour un *uptime*[24]. La différenciation entre attribut variable et attribut non variable n'a pas été réalisée mais elle pourrait permettre de réduire l'utilisation du réseau.

4.3. *Modèle de communication : RMI et Handler Jxta*

L'implantation de référence du projet JXTA étant réalisée en Java, j'ai donc décidé de ne pas sortir du monde Java afin de conserver tous les avantages qu'offre ce langage en terme de portabilité. Pour cette raison, j'ai décidé d'utiliser l'implémentation RPC[25] de Java, à savoir RMI[26]. Dans cette partie, j'explique que RMI n'est pas suffisant et que l'on a besoin d'un second moyen de communication. Ensuite je montre comment le gestionnaire prend connaissance de la population de ses agents grâce à un *handler* JXTA.

24. donnée de gestion qui représente la date de démarrage d'un pair ou d'un service
25. Remote Procedure Call
26. Remote Method Invocation, http ://www.java.sun/docs/books/tutorial/rmi/

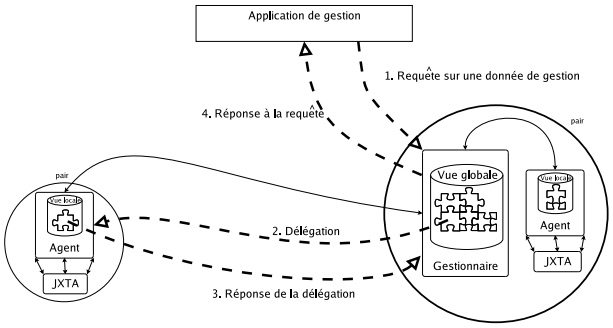

Figure 12. *Scénario détaillant le fonctionnement de la délégation*

4.3.1. *Interaction entre Gestionnaire et Agent : RMI*

Le but de RMI est de permettre l'appel, l'exécution et le renvoi du résultat d'une méthode exécutée dans une machine virtuelle différente de celle de l'objet l'appelant. J'utilise donc RMI afin de permettre à mon gestionnaire d'interagir avec les vues locales de ses agents. La consultation de la vue globale proposée par le gestionnaire sera également réalisée grâce à RMI.

Une lacune de RMI vient du fait qu'il faut connaître l'URL de connexion RMI de l'objet distant. Pour contourner ce problème, j'ai donc décidé de profiter de la puissance de la plateforme JXTA pour découvrir dynamiquement les URLs RMI.

4.3.2. *Découverte et enregistrement entre Gestionnaire et Agent : Handler JXTA*

Un problème se pose car le gestionaire ne possède pour le moment aucune information sur ses agents à propos de leur nombre et de leur interface de gestion. La première tâche à réaliser par le gestionnaire est donc de découvrir quels sont les agents, hébergés par des pairs JXTA, qui sont présents dans la même communauté. J'ai donc décidé de mettre en place un protocole permettant au gestionnaire de connaître l'interface de gestion de ses agents.

Premièrement, j'ai décidé de donner au gestionnaire la responsabilité de découvrir lui-même les agents. Ce principe de découverte contient en fait de nombreuses lacunes, à savoir :

– le gestionnaire ne pouvant savoir à quelle fréquence les agents apparaisent ou disparaissent, il est obligé de travailler par scrutation periodique afin d'être attentif à l'arrivée de nouveaux agents. Cela lui permet aussi de détecter la disparition d'un agent suite à la déconnexion d'un pair. Ce principe inonde le réseau de requête souvent inutiles ;

– si aucun gestionnaire n'est présent, il n'y a aucun moyen facile de le détecter ;

– il n'existe aucun mécanisme permettant de décider qui est gestionnaire ;

– et rien n'assure la présence d'un gestionnaire.

J'ai donc modifié le protocole en pensant que ce n'était pas au gestionnaire de découvrir ses agents, mais aux agents de prévenir de leur arrivée. En partant de ce postulat, chaque agent prévient toute la communauté lors de son arrivée. Celui-ci émet donc un message indiquant qu'il recherche un gestionnaire. Lors que le gestionnaire reçoit ce message, celui-ci référence l'agent et lui renvoie un acquitement. Ce système permet également de prendre en compte le cas où il n'existe pas de gestionnaire. En effet, si l'agent n'obtient aucune réponse à son message après une certaine période de temps, l'agent se proclame gestionnaire, en plus de sa fonction d'agent. Son nouveau statut de gestionnaire lui donne une nouvelle responsabilité, celle de répondre aux agents qui chercheront un gestionnaire. Ce principe permet également de ne pas inonder périodiquement le réseau de réquêtes afin de découvrir de potentiels nouveaux agents car le réseau n'est utilisé que lorsqu'un agent arrive.

