Supervision des réseaux et services pair à pair

Guillaume Doyen

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Supervision des réseaux et services pair à pair

THÈSE

présentée et soutenue publiquement le 12 décembre 2005

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1
(spécialité informatique)

par

Guillaume Doyen

Composition du jury

Président :	Nacer Boudjlida, Professeur des Universités	Université Henri Poincaré, Nancy
Rapporteurs :	Michelle Sibilla, Maître de Conférences, Hdr	Université Paul Sabatier, Toulouse
	François Spies, Professeur des Universités	Université de Franche Comté, Besançon
Examinateurs :	Olivier Festor, Directeur de Recherche	Inria Lorraine, Nancy
	Stéphane Frenot, Maître de Conférences	Insa, Lyon
	Emmanuel Nataf, Maître de Conférences	Université Nancy 2, Nancy

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Remerciements

Ce manuscrit de thèse n'aurait pas pu voir le jour sans l'aide de quelques personnes que je tiens à remercier particulièrement.

Je remercie tout d'abord les membres du jury, Michelle Sibilla, Nacer Boudjlida, Stéphane Frenot et François Spies pour l'intérêt qu'ils portent à mon travail.

Ensuite, j'adresse mes plus chaleureux remerciements à Olivier Festor, mon directeur de thèse, avec qui j'ai eu beaucoup de plaisir à travailler et qui a énormément contribué à mon évolution scientifique et technique.

Je remercie tout autant Emmanuel Nataf, mon encadrant de thèse, pour l'ensemble des conseils qu'il a su me donner au quotidien, aussi bien sur des questions de recherche que d'enseignement.

L'ambiance qui règne au sein de l'équipe Madynes a été un facteur important du bon déroulement de ce travail. Je remercie chaque membre pour sa bonne humeur et sa disponibilité.

Plus personnellement, je tiens à remercier Isabelle, *my personal English teacher, for all the advice she gave to me and all the time she spent reading and correcting my English papers and presentations !* En outre, sa patience et sa disponibilité m'ont été précieuses tout au long de ces trois années et particulièrement pour l'écriture de ce manuscrit.

Enfin, je tiens à remercier Evelyne, Isabelle et Marie, mes trois relectrices, pour le temps qu'elles ont passé à lire un manuscrit dont elles ne comprenaient pas un mot.

A toutes et à tous, merci.

Je dédie cette thèse à Isabelle, ma compagne.

Table des matières

Table des figures

Liste des tableaux

Introduction

Chapitre 1

Introduction générale

Sommaire

1.1 Cadre scientifique

Ce manuscrit présente la synthèse de trois années de travail effectué dans le cadre d'une thèse. Celle-ci s'est déroulée au sein de l'équipe MADYNES[1] du LORIA[2] à Nancy. Le LORIA est une unité mixte de recherche (UMR 7503) commune à plusieurs établissements qui sont le CNRS[3], l'INPL[4], l'INRIA[5], l'université Henri Poincaré[6] et l'université Nancy 2[7]. Ses missions concernent la recherche fondamentale et appliquée au niveau international dans le domaine des Sciences et Technologies de l'Information et de la Communication (STIC), la formation par la recherche en partenariat avec les universités lorraines et le transfert technologique par le biais de partenariats industriels et par l'aide à la création d'entreprises.

L'équipe MADYNES, dirigée par Olivier Festor, est organisée autour de deux axes de recherche : la gestion des réseaux et services, et la sécurité des réseaux. Dans ce cadre, plusieurs thématiques particulières font l'objet d'études et de propositions, notamment, la sécurité du plan de gestion, la performance des infrastructures de supervision, l'administration des réseaux Ad Hoc, et enfin la supervision des réseaux pair à pair. Ce dernier thème constitue le sujet de thèse sur lequel j'ai travaillé. Je présente ici les travaux effectués ainsi que les résultats obtenus.

[1] Management of Dynamic Networks and Services - http ://madynes.loria.fr
[2] Laboratoire Lorrain de Recherche en Informatique et Automatique - http ://www.loria.fr
[3] Centre National de Recherche Scientifique - http ://www.cnrs.fr
[4] Institut National Polytechnique de Lorraine - http ://www.inpl-nancy.fr
[5] Institut National de Recherche en Informatique et en Automatique - http ://www.inria.fr
[6] http ://www.uhp-nancy.fr
[7] http ://www.univ-nancy2.fr

1.2 Contexte

Depuis 1999, date à laquelle le logiciel Napster connut un succès sans précédent, le modèle pair à pair (P2P) attire l'attention des acteurs de la communauté des réseaux et des applications distribuées. Les scientifiques et les industriels voient en ce modèle une véritable alternative au modèle client-serveur qui est *le* modèle actuellement déployé dans les applications informatiques en réseaux. Le modèle P2P est un modèle distribué où les entités appelées pairs jouent le double rôle de client et serveur et interagissent afin d'offrir à une communauté un service de manière décentralisée. Cette décentralisation des ressources et des services présente de nombreux avantages qui repoussent les limites induites par le modèle client-serveur. Tout d'abord, l'agrégation potentielle des puissances de calcul et espaces de stockage de millions de machines offre aux usagers une puissance qui dépasse celle de toutes les infrastructures centralisées existantes. Ensuite, l'absence d'élément central permet d'augmenter considérablement la tolérance aux fautes des services car, si la disparition d'un serveur engendre l'indisponibilité du service offert, la disparation d'un pair est sans conséquence, étant donné l'ensemble des pairs présents. Troisièmement, la répartition équitable des données et des tâches à effectuer permet d'équilibrer le trafic du réseau sous-jacent ainsi que la charge attribuée à chaque participant. Enfin, l'utilisation de machines appartenant à des propriétaires différents permet de réduire les coûts liés à l'achat et la maintenance d'équipements.

Toutes ces bonnes propriétés expliquent l'attention particulière qui est portée au modèle P2P. Si les premières applications de ce modèle étaient exclusivement liées à l'échange souvent illégal de données soumises à des droits d'auteur, c'est maintenant l'ensemble des services réseaux qui peuvent être repensables et repensés pour être déployés selon une approche P2P. Dans le domaine du stockage de données, on trouve actuellement des applications comme OceanStore ou CFS qui proposent à des usagers une infrastructure robuste et fiable pour le stockage et l'accès à leurs données personnelles depuis n'importe quel point d'accès à l'Internet. Dans le domaine de la communication, Jabber et Skype offrent respectivement des solutions de communication et de voix sur IP qui permettent à chacun d'échanger et de dialoguer avec ses connaissances indépendamment de tout opérateur téléphonique et quel que soit son emplacement. Dans le domaine du calcul distribué, la puissance de calcul résultant de l'agrégation des processeurs de machines connectées repousse les limites des super-calculateurs actuels. On trouve ainsi de nombreuses applications comme SETI@Home ou Genome@Home qui proposent d'utiliser le modèle P2P pour distribuer un calcul colossal. Le modèle P2P ouvre donc de nouveaux horizons aux applications réseaux déployées, non seulement chez des particuliers, mais aussi dans le cadre plus formel des entreprises, administrations et universités.

1.3 Problématique

Cette nouvelle utilisation du modèle P2P, dans des environnements contraints où la qualité du service offert est primordiale, nécessite la mise en place d'une infrastructure de supervision qui puisse surveiller et contrôler les services reposant sur ce modèle. Par exemple, dans le cadre de l'utilisation d'une infrastructure P2P pour la mise en œuvre d'un service de système de fichiers distants comme NFS[8], la garantie d'un espace disque minimum qui soit accessible à chaque usager est nécessaire. Or, dans un environnement où chaque pair peut aller et venir de manière imprévisible, entraînant ainsi une fluctuation de l'espace disponible, cette garantie n'est ni automatique ni aussi simple à assurer que dans le cas du modèle client-serveur. L'intégration

[8]Network File System

de mécanismes de supervision aux services P2P est donc indispensable à leur utilisation sous contraintes de qualité de service.

Les approches de gestion actuelles sont conçues pour des services à caractère centralisé et s'adaptent mal aux propriétés du modèle P2P. C'est pourquoi le travail présenté ici concerne la conception, la validation et la mise en œuvre d'une infrastructure de supervision pour les réseaux et services P2P qui respecte les spécificités de ce modèle.

Remarque : Le modèle P2P n'a pas été initialement conçu pour exploiter un plan de supervision. Au fil de son évolution, des mécanismes de gestion comme les approches incitatives y ont été intégrés et montrent le besoin de gestion qui existe pour ce modèle. Notre contribution se place dans cet effort et consiste à proposer l'intégration de fonctions de supervision fondées sur des approches standard.

1.4 Organisation du manuscrit

Afin de présenter les différents travaux que nous avons conduits dans le cadre de la supervision des réseaux et services P2P, nous avons organisé ce manuscrit en trois parties principales qui sont : (1) l'état de l'art des domaines du pair à pair et de la supervision des réseaux, (2) les propositions de solutions que nous avons faites pour une gestion adaptée au modèle P2P, et (3) la validation de ces propositions à travers leur mise en œuvre sur des infrastructures existantes. Enfin, une conclusion et un ensemble de perspectives clôturent ce manuscrit.

1.4.1 Partie I : Etat de l'art

Cette partie, composée de trois chapitres, est une synthèse des différents domaines de recherche qui ont constitué le cadre de notre travail.

Le modèle pair à pair

Le chapitre 2, premier chapitre de cette partie, présente le modèle P2P. Afin de caractériser ce modèle, nous proposons une définition qui synthétise plusieurs définitions, issues de la communauté P2P. La distinction de trois sous-modèles, appelés sous-modèles pur, hybride et centralisé, relatifs au niveau de décentralisation des composants, permet d'affiner ce travail de formalisation. Dans un second temps, nous nous intéressons aux caractéristiques du modèle P2P. Chacune d'entre elles lui confère un certain nombre de propriétés. La décentralisation permet d'assurer le passage à l'échelle, la tolérance aux fautes, la répartition des coûts et l'équilibre de la charge et du trafic des applications P2P. L'auto-organisation des composants se répercute sur le plan des tâches, des rôles et de la topologie virtuelle. La connectivité intermittente des pairs nécessite la mise en œuvre de mécanismes de routage, stockage et accès adaptés. Enfin, l'utilisation d'un réseau virtuel permet d'abstraire les différences physiques des terminaux et nécessite le déploiement de services de nommage et de routage de niveau applicatif. La dernière partie de ce chapitre est dédiée aux applications P2P. Après avoir distingué les différentes classes d'applications, nous présentons Jxta, une plate-forme générique pour le développement d'applications P2P sur laquelle nous avons travaillé.

Les tables de hachage distribuées

Le chapitre 3 aborde un point crucial du modèle P2P : la découverte et la localisation de ressources dans un environnement dynamique. Après avoir mis en évidence les limites des

modèles à répertoire centralisé et à inondation, nous présentons le principe général des tables de hachage distribuées (DHT). Nous identifions plusieurs classes de DHTs qui se distinguent par le modèle topologique qu'elles utilisent. Nous présentons en particulier Chord et Pastry qui sont deux infrastructures sur lesquelles nous avons travaillé. Etant donné le nombre conséquent de propositions de DHTs, nous avons synthétisé la comparaison de certaines d'entre elles dans la troisième partie de ce chapitre. Enfin, nous terminons celui-ci en présentant brièvement les évolutions de ces infrastructures vers la standardisation et l'intégration de recherche sémantique.

Pair à pair et gestion de réseaux

Le chapitre 4, dernier chapitre de l'état de l'art, concerne la gestion de réseaux appliquée au modèle P2P. Dans un premier temps, nous présentons de manière générale le domaine de l'administration des réseaux, ses objectifs, ses modèles ainsi que les différentes approches existantes. Ensuite, nous détaillons le besoin de gestion qui existe pour les réseaux et services P2P et la nécessité de concevoir de nouvelles infrastructures adaptées. Une troisième partie énumère les différents travaux existants dans le domaine de la supervision du modèle P2P. Ceux-ci concernent principalement les mécanismes incitatifs et la gestion de la topologie. Dans un quatrième temps, nous considérons l'approche inverse, à savoir l'utilisation du modèle P2P pour la supervision des réseaux et services. Nous détaillons en particulier un travail de thèse qui considère cette approche. Enfin, nous terminons ce chapitre par la proposition d'une classification des applications P2P orientée vers la supervision. Celle-ci a pour objectif de mettre en avant les différences entre les applications P2P et la manière dont elles vont se répercuter sur la conception d'une infrastructure de supervision.

1.4.2 Partie II : Contributions

Cette partie présente les travaux de conception d'une infrastructure de gestion adaptée au modèle P2P que nous avons effectués. Ceux-ci sont relatifs à trois des quatre modèles que toute infrastructure de supervision doit définir et mettre en œuvre.

Modèle de l'information

Le chapitre 5 s'intéresse au modèle de l'information pour des données de gestion. Il présente une extension du modèle commun de l'information (CIM) pour les réseaux et services P2P. Ce modèle, reposant sur le paradigme orienté objet et proposant un nombre conséquent d'éléments réutilisables, est un choix judicieux pour notre travail de modélisation. Nous avons distingué cinq sous-modèles qui caractérisent le modèle P2P. Le premier est relatif à l'organisation fonctionnelle et topologique des pairs. Le second modélise les ressources disponibles et consommées dans le cadre d'un service P2P. Le troisième s'intéresse à la manière dont les pairs communiquent au sein d'une communauté. Le quatrième concerne le service offert par les pairs, d'une manière individuelle et agrégée. Enfin, le dernier sous-modèle représente les services de routage et de calcul de routes. Ainsi, ce modèle de l'information permet de représenter, du point de vue de la gestion, une communauté quelconque de pairs, et d'en fournir une vue abstraite à un gestionnaire.

Modèle fonctionnel de la performance

Le chapitre 6 s'intéresse au modèle fonctionnel de la gestion qui concerne la performance. Nous nous sommes intéressés aux tables de hachage distribuées et à la performance du service qu'elles offrent. Nous avons abstrait le fonctionnement d'une DHT et défini un ensemble de

métriques qui caractérisent son fonctionnement. Cette modélisation est générique et peut ainsi s'appliquer à n'importe quelle infrastructure existante. Pour formaliser les données de gestion issues de ces métriques, nous avons conçu une extension de notre modèle de l'information qui repose sur le modèle de métriques de CIM. Enfin dans le but de valider cette proposition, nous l'avons instanciée sur la DHT Chord, pour laquelle nous avons également ajouté des métriques particulières.

Modèle organisationnel

Le chapitre 7, dernier chapitre de cette partie, s'intéresse au modèle organisationnel du plan de gestion. Nous y présentons une proposition d'architecture de gestion dédiée au modèle P2P qui repose sur un modèle hiérarchique. La règle de construction de cette hiérarchie repose sur l'identification de préfixes communs dans les identifiants des nœuds. Les avantages de cette construction sont nombreux. Elle permet tout d'abord d'établir une structure équilibrée qui exclut la surcharge d'un pair particulier. Ensuite, de par l'utilisation de nombreux pairs gestionnaires, elle permet de distribuer la charge de gestion et minimise ainsi son impact sur la performance des nœuds impliqués. Enfin, son principe de construction totalement distribué permet aux pairs d'auto-organiser le plan de gestion sans avoir recours à un élément central. Nous avons implanté un prototype de cette proposition sur la DHT Pastry. Nous présentons les premiers résultats des tests de validation que nous avons effectués.

1.4.3 Partie III : Expérimentations

Le chapitre 8 concerne un travail de déploiement de nos propositions sur la plate-forme P2P Jxta. Pour celle-ci, nous avons spécialisé notre modèle de l'information générique de manière à intégrer les spécificités de la plate-forme. Ensuite, nous avons déployé ce modèle au sein d'une communauté de pairs Jxta. Nous avons organisé le plan de gestion selon un modèle centralisé qui comporte un gestionnaire et n agents. Nous avons montré que ce gestionnaire est capable de reconstituer une vue abstraite d'une communauté Jxta et de la formaliser à l'aide de notre modèle de l'information. De plus, nous avons conçu et mis en œuvre une application de gestion qui interagit avec un gestionnaire et fournit à un opérateur humain une vue topologique, organisationnelle et fonctionnelle de la plate-forme. En outre, un mécanisme d'interaction fondé sur le démarrage et l'arrêt de service a montré la possibilité de contrôler Jxta par le biais de la gestion.

Pour terminer ce manuscrit, nous présentons les conclusions que nous avons tirées du travail effectué durant ces trois années de thèse et nous proposons quelques pistes pour la suite.

Première partie

Etat de l'art

Chapitre 2

Le modèle pair à pair

Sommaire

Dans ce premier chapitre, nous présentons le modèle pair à pair. Nous proposons tout d'abord un ensemble de définitions qui le spécifient. Puis, nous passons en revue l'ensemble de ses caractéristiques ainsi que les propriétés qu'elles lui confèrent. Nous nous intéressons ensuite à ses applications. Nous proposons pour celles-ci une classification qui distingue quatre domaines et, pour chacun de ces domaines, nous présentons les principales applications. Enfin, nous présentons Jxta, la plate-forme de développement d'applications P2P sur laquelle nous avons travaillé.

2.1 Généralités

Le modèle pair à pair[1] (P2P)[2] est un modèle distribué composé d'éléments qui jouent le rôle de client et serveur. Ce modèle connait un très grand succès depuis la fin des années 90, époque à laquelle Napster, une application P2P de partage de fichiers permit à des millions d'usagers connectés à l'Internet de télécharger et partager librement des fichiers multimédias.

La fin du service Napster due à un procès initié par les grands noms de l'industrie du disque eut de fortes répercussions chez les acteurs des domaines de l'informatique et des réseaux. Les

[1] ou Poste à Poste

[2] Dans la suite de ce manuscrit, en fonction du contexte, on utilisera indifféremment les termes de pair et nœud pour désigner un pair.

industriels et les scientifiques saisirent qu'au delà de cette application, le modèle P2P est porteur d'une grande puissance alors qu'il n'était jusque-là que très peu considéré. En effet, le modèle client-serveur, du fait de la simplicité de sa mise en œuvre et de sa maîtrise acquise au fil des années, fut déployé pour fournir la majorité des services. Messagerie, *web*, calcul distribué, transfert de fichiers et voix sur IP sont quelques exemples d'applications qui sont construites selon ce modèle. Le modèle P2P, du fait de sa décentralisation et de l'utilisation potentielle des ressources de millions de machines, repousse considérablement les limites imposées par le modèle client-serveur.

Suite à l'arrêt du service Napster, un autre groupe de développeurs se lança dans le projet de conception d'une application similaire à Napster mais qui soit complètement distribuée et autonome, échappant ainsi à tout contrôle : Gnutella. Cette application montra alors qu'il est possible de proposer aux usagers un service qui ne repose sur aucune infrastructure physique concrète mais sur un simple logiciel distribué au sein d'une communauté d'utilisateurs. La vague P2P fut lancée.

Actuellement, la recherche et l'industrie voient en ce modèle une vraie alternative aux applications client-serveur et contribuent à de nombreux travaux dans ce domaine. En effet, les applications possibles ne comptent pas que le seul partage de fichiers multimédias. Toutes les applications actuelles qui sont construites selon le modèle client-serveur peuvent être repensées et améliorées pour fonctionner selon ce nouveau modèle. En outre, ce sont aussi de nouvelles applications informatiques qui peuvent être envisagées, conçues et déployées. Parmi celles-ci, on trouve le calcul distribué qui propose d'utiliser des machines oisives connectées à l'Internet pour effectuer de petites portions d'un calcul colossal, et les espaces collaboratifs qui proposent de construire des lieux d'échange et de communication avancés pour les utilisateurs de l'Internet.

Dans cette section, nous proposons un ensemble de définitions pour le modèle P2P. Nous identifions en particulier trois sous-modèles qui se différencient par leur degré de centralisation.

2.1.1 Définitions

Les systèmes informatiques peuvent être classifiés selon la taxonomie illustrée sur la partie gauche de la figure 2.1 qui permet de situer les systèmes P2P par rapport aux systèmes actuels. On voit tout d'abord que les systèmes informatiques se scindent en deux classes qui sont les systèmes centralisés, par exemple de type *Main Frame*, et les systèmes distribués. Les systèmes distribués se scindent eux-mêmes en deux familles différentes qui sont celles des systèmes client-serveur et P2P. Chacune de ces familles étant composée d'un ensemble de sous-modèles.

Dans la littérature, on trouve plusieurs définitions du modèle P2P. Schollmeier a d'ailleurs travaillé exclusivement sur la proposition de définitions pour les architectures P2P [146]. Ici, nous en citons trois qui sont proposées par différents acteurs de la communauté P2P :

1. *Peer-to-peer is a class of applications that take advantage of resources (storage, cycles, content, human presence) available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, peer-to-peer nodes must operate outside the DNS and have significant or total autonomy of central servers* [147].
2. *A distributed network architecture may be called a Peer-to-Peer network, if the participants share a part of their own hardware resources (processing power, storage capacity, network link capacity, printers, ...). These shared resources are necessery to provide the Service and content offered by the network. They are accessible by other peers directly, without passing intermediary entities. The participants of such a network are thus resource providers as well as resource requestors* [146].