Ce protocole est implémenté grâce à l'utilisation de messages pris en charge grâce à un *handler* JXTA, spécialisé pour ce protocole, que je charge dans chaque pair qui héberge un agent.

Le message envoyé par l'agent et reçu par le gestionnaire constitue le premier type de requête que peut recevoir un gestionnaire. J'utilise une deuxième requête dans la suite, qui sera présentée dans la section 5.2.1.

5. Application de gestion

Dans cette dernière partie, je présente l'application de gestion que j'ai réalisée afin de pouvoir facilement visualiser et interagir avec la vue globale proposée par un gestionnaire. En effet, sans cette application graphique, la vue globale serait difficilement exploitable.

5.1. *Fonctionnalités de l'application*

Dans cette partie je détaille les grandes fonctionnalités que doit fournir l'application de gestion. Celles-ci sont en accord avec le modèle fonctionnel défini précédemment.

5.1.1. *Connaître le réseau : Vue topologique*

La première utilité de cette application graphique est de pouvoir obtenir une vue topologique du réseau que l'on gère.

Il faut noter que les pairs qui hébergent un agent coexistent avec ceux qui ne proposent pas d'interface de gestion. La différenciation est réalisé au niveau de la couleur du pair sur la vue topologique. Les pairs possédant une interface de gestion sont verts, les autres rouges.

Il faut pouvoir voir facilement la communauté de pairs ainsi que l'état basique de chacun, ce dernier comprenant :

– le nom du pair ;
– l'identifiant unique du pair ;
– son heure d'arrivée ;
– son rôle au sein de la communauté (*Edge*, *Rendezvous* ou *Relay*) ;
– et le nom de l'élément qui héberge le pair.

La figure 13 montre un pair et l'affichage des informations de base citées précédemment.

Figure 13. *Affichage rapide des informations de base concernant un pair*

Pour les pairs n'offrant pas d'interface de gestion, on ne peut visualiser que leur nom et numéro d'identification.

La nature de chaque pair est transmise directement sur la vue topologique. En effet, les pairs de *Rendezvous* sont représentés par de plus gros icônes que les pairs simples, comme le montre la figure 14.

Figure 14. *Un pair de* Rendezvous *à coté d'un pair simple*

Ensuite, l'activation des liens de *Rendezvous* permet d'obtenir des détails sur les connexions aux pairs de *Rendezvous*. Cette relation topologique entre deux pairs est matérialisée par un fléche partant du pair client, pointant vers le pair de *Rendezvous*. L'affichage de ce niveau de détails peut être activé ou désactivé à volonté. Un bouton similaire permet d'obtenir une visualisation des *pipes* existants entre deux pairs. De la même manière, un *pipe* est représenté par un fléche partant du pair émetteur pointant vers le pair récepteur. Ce niveau de détail peut également être activé ou désactivé à volonté.

Une dernière fonctionnalité permet, uniquement sur les pairs ayant le rôle de *Rendezvous*, de rendre visible le *Rendezvous Peer View*. Cela correspond à la connaissance qu'un pair de *Rendezvous* posséde à propos des autres pairs de *Rendezvous*. Ce voisinage de *Rendezvous* est utilisé par ceux-ci afin de s'échanger aléatoirement, entre *Rendezvous*, des informations concernant les ressources proposées par le réseau afin de répartir équitablement la charge de chaque pair de *Rendezvous*.

La figure 17 montre, en arrière plan, une communauté composée de pairs normaux et de pairs offrants une interface de gestion. Certains sont des pairs de *Rendezvous*, d'autres des pairs simples. Les relations de *Rendezvous* ainsi que les *pipes* sont activées. De plus on peut voir le *Rendezvous Peer View* du pair *Rende... 4*.