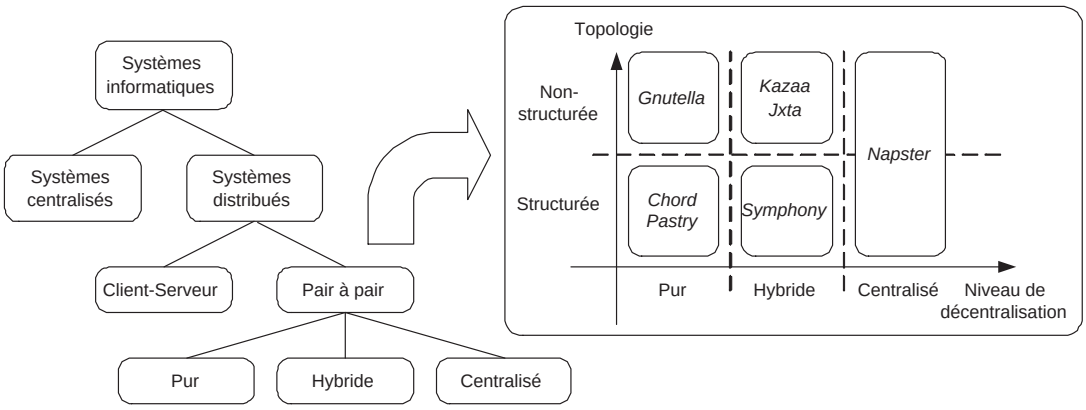


FIG. 2.1 – Classification des systèmes informatiques et des applications P2P

3. *Peer-to-peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority* [9].

Ces trois définitions diffèrent quelque peu dans la vision qu'elles apportent du modèle P2P. Néanmoins, elles introduisent toutes plusieurs concepts généraux qui sont :
- le partage des ressources (introduit dans les définitions 1, 2 et 3) ;
- le caractère dynamique des participants (introduit dans les définitions 1 et 3) ;
- la capacité à s'auto-organiser (introduit dans les définitions 1 et 3) ;
- l'absence d'élément central (introduit dans les définitions 1, 2 et 3).

De notre côté, la définition du modèle P2P que nous proposons fait abstraction des concepts de partage de ressources et de dynamisme pour se concentrer plutôt sur la nature décentralisée du modèle :

Définition 1 *Le terme pair à pair désigne un modèle distribué où les entités appelées pairs jouent le double rôle de client et serveur et interagissent afin d'offrir à une communauté un service de manière décentralisée.*

Cette nouvelle définition élargit l'étendue des définitions 1, 2 et 3. En ne spécifiant aucun critère de dynamicité, elle permet de prendre en compte des systèmes relativement statiques comme par exemple l'organisation P2P de serveurs OceanStore [93] ou les confédérations de routeurs fonctionnant à l'aide du protocole RIP [104]. Ensuite, bien qu'elle n'exprime pas explicitement la notion de partage de ressources, celle-ci apparaît à travers le rôle de serveur qui incombe à chacun des pairs participants. Dans la suite de ce manuscrit, les infrastructures, services et applications P2P que nous mentionnons suivent tous la définition 1 du modèle P2P.

2.1.2 Différents niveaux de décentralisation

La définition du modèle P2P que nous proposons indique qu'au sein d'une communauté, les pairs interagissent de manière à fournir un service par leurs propres moyens. Néanmoins, elle ne définit pas la manière avec laquelle les pairs se partagent les tâches à accomplir pour fournir le service qu'ils offrent. Actuellement, les applications P2P sont construites selon un

degré de décentralisation variable qui représente une distribution plus ou moins forte des tâches accomplies. Les différentes propositions de classification des infrastructures P2P en fonction de leur degré de décentralisation scindent le modèle P2P en deux [112, 146] ou trois [3, 121] sous-modèles. Nous considérons aussi que le modèle P2P peut se scinder en trois sous-modèles qui sont les modèles pur, hybride et centralisé. La figure 2.1 représente un exemple de topologie de chacun de ces trois sous-modèles.

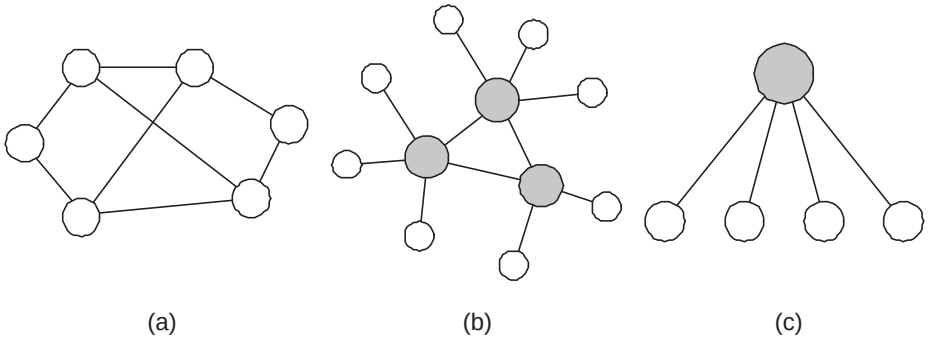


FIG. 2.2 – Exemple de topologies P2P qui présentent différents niveaux de décentralisation. (a) Topologie construite selon le modèle pur. (b) Topologie construite selon le modèle hybride. (c) Topologie construite selon le modèle centralisé.

Le modèle pur

Dans [146], Schollmeier propose une définition du modèle pur qui le caractérise par le fait que, dans une communauté de pairs, la suppression de n'importe lequel d'un des pairs présents n'affecte pas le service offert. La définition que nous proposons définit le même concept, avec toutefois une formulation plus explicite :

Définition 2 *Le modèle P2P pur représente un modèle P2P tel qu'il est spécifié dans la définition 1 et dans lequel tous les pairs sont strictement équivalents.*

Un exemple de topologie construite selon le modèle pur est représenté sur la figure 2.2.a où l'on voit qu'aucun élément ne joue de rôle particulier. Actuellement, de nombreuses applications P2P sont construites selon ce modèle. On trouve par exemple Gnutella [84], tel qu'il était déployé dans sa première version, et toutes les infrastructures à base de table de hachage distribuée telles que Chord [153], Pastry [137], Tapestry [170], CAN [124], ...

Le modèle hybride

Le modèle hybride, dont un exemple de topologie est représenté sur la figure 2.2.b, ajoute un degré de hiérarchie au modèle pur. La définition que nous proposons pour ce modèle est la suivante :

Définition 3 *Le modèle P2P hybride représente un modèle P2P tel qu'il est spécifié dans la définition 1 et dans lequel certains pairs jouent un rôle particulier. Ces derniers exécutent des fonctions différentes des autres pairs rendant les pairs non-équivalents et apportant ainsi un certain degré de centralisation.*

Les pairs *particuliers* utilisés dans le modèle hybride sont généralement appelés des super-pairs[3]. Le rôle qu'ils jouent n'est pas figé et varie d'une infrastructure à une autre. En général, ils sont utilisés pour assurer des fonctions relatives au routage [68], à la comptabilité [73] ou à l'organisation fonctionnelle [161] des pairs. Par exemple, Kazaa[4] et Jxta [68] les utilisent pour la découverte et la localisation de ressources.

Le modèle centralisé

Le modèle centralisé est à la limite du modèle P2P car il repose sur un serveur dédié qui centralise et maintient l'ensemble des connaissances de la communauté, les ressources étant toujours hébergées sur les pairs. Ce modèle est utilisé dans des applications telles que Napster ou Skype. Un exemple de topologie P2P centralisée est représentée sur la figure 2.2.c qui montre que chaque pair ne possède au minimum qu'une connaissance du serveur central et pas des autres pairs, bien qu'une fois les opérations de découverte et localisation effectuées, il puisse interagir directement avec eux. Ainsi, cette interaction possible entre les pairs différencie le modèle P2P centralisé du modèle client-serveur. La définition que nous proposons pour ce modèle est la suivante :

Définition 4 *Le modèle P2P centralisé représente un modèle P2P tel qu'il est spécifié dans la définition 1 et dans lequel un serveur dédié est utilisé pour assurer les fonctions de découverte et localisation de ressources.*

Remarque : Certaines classifications des architectures P2P considèrent les modèles hybrides et centralisés, tels que nous les définissons comme identiques ; le modèle centralisé étant alors un cas particulier du modèle hybride contenant un seul super-pair. Toutefois, nous préférons conserver la distinction entre ces deux modèles, car le modèle hybride peut être vu comme la composition des modèles purs et centralisés, en ce sens que, chaque super-pair agit comme une entité centrale pour les pairs qui lui sont connectés, mais que l'organisation des super-pairs suit le modèle pur.

2.2 Caractéristiques

Le modèle P2P apporte une nouvelle manière de concevoir et mettre en œuvre les applications réseaux. De par ses caractéristiques intrinsèques, il permet de résoudre certains problèmes posés par le modèle client-serveur, comme par exemple, la limite du passage à l'échelle ou le coût important pour l'achat et la maintenance des équipements. Toutefois, il en induit d'autres, liés par exemple à la sécurité ou la pérennité des ressources. Le tableau 2.1 met en évidence plusieurs différences fondamentales entre les modèles client-serveur et P2P. D'une manière générale, on constate que le modèle client-serveur est associé à un fonctionnement, un environnement et des ressources statiques et connues, alors que le modèle P2P est lié au concept de dynamicité. Dans cette section, nous passons en revue l'ensemble des caractéristiques que présente le modèle P2P ainsi que les différentes propriétés qu'elles lui confèrent.

[3] En anglais, *super-peers*
[4] http ://www.zeropaid.com

Critère	Modèle Client-Serveur	Modèle P2P
Gestion	Supervisé	Auto-organisé
Présence	Permanente	Ad Hoc
Accès au ressources	Recherche	Découverte
Organisation	Hiérarchique	Distribuée, Equitable
Mobilité	Statique	Mobile
Disponibilité	Dépendante du serveur	Indépendante des pairs
Nommage	Reposant sur le DNS	Indépendant
Modèle de programmation	RPC	Asynchrone

TAB. 2.1 – *Extrait de [112].* Comparaison des infrastructures client-serveur et P2P

2.2.1 Décentralisation

La décentralisation est *la* caractéristique du modèle P2P. Elle s'applique à différentes fonctions et aux trois sous-modèles. Dans le cas du modèle P2P centralisé, seules les ressources sont décentralisées mais les mécanismes de recherche et de localisation restent centralisés. Par contre, dans le cas du modèle pur, tout est décentralisé, des ressources aux mécanismes de découverte, localisation, sécurité, routage, ... Quelle que soit sa nature, cette décentralisation confère au modèle P2P un ensemble de propriétés que nous détaillons maintenant.

L'équilibre de la charge et du trafic : La première conséquence induite par la décentralisation sur les applications P2P, concerne l'équilibre de la charge et du trafic. Dans le cas du modèle client-serveur, le serveur de l'application concentre tout : les données, l'ensemble des mécanismes qui assurent l'accès à celles-ci et leur manipulation, mais aussi le trafic généré sur le réseau qui héberge le serveur. Au contraire, le modèle P2P permet de répartir et équilibrer au mieux la charge induite par l'exécution du service ainsi que le trafic généré sur le réseau.

Pour une ressource particulière, le pair qui l'héberge agit comme un serveur central. Une bonne répartition des ressources, nécessitant éventuellement l'utilisation de mécanismes de dissémination et de duplication [26], permet d'éviter que le système bascule dans un fonctionnement client-serveur avec des pairs qui ne sont pas dédiés à cette fonction.

Le passage à l'échelle : Le bon équilibre de la charge et du trafic des services P2P se répercute directement sur le passage à l'échelle des applications qui, comparativement au modèle client-serveur, s'en trouve fortement amélioré. Par exemple, les applications P2P de partage de fichiers comptent un très grand nombre de participants et fonctionnent sans aucun problème. D'après une étude menée en 2001 [143], Gnutella [133] compte en moyenne dix mille participants simultanés et OceanStore [93], une application P2P de stockage de fichiers, permet de gérer 10^{10} utilisateurs stockant au total plus de 10^{14} fichiers.

Le modèle P2P centralisé possède lui aussi un très bon passage à l'échelle car il ne centralise pas les ressources mais un index de références. Ainsi la charge et le trafic qui lui sont attribués sont acceptables et permettent un bon passage à l'échelle des applications qui reposent sur ce modèle. Les deux exemples-phares qui ont révélé cette bonne propriété sont Napster [147], qui permit à plusieurs dizaines de milliers d'usagers d'utiliser simultanément son service, et Seti@Home [8], une application de calcul distribué qui compte plusieurs milliers d'utilisateurs simultanés.

La répartition des coûts : Déployer une infrastructure centralisée de services en réseaux qui

puisse prendre en compte des milliers d'utilisateurs repartis sur tout l'Internet, a un coût qui peut être lourd pour l'organisation qui la déploie. Ce coût se répartit sur l'ensemble du cycle de vie de l'infrastructure et comprend la conception, l'achat d'équipements, la mise en œuvre, la supervision, la maintenance, la formation des usagers, la mise à jour des composants logiciels, ... Le modèle P2P permet de réduire fortement ce coût par l'utilisation de machines situées en bordure de l'Internet. Ces machines appartiennent toutes à des propriétaires différents qui peuvent être par exemple des particuliers, des universités, des entreprises ou des administrations et qui sont toutes achetées, mises en œuvre et maintenues par ces différentes organisations. L'utilisation du modèle P2P permet donc à un fournisseur de service de réduire fortement les coûts d'équipements et, au delà de l'aspect financier, de concentrer son action sur l'aspect logiciel de l'infrastructure qu'il propose.

La tolérance aux fautes : La dernière propriété induite par la décentralisation concerne la tolérance aux fautes. Dans l'architecture client-serveur la disponibilité d'un service repose intégralement sur celle du serveur. Si celui-ci s'écroule, le service qu'il fournit devient indisponible. Dans un contexte P2P, il n'existe potentiellement aucun point central de faute. Si un pair disparaît, le service continuera d'être fourni par ceux qui restent. En d'autres termes, la disponibilité d'un service n'est plus liée aux pairs mais à la communauté de pairs qui le fournissent.

2.2.2 Auto-organisation

L'absence d'élément central dans les applications P2P nécessite la mise en place de mécanismes d'auto-organisation qui permettent à une communauté de délivrer son service quels que soient les allers et venues des pairs et la disponibilité des ressources. Cette auto-organisation couvre plusieurs aspects qui peuvent être fonctionnels, communautaires et topologiques.

Aspect fonctionnel : Le modèle P2P hybride montre clairement que certains pairs peuvent se charger de l'exécution de fonctions particulières, segmentant ainsi une communauté en pairs spécialisés. Jxta, que nous présentons dans la section 2.3.2 est une bonne illustration d'organisation fonctionnelle avec l'utilisation de pairs minimaux qui sont de simples consommateurs d'une communauté, de pairs simples qui n'ont pas de rôle particulier, de pairs de rendez-vous qui assurent les fonctions de découverte et de localisation, et de pairs relais qui s'occupent du routage. En outre, les applications P2P possèdent souvent la capacité d'assigner ou de supprimer par elles-mêmes les fonctions attribuées aux pairs pour garantir le bon fonctionnement de son service.

Aspect communautaire : Ensuite, les applications P2P sont capables de regrouper leurs pairs par centres d'intérêts communs créant ainsi des communautés qui peuvent elles-mêmes contenir des sous-communautés. On trouve ce type d'organisation dans les services de communication et d'échange de données personnelles telles que Jabber[5] [111] qui regroupe les pairs en fonction des sujets sur lesquels ils souhaitent échanger comme par exemple, le sport, le cinéma ou la politique.

Aspect topologique : Le dernier aspect pour lequel les applications P2P proposent des mécanismes d'organisation concerne la topologie. La partie droite de la figure 2.1 présente quelques applications P2P classées en fonction de leur organisation topologique. De manière générale, il existe deux grandes classes de topologies P2P qui sont les topologies

[5]http ://www.jabber.org

structurées et non structurées. Les topologies structurées sont construites à l'aide d'un algorithme reposant souvent sur un modèle mathématique ou un graphe particulier. Dans le chapitre 3, nous présentons les tables de hachage distribuées qui sont des infrastructures de routage et de localisation pour les services P2P et qui sont organisées, pour la plupart d'entre elles, selon une topologie structurée. Les topologies non structurées, quant à elles, ne respectent aucune règle de construction particulière. Gnutella [84], dans sa première version[6], reposait sur une topologie non structurée. On peut néanmoins remarquer que les topologies non-structurées, du fait des différences de comportement des pairs, peuvent tendre au fil du temps à s'organiser selon une topologie de type *Small World* [110], un modèle topologique comportant quelques nœuds à fort degré de connexion et de nombreux autres à faible degré de connexion et connectés à ces premiers [74]. La figure 2.3 illustre ce propos avec deux représentations de topologies Gnutella. La première (figure 2.3.a) montre une topologie non structurée et la seconde (figure 2.3.b)[7] en montre une autre qui tend à s'organiser selon le modèle *Small World*.

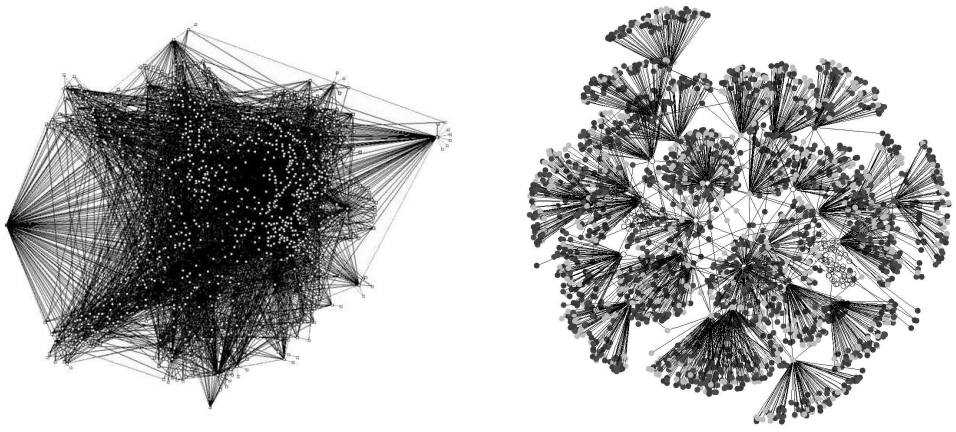


FIG. 2.3 – Topologies Gnutella. (a) *Extrait de [81]* Topologie non-structurée. (b) Topologie s'approchant du modèle *Small World*.

2.2.3 Connectivité Ad Hoc

Le modèle P2P se caractérise par une connectivité intermittente des pairs qui le composent. Cette nature Ad Hoc est principalement due à deux phénomènes. Le premier concerne le comportement des usagers qui utilisent les services P2P. Les pairs sont en effet exécutés dans un cadre de travail ou personnel sur des machines d'utilisateurs qui peuvent se connecter et se déconnecter de manière spontanée et donc imprévisible. Le second phénomène concerne la mobilité. Les ordinateurs portables munis d'interfaces de communication sans fil sont de plus en plus courants. Les usagers disposant de cette technologie ont souvent un comportement nomade, se trouvant certaines fois sur des sites qui permettent de se connecter à l'Internet, et d'autres fois pas. En outre, la manière dont ils atteignent une passerelle de connexion varie. Elle peut être

[6] à savoir, la version 0.4 du protocole
[7] Source : http ://www.cs.berkeley.edu/∼boonloo/research.html

directe ou nécessiter le routage des données à travers différents terminaux mobiles qui forment un réseau Ad Hoc.

Cette présence dynamique des pairs se répercute directement sur la disponibilité des ressources qui se trouve être variable. Pour garantir une bonne disponibilité des ressources offertes à une communauté, les infrastructures P2P doivent ainsi mettre en place des mécanismes de duplication et de synchronisation qui puissent pallier ce problème. En outre, l'absence d'une ressource ou d'un pair censé être présent ne doit pas être considéré comme une faute. D'ailleurs, dans Tapestry [170], une infrastructure de routage et de localisation de ressources, l'absence de réponse d'un pair inscrit dans une table de routage n'entraîne pas son retrait de la table. L'infrastructure tente de contacter le pair plusieurs fois. Après plusieurs tentatives, si aucune réponse n'est donnée, le pair est supprimé de la table mais une référence est tout de même conservée.