5.1.2. *Surveiller le réseau*

Obtenir une vue topologique du réseau à un moment donné est intéressant mais pouvoir surveiller l'évolution de ce réseau sur une période donnée l'est encore plus. Pour cela, il est utile de pouvoir surveiller l'évolution d'une donnée de gestion en particulier. La figure 15 présente la fenêtre permettant la surveillance d'une métrique[27].

Le suivi de cette évolution pouvant permettre, lors du dépassement d'une certaine valeur, appelée seuil, le déclenchement d'une alarme. Cette alarme se matérialise sous forme d'une action spécifique, par exemple l'enregistrement dans un fichier journal, ou bien l'apparition d'un message sur l'écran. L'envoi d'un message électronique a aussi été envisagé mais non implanté.

Afin de rendre plus visuel le suivi d'une métrique, j'ai réalisé un histogramme, présenté par la figure 16. Lorsque la métrique dépasse le seuil, les barres de l'histogramme se colorient en rouge.

5.1.3. *Interagir avec le réseau*

Le dernier but d'une infrastructure de gestion et de pouvoir interagir avec le réseau. Sur une plateforme JXTA, je propose de pouvoir arrêter ou démarrer un service à la demande.

---

27. donnée de gestion numérique

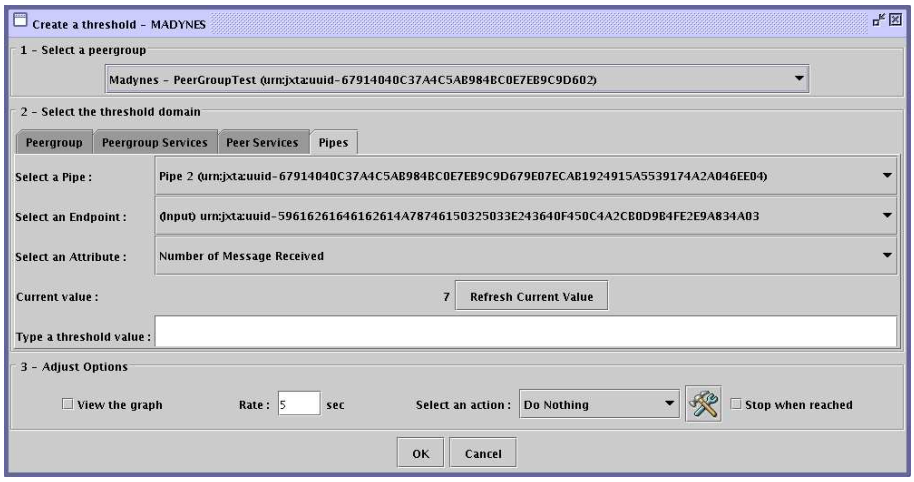

Figure 15. *Fenêtre pour créer une alarme sur une métrique particulière*

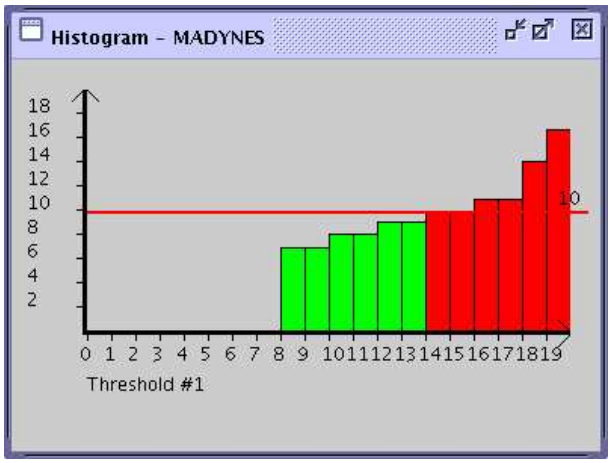

Figure 16. *Histogramme pour suivre l'évolution d'une métrique*

Pour cela, un simple clic droit sur un pair hébergeant un agent permet d'obtenir une liste de tous les services. Ensuite pour chaque service, un menu permet d'envoyer un ordre d'arrêt ou de démarrage du service séléctionné. La figure 17 montre l'effet d'un clic droit sur un pair offrant une interface de gestion.

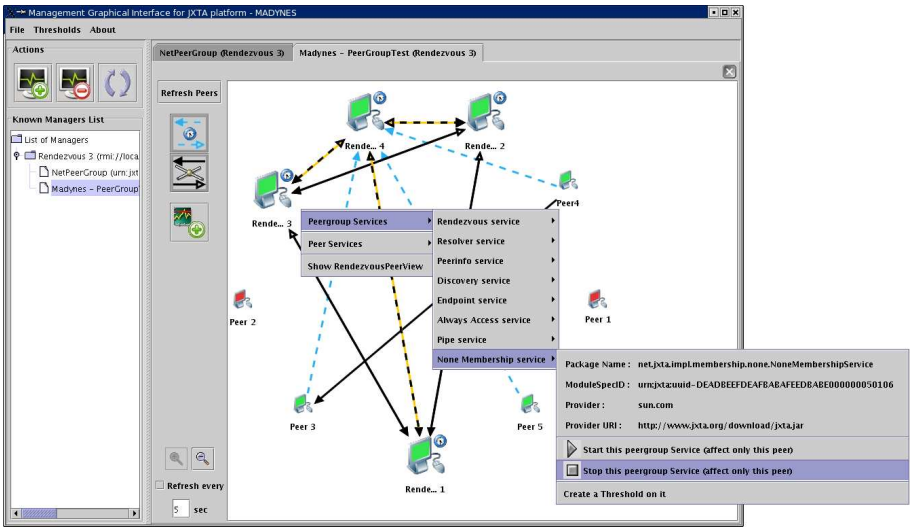

Figure 17. *Possibilité d'interaction avec un pair offrant une interface de gestion*

En réponse, le pair retourne une valeur correspondant au succès ou à l'échec de l'ordre. Cette réponse est interprétée dans l'application graphique afin de la rendre plus compréhensible.

5.2. *Fonctionnement*

Pour réaliser l'ensemble des tâches décrites précédemment, l'application de gestion doit se connecter à un gestionnaire.

5.2.1. *Connexion à un gestionnaire*

La manière la plus simple de se connecter à un gestionnaire est d'utiliser son URL[28] RMI. Cependant, comme les gestionnaires se proclament automatiquement, il peut être difficile d'obtenir cette URL statique mais aléatoire.

Afin de palier ce problème, j'ai mis en place un système de découverte. L'application graphique démarre un pair afin de se fondre dans le réseau JXTA, et envoie une requête. Cette requête utilise le même principe que celle utilisée par les agents afin de s'enregistrer auprès d'un gestionnaire. Cependant afin de ne pas se faire référencer comme agent, ce que l'application graphique n'est pas, un message différent est utilisé. La requête étant différente, la réponse l'est

28. Uniform Ressource Locator

également puisque le gestionnaire retourne son URL de connexion ainsi que la liste des *peergroups* dans lesquels il possède ce rôle.

De cette méthode peut résulter la connexion à plusieurs gestionnaires différents puisque tout gestionnaire recevant ce deuxième type de requête se doit de répondre.

5.2.2. *Scrutation de la vue globale à la demande*

Afin de proposer une vue graphique présentant tous les éléments d'une vue globale, l'application graphique fonctionne par scrutation de celle-ci. C'est à dire que l'application va collecter toutes les informations dont elle a besoin avant de réaliser le travail de rendu graphique. Il en résulte que le changement du réseau n'affecte pas la vue graphique proposée par l'application. L'activation du rafraîchissement automatique donne l'illusion de voir le réseau évoluer en temps réel mais le fonctionnement par notification reste au final la meilleure solution en terme de performances. Cette amélioration dans le fonctionnement de l'application reste tout de même complexe.

5.3. *Outils utilisés : Swing, jGraph*

Dans cette partie, je présente les concepts et outils que j'ai utilisés pour réaliser mon application de gestion.

5.3.1. *Modèle de conception : MVC*

Dans un but de clarté et de performance, j'ai suivi le modèle de conception MVC[29] pour l'organisation de mon application de gestion. Ce modèle de conception est largement utilisé pour le développement d'interface graphique.

5.3.2. *Java et application graphique. Librairies AWT et Swing*

Afin de réaliser cette application de gestion, j'ai choisi d'utiliser Java et de ses librairies graphiques, ceci dans un souci de compatibilité avec la spécification JMX et de portabilité, afin de pouvoir l'utiliser sur n'importe quel système d'exploitation. Java propose un ensemble d'éléments graphiques regroupés dans les librairies AWT, et Swing. AWT et Swing implantent le modèle de conception MVC. Ces librairies étant très connues et largement documentées[30], je choisi de ne pas détailler plus celles-ci.