2.2.4 Un réseau virtuel

Les pairs participant à un service P2P forment souvent un réseau virtuel, appelé *overlay* [49] construit au-dessus de la couche transport[8]. Un exemple d'*overlay* est représenté sur la figure 2.4. Généralement, un tel réseau virtuel présente une propriété de transparence qui s'applique à plusieurs points [28]. Tout d'abord, il permet de faire abstraction des différences de nature des pairs. Une communauté peut être composée de pairs dont les caractéristiques diffèrent sur les plans (1) matériel, avec par exemple des stations de travail, des téléphones mobiles, des assistants personnels, ou un *cluster* de machines, (2) logiciel, au niveau des systèmes d'exploitation et langages de programmation, et (3) de la communication, avec l'utilisation de technologies et de piles de protocoles différentes. La transparence s'applique aussi sur le routage effectué au niveau sous-jacent : deux voisins de la topologie virtuelle ne le sont pas forcément physiquement : ils peuvent être situés dans des espaces physiques et sur des réseaux différents. L'*overlay* rend ainsi transparent le routage effectué au niveau physique. Enfin, concernant les ressources, un *overlay* rend transparent leur accès, leur localisation et leur duplication. Lorsqu'un pair accède à une ressource, il ne sait pas si cette ressource est locale ou distante. Dans le cas où la ressource est distante, il n'a pas de connaissance de l'hôte qui l'héberge, et si elle est dupliquée, il ne sait pas à quel réplica il accède.

L'utilisation d'un *overlay* nécessite toutefois la mise en œuvre des mécanismes de nommage et routage dédiés. Les pairs ne sont donc plus représentés par leur adresse de niveau transport ou adresse physique mais par un identifiant défini dans le cadre de l'*overlay*. En outre, pour pouvoir découvrir et accéder à des ressources, des services de routage, calcul de routes, découverte et accès sont mis en place. Dans le chapitre 3, nous nous intéressons particulièrement à cet aspect du modèle P2P.

Remarque : Un réseau P2P ne constitue par obligatoirement un *overlay*. Des applications reposant sur les protocoles NNTP[9] [87] ou RIP [104] sont construites selon le modèle P2P en ce sens qu'elles ne présentent aucune centralisation et que chacun de leurs élements agit comme un client et un serveur sans pour autant constituer un réseau virtuel au-dessus du réseau IP.

2.2.5 Anonymat

L'anomymat est une fonction qui permet de cacher son identité. Dans le cadre des systèmes distribués relatifs au partage de documents, six formes d'anonymat ont été recensées [37]. Elles

[8] Dans la suite de ce manuscrit, par abus de langage, on utilise le terme de réseau physique pour désigner le réseau sous-jacent à l'*overlay*.

[9] Network News Transfert Protocol

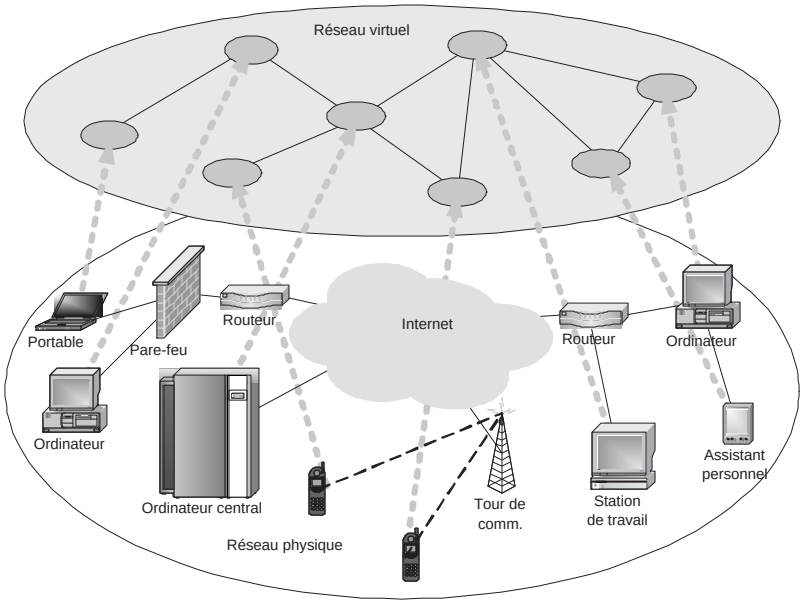


FIG. 2.4 – Exemple de réseau P2P qui forme un *overlay* au-dessus d'un réseau physique composé d'éléments hétérogènes.

concernent : l'auteur, l'éditeur, le lecteur, le serveur, le document et la requête.

L'anonymat est utilisé dans plusieurs applications qui voient dans le modèle P2P une infrastructure qui se prête particulièrement à cette fonction. Tout d'abord, Freenet [95] utilise une forme de routage qui garantit l'anonymat du serveur, de l'auteur et du lecteur en ne permettant à aucun nœud de savoir quelle est la source et la destination d'une requête. Ensuite, FreeHaven [38] et Publius [163] mettent explicitement en œuvre des mécanismes d'anonymat qui ont pour rôle principal d'assurer la persistance et la disponibilité de documents dans un environnement soumis à la censure.

Enfin, on trouve maintenant des infrastructures P2P dont le but est simplement de fournir un service d'anonymat à des applications P2P ou centralisées. Parmi les différentes propositions, nous en retenons trois qui sont Crowds [126], Morphmix [128, 127] et Tarzan [64] que nous présentons. Tarzan est une infrastructure qui rend anonyme la couche réseau IP. Pour ce faire, elle utilise un modèle P2P pur où les pairs forment des tunnels construits de manière pseudo-aléatoire. La manière dont les tunnels se forment est quasi-impossible à détecter et à compromettre et le trafic qu'ils transportent est crypté. De plus, chaque pair est associé à quelques autres, appelés mimes, qui vont effectuer les mêmes opérations. Par ce biais, Tarzan empêche la détection de la source d'un message.

Dans cette section, nous avons présenté le modèle P2P. Nous l'avons tout d'abord défini de manière générale, puis nous avons affiné cette définition en distinguant trois sous-modèles, relatifs au niveau de décentralisation de ses composants, qui sont les modèles P2P pur, hybride et centralisé. Nous avons ensuite mis en évidence les caractéristiques induites par le modèle P2P qui concernent la décentralisation, l'auto-organisation, la connectivité Ad Hoc, l'utilisation d'un *overlay* et l'anonymat.

2.3 Applications

Dans cette section, nous passons en revue les différents domaines d'application du modèle P2P. Nous nous intéressons en particulier à Jxta qui est une des plates-formes P2P sur laquelle nous avons déployé et validé nos contributions.

2.3.1 Classification des différents domaines

Actuellement, on distingue quatre grands domaines d'applications couverts par le modèle P2P [112]. Nous les avons représentés sur la figure 2.5 qui les classifie. Il s'agit des plates-formes de développement, du partage et la distribution de contenu, de la collaboration et du calcul distribué. Nous présentons maintenant chacun d'eux.

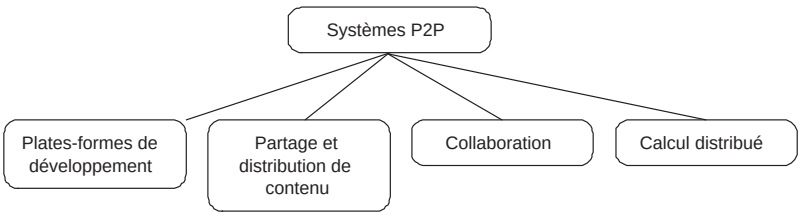


FIG. 2.5 – *Extrait de [112].* Classification des applications P2P

Les plates-formes de développement

Les applications de partage de fichiers qui ont popularisé le modèle P2P ont aussi mis en évidence un besoin fort d'interopérabilité pour ce type d'application. C'est pourquoi, plusieurs propositions de plates-formes génériques de développement qui permettent cette interopérabilité ont vu le jour. Celles-ci permettent aux développeurs d'applications de s'abstraire des mécanismes de bas niveau du modèle P2P et de proposer des applications toutes construites sur les mêmes fondements.

Jxta [68], une initiative de Sun présentée dans la section 2.3.2, est une des premières propositions de plate-forme générique et est actuellement une des plus complètes et déployées. Microsoft, de son côté propose deux infrastructures pour le développement des applications P2P. La première est une extension de la plate-forme de développement de services *web* .NET[10] [120] qui présente une manière d'intégrer des services P2P à cette infrastructure. Cette proposition s'apparente plus à l'adaptation d'une plate-forme initialement dédiée à un fonctionnement centralisé plutôt qu'à une véritable proposition de plate-forme qui intégre les mécanismes de base du modèle P2P. La seconde proposition s'appelle *Windows Peer-to-Peer* [34] et propose un ensemble de mécanismes utilisables par les concepteurs d'applications Windows pour développer des services P2P. Néanmoins, son lien étroit avec Windows XP et l'utilisation de TCP/IPv6 pour la communication restreignent fortement son champ d'utilisation.

Concernant l'IETF[11], la proposition de deux nouveaux protocoles qui intègrent des mécanismes de communication P2P a été faite. Ce sont BEEP [135] et APEX [136], des protocoles respectivement de niveau session et application. Le premier permet d'établir une session P2P entre deux entités distantes et prend en charge des mécanismes tels que l'authentification, le

[10] www.microsoft.com/net/
[11] www.ietf.org

cryptage, la gestion de profils, ... Le second, considère des problèmes de plus haut niveau tels que la découverte de services, la gestion de présence, l'adressage de niveau applicatif, ... Par le biais de ces deux protocoles, les nouvelles applications peuvent fonctionner indifféremment selon un modèle client-serveur ou P2P et peuvent s'abstraire de certains mécanismes génériques à un grand nombre de services.

Enfin, concernant les propositions faites par des instituts universitaires, on trouve principalement deux propositions de plate-forme de développement P2P. Shine [169] est une infrastructure générique pour les applications de type *socialware*, c'est à dire fondées sur la communication et le partage de contenu. Enfin, Anthill [12] repose sur une adaptation du comportement de communautés biologiques comme les fourmis ou les abeilles pour construire une plate-forme générique composée d'agents mobiles.

Le partage et la distribution de contenu

Le modèle P2P connaît son succès actuel grâce aux applications de partage de fichiers. Initié par Napster [147], ce type d'application consiste à créer des communautés de pairs qui partagent des fichiers qui sont stockés sur leur machine. Les protocoles et implantations d'applications de partage de fichiers sont nombreuses[12]. Parmi les plus connues, on trouve Gnutella, Morpheus, Kazaa, E-Mule, Bit-Torrent [158]. Souvent aucune interopérabilité n'existe entre celles-ci. Pour pallier ce problème, on trouve maintenant des applications qui implantent plusieurs protocoles et permettent aux usagers de se connecter à plusieurs communautés, augmentant ainsi le volume de données accessibles.

Un des principaux problèmes mis en évidence par les applications de partage de fichiers concerne le *free riding* [4], un comportement qui consiste à télécharger des données sans en partager aucune. Pour le résoudre, les applications de partage de fichiers les plus récentes mettent en place des mécanismes apparentés à l'auto-gestion. Les principaux reposent sur un classement régulier des pairs qui dépend de leur comportement : les pairs les plus stables, fiables et généreux vont ainsi voir leurs recherches de données être traitées plus efficacement et pouvoir télécharger plus de données, plus rapidement.

Le second type d'application relatif aux fichiers et à leur accès par le biais du modèle P2P concerne le stockage de fichiers. On trouve plusieurs applications telles que CFS [30], PAST [50] ou OceanStore [93, 130] qui sont toutes construites sur des tables de hachage distribuées et qui proposent de construire un système de fichiers de type Unix qui soit distribué parmi une communauté de pairs. L'objectif est de fournir un service similaire à NFS qui ne nécessite aucune architecture centralisée.

La collaboration

Les applications P2P collaboratives proposent à des communautés d'usagers des services de communication et d'échange de données. Différents moyens de communiquer sont souvent offerts, avec notamment le *chat*, la messagerie instantanée, la voix et la vidéo par le biais de la visio-conférence ou des web-cams. L'échange de données, selon le contexte d'utilisation, va des photos ou vidéos personnelles aux documents d'un projet de travail. Deux types d'usagers sont visés par ce type d'application qui sont le particulier et les entreprises. On trouve aujourd'hui une multitude d'applications qui ciblent ces deux catégories, avec par exemple Jabber [140,

[12]http ://www.zeropaid.com est un portail qui recense les différentes implantations actuelles

141], ICQ[13] ou Skype[14] pour la communication et Groove, Ikimbo ou Ocolus pour le travail collaboratif.

Le calcul distribué

La possibilité de pouvoir utiliser la puissance de calcul de millions de machines connectées à l'Internet intéresse fortement les acteurs de la communauté du calcul distribué. En 1999, l'université de Berkeley lance le projet Seti@Home[15] [8] qui a pour objectif d'analyser des données transmises par des récepteurs du projet SETI qui écoutent l'univers afin de détecter une quelconque forme de communication. Etant donné le nombre colossal de données à analyser, l'équipe propose de développer une application assimilable à un économiseur d'écran qui, lorsque l'ordinateur sur lequel elle est hébergée est oisif, télécharge un morceau de données sur un serveur dédié, les analyse et lui retourne le résultat. Cette application connaît un très grand succès et est actuellement utilisé par des milliers d'utilisateurs, motivés par des classements réguliers de participations et par la possibilité d'être celui qui aura traité *le* bloc qui contient un extrait de communication extra-terrestre. En outre, des clones sont aujourd'hui utilisés dans de nombreux autres domaines[16]. On considère ainsi Seti@Home comme le précurseur d'une problématique de recherche à part entière qui concerne la manière de distribuer un calcul sur une infrastructure P2P.

Actuellement, la distribution par un serveur central d'un calcul sur une grille est une tâche maitrisée et mise en œuvre dans plusieurs infrastructures telles que Globus [60] ou eXtremWeb[17] [55, 20]. Les travaux actuels portent donc maintenant sur la manière de faire coïncider les domaines du *Grid Computing*[18] et du P2P qui présentent de nombreuses différences. Celles-ci sont mises en évidence dans le tableau 2.2. D'une manière très générale on s'aperçoit que, bien que ces deux modèles reposent sur des infrastructures distribuées, leur contexte de mise en œuvre est plutôt opposé : l'environnement est statique, clos et sûr dans le cas du *Grid Computing*, et potentiellement dynamique, ouvert, et non sûr pour le P2P. Il est donc nécessaire de concevoir et mettre en œuvre de nouveaux mécanismes qui permettent au calcul distribué de s'intégrer dans une infrastructure P2P.

Critère	**Pair à pair**	**Grid**
Rôle des entités	Client et serveur	Grid - serveur
Opérations	Initiées par le client	Initiées par le serveur
Participants	Volontaires à présence volatile	Prédéterminés et enregistrés
Fiablilité	Partielle : pairs inconnus	Garantie
Contrôle	Décentralisé	Centralisé
Connectivité	Occasionnelle, faible capacité	statique, haut débit

Tab. 2.2 – *Extrait de [159].* Comparaison des infrastructures P2P et *Grid*

Parmi les nombreuses propositions actuelles, nous présentons celles qui nous semblent être les principales.

[13] I seek you
[14] http ://www.skype.com
[15] setiathome.ssl.berkeley.edu
[16] Genome@home, ...
[17] http ://www.lri.fr/∼fedak/XtremWeb/
[18] http ://www.gridcomputing.com

La technologie Java qui, après avoir été fortement contestée dans la communauté du calcul distribué, semble être maintenant reconnue pour sa portabilité, sa facilité de programmation et son extensibilité. On trouve actuellement trois propositions qui reposent sur cette technologie [161, 61]. L'une d'entre elles est ProActive[19] [21], une proposition de conception et d'implantation de bibliothèque Java pour le développement d'applications parallèles ou distribuées et le calcul concurrent sur les grilles. Elle repose sur un modèle de programmation MIMD[20] et sur la notion d'objets actifs qui possèdent une activité propre. Les principaux avantages de ProActive résident dans l'utilisation de Java pour les raisons citées précédemment, la transparence de la bibliothèque qui n'ajoute aucune syntaxe particulière au langage, facilitant ainsi le travail aux développeurs d'applications qui l'utilisent, et son ouverture vers des extensions.

P2P Grid [159] est une proposition qui definit une manière unifiée de représenter les ressources inhérentes au calcul distribué comme les processeurs ou les systèmes d'exploitations des machines disponibles. Cette représentation est fondée sur CIM[21] pour la formalisation des informations et XML pour leur représentation. Ensuite, ce travail propose d'utiliser un modèle hybride, similaire à la dernière version de Gnutella qui interconnecte des pairs à des super-pairs, eux-mêmes connectés entre eux. La recherche et l'organisation de ressources s'effectue par le biais des super-pairs qui consultent avant tout les pairs qui leur sont connectés puis, si aucune ressource satisfaisante n'est trouvée, d'autres super-pairs. Différentes simulations ont montré le bon comportement de cette infrastructure dans des communautés comptant jusqu'à 2500 nœuds. Enfin, Paradropper [48] est une architecture distribuée reposant sur un modèle pur qui construit une topologie virtuelle de type *Small World* [110]. Elle est avant tout une proposition d'organisation topologique qui semble être adéquate pour le calcul distribué, mais aussi un algorithme d'ordonnancement totalement réparti parmi les pairs participants.

Nous venons de passer en revue les quatre domaines d'application du modèle P2P. Nous avons mentionné différentes applications de chacun de ces domaines et pointé les différences qu'elles présentent. Nous détaillons maintenant Jxta qui est la plate-forme sur laquelle nous avons travaillé.

2.3.2 Le projet Jxta

En 2001, Sun lance un projet de conception et de développement d'une plate-forme générique P2P appelée Jxta[22] [68, 119]. Celle-ci permet aux développeurs de s'abstraire des mécanismes de base relatifs au modèle P2P et, par le biais de concepts et protocoles communs, elle autorise l'interopérabilité des applications. Différentes implantations de la plate-forme existent, écrites dans différents langages (C, Java, Python, ...), pour différents terminaux (Téléphone portable, PDA, PC, ...) interagissant à travers différents moyens de communication (Bluetooth, UMTS, ...). Actuellement, le projet Jxta connaît sa deuxième version [157] et implique une large communauté de développeurs qui travaillent sur des projets applicatifs couvrant de nombreux domaines (partage de contenu, collaboration, communication, ...).

Plus formellement, le projet Jxta est en fait une proposition de concepts génériques qui s'instancient à travers différents protocoles de communication et une architecture fonctionnelle. Jxta est la principale plate-forme sur laquelle nous avons mis en œuvre nos travaux de recherche.

[19] http ://www-sop.inria.fr/oasis/ProActive
[20] Multiple Instructions Multiple Data
[21] cf. section 5.2 du chapitre 5
[22] http ://www.jxta.org

C'est pourquoi, dans la suite de cette section, nous détaillons particulièrement les concepts de Jxta et sa pile de protocoles.

Les concepts

Pour pouvoir mettre en œuvre une plate-forme générique, applicable à n'importe quel type de service P2P, Jxta a tout d'abord identifié les différents concepts induits par ce modèle [155]. Ceux-ci sont :

Le pair : un pair est un système muni d'une interface de communication qui implante la pile de protocoles Jxta. L'hôte qui héberge le pair peut être de différente nature, qui va du téléphone portable au super-calculateur formé par un *cluster* de machines. Durant son cycle de vie, un pair peut occuper différentes fonctions qui lui confèrent un rôle particulier. Jxta fait ainsi la distinction entre quatre types de pairs qui sont :

- Les pairs minimaux : Ce sont souvent des terminaux disposant de faibles ressources qui reçoivent les messages mais ne participent pas activement à la communauté à laquelle ils appartiennent.
- Les pairs simples : Ce sont les pairs classiques qui constituent une communauté. Ils exécutent des applications P2P et y participent activement.
- Les pairs de rendez-vous : Ce sont des pairs simples qui agissent comme points de rendez-vous pour d'autres pairs et assurent ainsi la découverte de ressources.
- Les pairs relais : Leur fonction entre dans le cadre du routage notamment pour permettre la communication des pairs qui sont situés derrière un pare-feu ou qui utilisent des couches transport différentes.

La communauté : Une communauté Jxta, appelé *Peergroup*, est constituée d'un ensemble de pairs qui exécutent un ensemble de services communs. La création d'un *Peergroup* peut être motivée par plusieurs raisons qui sont : le regroupement des pairs autour d'intérêts communs, l'établissement d'un environnement sécurisé dont l'accès est protégé par un mécanisme d'authentification et le contenu crypté, et la mise en place d'un procédé de surveillance d'autres pairs.

Les *Peergroups* s'organisent selon une structure arborescente et un pair peut appartenir à plusieurs groupes à la fois[23] : le groupe racine auquel chaque pair appartient par défaut est le *NetPeerGroup* qui contient divers *Peergroups* qui eux-mêmes peuvent contenir des sous-groupes, ... Par le biais de ce concept, Jxta autorise le partitionnement d'une communauté P2P en sous-ensembles qui exécutent des services communs.