---

29. Modèle/Vue/Controlleur
30. The Java Tutorial, http ://java.sun.com/docs/books/tutorial/uiswing/

5.3.3. *jGraph*

Pour le rendu graphique de la vue topologique de la plateforme JXTA, on m'a proposé d'utiliser jGraph[31], une librairie Java *open source*. Cette librairie permet, à partir d'un modèle spécifique, d'afficher un graphe. Dans cette partie, je présente les concepts de base permettant de comprendre l'utilisation de cette librairie graphique.

Un graphe est un ensemble d'éléments, appelés cellules (*Cells*). Ces cellules sont soit des sommets (*Vertices*) soit des arcs (*Edges*). Les sommets représentent les cases d'un graphe, les arcs, les liaisons entre sommets. Dans notre cas précis, un sommet sera utilisé pour afficher un pair, alors que un arc sera une relation topologique, un *pipe* par exemple, entre deux pairs.

Ensuite, afin d'afficher une vue topologique du réseau, il faut créer une vue qui se repose sur un modèle, peuplé de sommets et d'arcs. Afin de pouvoir obtenir une vue au visuel confortable, il est nécessaire d'appliquer une disposition spéciale, ou *layout*, celle-ci utilisant un algorithme complexe. En effet, jGraph est fournie avec plusieurs *layouts* permettant de disposer les sommets d'une manière différente suivant le style de graphe que l'on souhaite afficher. La disposition en cercle est la plus simple, mais il en existe de plus compliquées, ou spécifiques, comme par exemple la disposition en arbre et même en arbre radial. Pour mon projet j'ai utilisé une disposition en cercle, car l'utilisation des dispositions complexes n'est pas aisée.

6. Conclusion

Durant tout ce stage, j'ai poursuivi le projet que j'avais commencé lors de ma micro-thèse, celle-ci ayant servi d'introduction au stage.

Le projet a abouti au but initialement fixé : la réalisation d'une application de gestion d'une plateforme pair à pair en Java, en passant par la validation d'un modèle de l'information pour les réseaux et services pair à pair sur la plateforme JXTA et le déploiement d'une infrastructure de gestion sur celle-ci.

Mon travail va faire l'objet d'une demande d'intégration auprès de la communauté de développeurs travaillant sur le projet JXTA. Ceux-ci pourraient ainsi le proposer au sein de leurs distributions futures de la plateforme JXTA.

Dans ce rapport, je viens de mettre en avant le besoin de gestion dans le domaine des réseaux, et plus particulièrement pour les réseaux pair à pair. J'ai présenté ces derniers dans leurs grandes caractéristiques, et je me suis appuyé sur une plateforme en particulier : JXTA. Ensuite j'ai détaillé la démarche de conception d'une infrastructure de gestion et grâce au modèle de l'information de gestion pour le pair à pair de l'équipe MADYNES, j'ai réalisé ma propre

31. jGraph Home Page : http ://www.jgraph.com

application de gestion pour la plateforme JXTA. Avant tout, j'ai du valider le modèle de l'information de gestion pour le pair à pair tout en l'étendant, ce qui m'a permis de rendre compte de tous les détails de la plateforme JXTA.

Pendant ce stage, j'ai été confronté à divers problèmes. L'un d'entre eux vient du fait qu'il a fallu assimiler les différentes technologies abordées. Ensuite, une phase de compréhension globale m'a permi de mieux visualiser comment l'imbrication de ces dernières me permettait d'arriver à l'objectif fixé. Aussi, étant amené à travailler à partir de travaux regroupant des communautés à l'échelle mondiale, la majorité des documentations sont diffusées en anglais, langue technique internationale. Ceci a ralenti la compréhension et l'assimilation des documentations relatives aux différentes technologies utilisées.

Le déploiement de l'infrastructure de gestion peut être amélioré. Le fonctionnement par scrutation n'est pas le meilleur d'un point de vue de la performance. La mise en place d'un système de notifications est bien sûr plus judicieux bien que plus complexe à implanté. Sinon des tests de passage à l'échelle doivent être envisagés, car la vue globale, hébergée par le gestionnaire, regroupe rapidement quelques milliers de MBeans, et ce pour une communauté de pairs réduite. Une dizaine de pairs se sont vus représenter par environ 1700 MBeans lors de mes premiers tests. Le gestionnaire pourrait donc souffrir d'un tel besoin de ressources.

Remerciements

Je tiens à remercier Monsieur Olivier Festor, Directeur de l'équipe Madynes, de m'avoir accueilli pendant la durée de mon stage, ainsi que son équipe qui fut disponible et attentive à mes questions.

Je remercie particulièrement Guillaume Doyen pour son accueil, ses conseils et sa disponibilité.

7. Bibliographie

[BUM 00] Bumpus W., Sweitzer J. W., Thompson P., R. W. A., Williams R. C., *Common Information Model*, Wiley, 2000.

[CAS 90] Case J., Fedor M., Schoffstall M., Davin J., « Simple Network Management Protocol (SNMP) », Internet Engineering Task Force, may 1990, RFC 1157 (Historic).

[Dis 00] Distributed Management Task Force, « Common Information Model (CIM) Core model, version 2.4 », www.dmtf.org/standards/documents/CIM/DSP0111.pdf, August 2000.

[Dis 03] Distributed Management Task Force, « CIM Concepts White Paper, CIM versions 2.4+ », www.dmtf.org/standards/documents/CIM/DSP0144.pdf, June 2003.

[DOY 04a] Doyen G., Festor O., Nataf E., « A CIM extension for peer-to-peer network and service management », De Souza J., Dini P., Eds., *Proceedings of the 11th International Conference on Telecommunication - ICT'04*, n° 3124 LNCS, Springer-Verlag, 2004, p. 801-810.

[DOY 04b] Doyen G., Nataf E., Festor O., « A performance-oriented management information model for the Chord P2P framework », Vicente J., Hutchison D., Eds., *Management of Multimedia Networks and Services - MMNS'04*, n° 3271 LNCS, Springer-Verlag, 2004, p. 200-212.

[DOY 05] Doyen G., Nataf E., Festor O., « Une architecture hiérarchique pour une gestion distribuée des réseaux et services pair à pair », Benzekri A., Sibilla M., Eds., *6ième colloque Francophone sur la Gestion des Réseaux et Services - GRES'05*, 2005.

[FES 03] Festor O., Ben Youssef N., « *Standards pour la gestion des réseaux et des services* », chapitre Le modèle CIM, p. 113-157, Hermes, 2003.

[JAS 02] Jasnowski M., *JMX Programming*, Wiley, 2002.

[KAN 01] Kan G., Gnutella, GoneSilent.com, « *Peer-to-peer : Harnessing the Power of Disruptive Technologies* », chapitre Gnutella, p. 94-122, O'Reilly & Associates, Inc., 2001.

[MIL 02] Milojicic D., Kalogeraki V., Lukose R., Nagaraja K., Pruyne J., Richard B., Rollins S., Xu Z., « Peer-to-Peer Computing », rapport n° HPL-2002-57, 2002, HP Laboratories.

[MOH 04] Mohanti R., « Instrumentation of the Jxta peer-to-peer framework », Master's thesis, Indian Institute of Technology - Kharagpur, Novembre 2004.

[OAK 02] Oaks S., Traversat B., Gong L., *Jxta in a nutshell*, O'Reilly, 2002.

[ROW 01] Rowstron A., Druschel P., « Pastry : Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems », *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms - Middleware'01*, n° 2218 LNCS, Springer-Verlag, 2001, p. 329-350.

[SHI 01] Shirky C., « *Peer-to-peer : Harnessing the Power of Disruptive Technologies* », chapitre Listening to Napster, p. 21-37, O'Reilly & Associates, Inc., 2001.

[STO 01] Stoica I., Morris R., Karger D., Kaashoek M. F., Balakrishnan H., « Chord : A scalable peer-to-peer lookup service for internet applications », *Proceedings of the ACM Conference on Applications, Technologies, Architectures and Protocols for Computer Communication - SIGCOMM'01*, ACM Press, 2001, p. 149-160.

[Sun05] Sun Microsystems, « JXTA v2.3.x : Java Programmer Guide », April 2005, http ://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf.

[WIL 02] Wilson B. J., *JXTA*, New Riders Publishing, 2002.