La communication : La communication entre les pairs Jxta est assurée par le *Network Transport*, un service constitué de trois éléments qui sont les *pipes*, les *endpoints* et les messages. Un *pipe* représente un canal de communication unidirectionnel virtuel par lequel des données sont échangées au sein d'un *Peergroup*. Jxta distingue deux types de *pipes* qui sont les *pipes unicasts* qui relient une ou plusieurs sources à une destination et les *pipes* propagés qui relient une source à plusieurs destinations. Chaque extrémité d'un *pipe* est représentée par un *endpoint* qui représente l'interface entre le réseau virtuel Jxta et le niveau transport ou applicatif sous-jacent[24] par laquelle la communication est effectuée. Enfin, les données échangées entre des pairs sont encapsulées dans des messages qui sont des conteneurs gé-

[23] Toutefois, un pair ne peut être actif que dans un seul groupe à la fois

[24] Jxta peut en effet communiquer par le biais de protocoles de transport tels que TCP, UDP ou de protocoles de niveau applicatif comme HTTP ou SMTP

nériques de données. La figure 2.6 illustre les trois concepts qui constituent le principe de communication de Jxta.

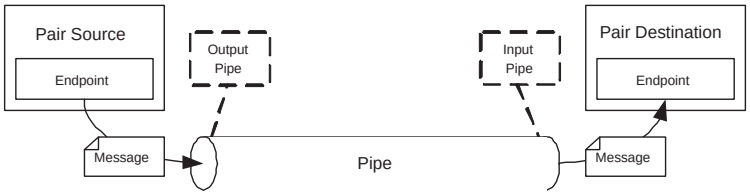


FIG. 2.6 – Illustration du modèle de communication Jxta fondé sur des *pipes*, des *endpoints* et des messages

Le service : La finalité de Jxta est de permettre à des pairs de participer et profiter de services d'une manière totalement distribuée. Deux types de services sont reconnus par la plateforme : les services de groupes et les services de pairs. Les services de groupes sont des services offerts aux pairs participant au *Peergroup* qui les héberge. Ils ont une durée de vie liée à celle du groupe et pour rejoindre le groupe dans lequel les services s'exécutent, un pair doit obligatoirement en posséder une implantation. Par contre, les services de pairs sont offerts par un pair particulier. Leur existence est donc liée à la présence du pair qui les héberge.

L'annonce : Ce concept est l'un des plus forts de ceux proposés par Jxta. Il stipule que chaque ressource et élément participant au monde Jxta est formalisé par une annonce qui est traduite sous la forme d'un document XML. Ainsi, les pairs, les *Peergroups*, les services, les pipes, les messages, les *endpoints*, ainsi que les autres ressources Jxta sont formalisés à travers ce concept qui unifie la représentation des ressources. Découvrir un pair revient alors à découvrir son annonce.

Le listing 2.1 présente un extrait d'annonce de pair. Le type de ressource représenté par le document, à savoir une annonce de pair, est spécifié à la ligne 2 avec le mot clé `PA`[25]. L'identifiant du pair est spécifié à la ligne 5 entre les balises `<PID>` et `</PID>`. De la même manière, l'identifiant du groupe auquel appartient le pair est indiqué à la ligne 8, entre les balises `<GID>` et `</GID>`. Pour finir, des informations textuelles constituées du nom (ligne 11) et d'une description (lignes 14 et 15) sont fournies.

Le nommage : Ce dernier concept permet d'identifier de manière unique les ressources et les éléments présents dans le monde Jxta. Il est totalement indépendant des l'infrastructures qui hébergent les pairs, comme leur système d'exploitation et les réseaux par lesquels ils communiquent. Chaque idenfiant est composé d'une chaîne de 17 ou 30 caractères qui commence par `urn:jxta:uuid-`. Dans le listing 2.1, en en trouve deux exemples aux lignes 5 et 8.

La pile de protocoles

Les concepts définis par Jxta et présentés ci-dessus sont instanciés et manipulés par la plateforme par le biais de six protocoles [154] dont la pile est représentée sur la figure 2.7. En voici une description succincte.

[25] *Peer Advertisement*

Listing 2.1 – Extrait d'une annonce de pair Jxta

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jxta:PA>
<jxta:PA xmlns:jxta="http://jxta.org">
    <PID>
        urn:jxta:uuid-59616261646162614A7874615032503...
    </PID>
    <GID>
        urn:jxta:uuid-A737CDF37EF44C568A3492FB0433209502
    </GID>
    <Name>
        Guillaume's peer
    </Name>
    <Desc>
        Platform Config Advertisement created by :
        net.jxta.impl.peergroup.DefaultConfigurator
    </Desc>
...
</jxta:PA>
```

***Endpoint Routing Protocol* :** Lors de l'envoi de messages, ce protocole permet à un pair de trouver et établir un chemin jusqu'à un pair distant ; ce chemin pouvant être composé de pairs relais intermédiaires.

***Peer Resolver Protocol* :** Il est le protocole à travers lequel les pairs peuvent émettre des requêtes et recevoir des réponses. Les requêtes sont génériques et peuvent être initiées par différents protocoles de niveau supérieur. La portée des recherches est limitée au groupe auquel le pair appartient.

***Rendezvous Protocol* :** C'est le protocole qui gère les relations entre les pairs simples et les pairs de rendez-vous. Afin de propager les requêtes émises par différents pairs d'une communauté, les pairs simples se connectent à des pairs de rendez-vous qui sont eux-mêmes connectés à d'autres pairs de rendez-vous. Cette interconnexion des pairs de rendez-vous forme une table de hachage distribuée appelée SRDI [156].

***Peer Discovery Protocol* :** Il permet à des pairs de publier et rechercher des annonces chez d'autres pairs. Par le biais de ce protocole, les pairs peuvent découvrir et localiser les ressources et les éléments présents dans une communauté.

***Peer Information Protocol* :** Ce protocole, orienté vers la gestion, permet à un pair d'obtenir des informations sur d'autres pairs de la communauté. Actuellement, ce protocole est utilisé dans le cadre d'un projet de monitorage de la plate-forme, nommé MMP, que nous présentons dans la section 4.3.3.

***Pipe Binding Protocol* :** Ce dernier protocole est utilisé dans le cadre de la création des *pipes* pour effectuer le lien entre deux ou plusieurs *endpoints*.

2.4 Synthèse

Le modèle P2P est un modèle distribué où les entités appelées pairs jouent le rôle de client et serveur. Différents niveaux de décentralisation permettent de scinder le modèle P2P en trois sous-modèles qui sont le modèle P2P pur, hybride et centralisé. Le modèle pur est constitué de pairs strictement équivalents. Le modèle hybride utilise des super-pairs qui présentent des fonctions avancées. Enfin le modèle centralisé repose sur un serveur dédié qui assure les fonctions de découverte et localisation.

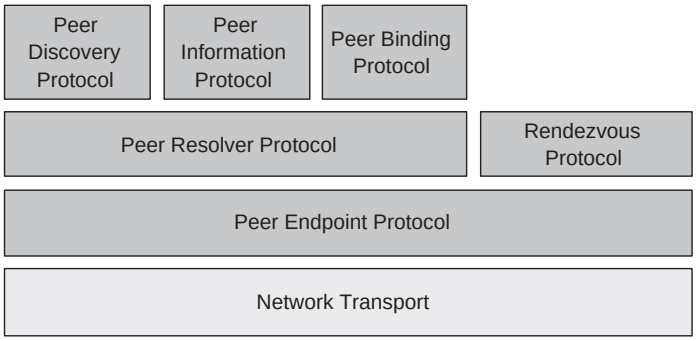


FIG. 2.7 – *Extrait de [165].* La pile de protocoles Jxta

Les caractéristiques du modèle P2P sont nombreuses. La principale est la décentralisation qui confère aux applications P2P et, comparativement au modèle client-serveur, un bon equilibre de la charge, un meilleur passage à l'échelle, une répartition des coûts de mise en œuvre et maintenance du service offert et une bonne tolérance aux fautes. Les autres caractéristiques de ce modèle sont (1) l'auto-organisation qui permet aux communautés de pairs de délivrer leur service de manière autonome, (2) la connectivité Ad Hoc des pairs qui induit une dynamicité des données et des ressources, (3) l'utilisation d'un *overlay* qui abstrait les caractéristiques physiques des éléments, et (4) l'anonymat qui permet aux pairs d'agir librement sans révéler leur identité.

Enfin, les applications P2P se regroupent en quatre catégories qui sont les plates-formes de développement, le partage et la distribution de contenu, la collaboration et le calcul distribué. Pour chacun de ces domaines, nous avons présenté les applications-phares. Nous avons particulièrement détaillé Jxta, une plate-forme générique pour le développement d'applications P2P proposée par Sun, qui permet à des développeurs de s'abstraire des mécanismes de base inhérents au modèle P2P. Jxta est la plate-forme que nous avons utilisée pour la mise en œuvre de certaines de nos propositions.

Chapitre 3

Les tables de hachage distribuées

Sommaire

Dans ce chapitre, nous présentons les tables de hachage distribuées (DHT[1]) qui sont des infrastructures de routage et de localisation reposant sur un modèle P2P pur. Dans un premier temps, nous présentons leur principe général de fonctionnement. Ensuite, nous distinguons plusieurs classes de DHTs et pour chacune d'entre elles, nous détaillons une proposition. La dernière partie de ce chapitre est dédiée à la comparaison des travaux actuels et à la présentation des nouvelles pistes qui sont explorées.

3.1 Problème de localisation et de routage

L'utilisation du modèle P2P, avec ses caractéristiques de décentralisation et de dynamicité pose un problème très simple : *comment découvrir et accéder à des ressources dans un tel*

[1]Distributed Hash Table

contexte ? En effet, étant donné le comportement allant-venant des pairs, il est très difficile d'obtenir une vue globale de l'ensemble des ressources d'une communauté P2P mais surtout de savoir où se situe une ressource particulière. Pour répondre à ce problème, trois solutions ont été envisagées.

La première repose sur l'utilisation d'un serveur central de méta-données. Comme le montre l'exemple de la figure 3.1.a, celui-ci maintient à jour une liste des ressources couramment accessibles et leur associe une ou plusieurs adresses de machines qui les hébergent. Napster [147] repose sur ce principe et montre clairement les avantages de cette méthode. En effet, les serveurs et réseaux actuels disposent de capacités suffisantes pour effectuer aisément ce genre de tâches, même à grande échelle et, d'un point de vue des performances, cette méthode est peu coûteuse ; l'accès à une ressource particulière n'implique que deux machines à travers deux requêtes : le serveur de méta-données pour la découverte et la machine hôte pour l'accès à la ressource. Par contre, son principal inconvénient est lié au caractère indispensable du serveur de méta-données : si celui-ci disparaît, le service devient indisponible alors que les ressources sont toujours présentes.

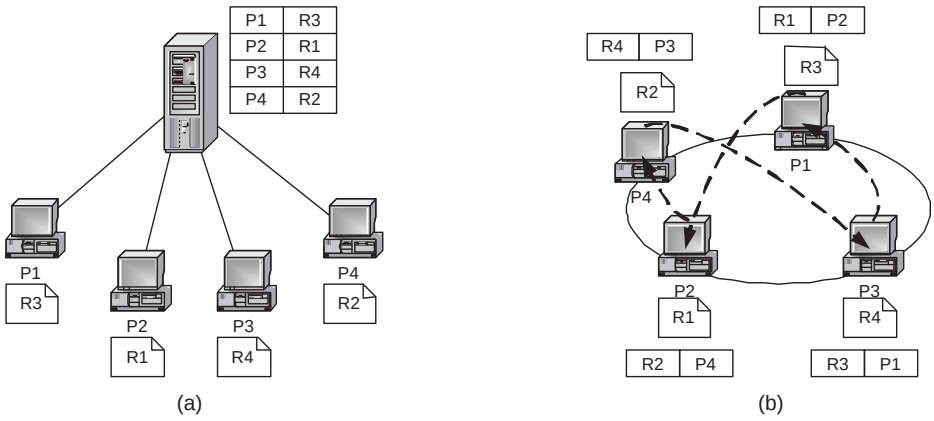


FIG. 3.1 – La découverte de ressources. (a) Par annuaire centralisé. (b) Par une table de hachage distribuée.

Pour répondre à ce problème, une seconde solution, de nature distribuée et reposant sur une méthode d'inondation, a été envisagée. Gnutella la met en œuvre et en montre les limites. Tout d'abord, de par l'utilisation d'un horizon limité, la présence d'une ressource dans une communauté ne garantit pas le succès de sa découverte. A ce propos, [158] a montré que, dans une communauté Gnutella de 10000 pairs statiques, pour une ressource donnée, seules 75% des requêtes qui lui étaient destinées fournissaient une réponse positive. Ensuite, le principe même d'inondation présente des performances médiocres ; cette même étude a montré qu'en moyenne, plus de 100 messages étaient nécessaires pour acheminer une requête de découverte de ressource. Le modèle d'inondation s'apparente donc à une première étape vers la distribution du processus de découverte de ressources mais son principe reste coûteux et non fiable.

Actuellement, la solution la plus étudiée pour découvrir des ressources dans un environnement distribué repose sur le principe d'une table de hachage distribuée [9]. La figure 3.1.b montre, à titre d'exemple, la manière dont la fonction de découverte d'un serveur central peut être distribuée parmi un ensemble de pairs. De cette manière, une application construite sur une telle infrastructure va pouvoir découvrir et accéder à des ressources de manière distribuée et fiable.

Dans la suite de ce chapitre, nous présentons le principe général d'une table de hachage distribuée et les propriétés qu'elle présente. Ensuite nous étudions en détail quelques DHTs actuelles et sur lesquelles nous avons travaillé. Dans un troisième temps, nous effectuons une analyse comparative de leurs propriétés et de leur comportement afin d'en pointer les différences. Enfin nous passons en revue les différents travaux qui proposent des améliorations et extensions à ces infrastructures.

3.2 Les tables de hachage distribuées

La première proposition de DHT est apparue en 2001 et fut proposée par Ratnasamy et al. Elle porte le nom de CAN [124] : *A Content Addressable Network*[2]. Ce nom est maintenant utilisé de manière générique pour désigner l'ensemble des infrastructures de type DHT [63]. Dans cette section, nous présentons les principes généraux, applicables à l'ensemble des DHTs actuelles, ainsi que leurs propriétés.

3.2.1 Principe général

Une table de hachage est une structure de données qui associe une clé à une ressource. Chaque clé de la table est le résultat d'une fonction de hachage appliquée à un élément de la ressource, comme par exemple, son nom. La fonction de hachage garantit que pour deux ressources différentes, les clés générées le seront aussi. Ainsi, l'unicité de la clé permet d'identifier et de retrouver de manière fiable la ressource à laquelle elle est associée. Dans le cadre des DHTs, des fonctions de hachage standard telles que SHA-1 [160] ou MD5 [134] sont utilisées et c'est le nom de la ressource qui est haché.

L'innovation apportée par les tables de hachage *distribuées* est la manière de distribuer une telle structure de données ; car, si dans le cas d'un modèle centralisé, il est simple de parcourir la table pour rechercher une donnée, dans le cas d'un modèle distribué comme le P2P, les questions de la manière de distribuer la table et de l'accès à ses entrées se posent. Le principe utilisé dans les infrastructures actuelles est le suivant : chaque identifiant de pair, comme par exemple son adresse IP ou bien le nom de machine ou d'utilisateur, est haché avec la même fonction de hachage que celle utilisée pour les ressources. Ensuite, la table est distribuée de manière à ce que les clés des ressources soient stockées sur les pairs dont le résultat de la fonction de hachage est le plus proche, d'après une fonction de distance donnée.

Pour illustrer ce principe, reprenons l'exemple illustré sur la figure 3.1.b. On peut y voir que le pair P_1 héberge la partie de la table correspondant à la ressource R_1 qui possède le même identifiant numérique, à savoir 1. De même, P_2 héberge l'entrée de la table pour R_2, et ainsi de de suite. De plus, on constate que chaque pair hébergeant une partie de la table de hachage, ne possède qu'une référence vers une ressource et non la ressource elle-même. Par exemple, le pair P_3 héberge physiquement la ressource R_4 et possède un pointeur vers P_1 qui héberge R_3, la ressource qui possède le même identifiant. De manière générale, on appelle le pair possédant une référence vers le pair hébergeant une ressource, la racine de cette ressource. Cette manière fiable et connue de distribuer la table de hachage change ainsi la manière de découvrir une ressource. Découvrir une ressource particulière revient à découvrir le pair d'identifiant le plus proche de celle-ci. Ceci explique le terme de *réseau à contenu adressable* [124, 63].

Pour effectuer cette recherche, un algorithme de découverte et de routage qui garantisse ce fonctionnement doit être mis en place. Les graphes de Caley [7] présentent des propriétés inté-

[2] Un réseau au contenu adressable

ressantes pour le routage dans les réseaux, et [123] a montré que la majorité des propositions actuelles reposent en fait des déclinaisons particulières de ces graphes. Parmi celles-ci, on trouve les graphes de De Bruijn [35], les réseaux Butterfly [149], les hypercubes, les graphes en anneau [65] ou les tores de dimension d. En outre, conjointement à cette famille de graphes, d'autres propositions reposent sur l'algorithme de Plaxton [122], les filtres de Bloom [15], ou le modèle *Small World* [89] pour organiser les pairs selon une topologie structurée et proposer un algorithme de routage fiable et efficace. Etant donné un ensemble de pairs participant à une DHT et une topologie reposant sur un modèle connu duquel on décline un algorithme de routage, la découverte d'une ressource s'effectue par approches successives, en minimisant à chaque saut la distance[3] comprise entre le nœud courant et le nœud source de la clé requise.

3.2.2 Propriétés

Les DHTs présentent de bonnes propriétés qui sont liées à l'utilisation d'un modèle P2P pur : aucun pair ne présente de rôle particulier ou central et chacun agit de manière strictement équivalente. On leur confère ainsi les propriétés suivantes [82, 117] :

La fiabilité : L'utilisation d'un algorithme de découverte et de routage qui garantisse que pour une clé donnée, il soit capable de déterminer le pair d'identifiant le plus proche, rend les DHTs fiables. Ceci, dans le sens où, dans des conditions statiques, une réponse négative à une requête signifie que la ressource requise n'est pas disponible dans la communauté.

La performance : Pour la plupart des DHTs, dans des conditions normales de fonctionnement, le nombre de sauts nécessaires à l'acheminement d'une requête s'exprime en $O(\log_B(N))$, avec B, la base des identifiants pour les pairs et les ressources. Ceci confère aux DHTs de très bonnes performances. Par exemple, dans une communauté de 10^6 pairs utilisant une base d'identifiants hexadécimale, la longueur moyenne d'une requête avoisine 5 sauts.

Le passage à l'échelle : Deux caractéristiques confèrent aux DHT une bonne propriété de passage à l'échelle. La première est liée au nombre moyen de sauts nécessaires au routage des requêtes qui reste petit même dans le cas de communautés comptant un grand nombre de participants. La seconde est relative aux tables de routage qui restent elles aussi d'une taille raisonnable en regard du nombre de participants, à savoir de taille constante $O(1)$ ou de taille logarithmique $O(\log(N))$.

L'équilibre de la charge et du trafic : L'utilisation de fonctions de hachage pseudo-aléatoires régulières pour les identifiants de pairs et de clés permet de créer des communautés équilibrées où les pairs ont statistiquement en charge une part égale de ressources à référencer. De plus, cet équilibre de la charge induit l'équilibre du trafic au sein de la communauté, dans l'hypothèse où chaque ressource est sollicitée de manière équivalente.

La tolérance aux fautes : De part l'absence de centralisation, qui exclut tout point central de faute, les DHTs présentent une bonne tolérance face à des suppressions aléatoires de nœuds. Les requêtes peuvent être acheminées même si une fraction de nœuds disparaît. Par contre, chaque nœud racine d'une ressource particulière s'apparente à un point central de fautes [170]. Des mécanismes de redondance sont souvent mis en place pour éviter l'inaccessibilité à une ressource pourtant présente dans une DHT.

Le coût de maintenance : La nature dynamique du modèle P2P nécessite la mise en place d'un processus de maintenance qui permette de garantir le bon fonctionnement d'une DHT.

[3] définie par une fonction particulière

Les données référencées par une DHT doivent être accessibles de manière fiable et celle-ci doit garantir un niveau de performance donné. Pour cela, le processus de maintenance, exécuté de manière asynchrone par chaque pair, vise à vérifier régulièrement que les entrées des tables de routage sont correctes, à savoir, qu'elles sont en accord avec les propriétés de la structure topologique, et que les pairs référencés sont joignables. Ces opérations de maintenance doivent, en outre, nécessiter le minimum de ressources de traitement local et de communication.

Dans cette section, nous avons présenté le principe général des DHTs ainsi que leurs propriétés. Au niveau des implantations, comme le montre la figure 3.2, il existe plusieurs grandes classes de DHTs qui se différencient principalement par le modèle topologique sur lequel elles reposent. En outre, au sein de chaque classe, il existe de nombreuses propositions de DHTs. Dans la suite de ce chapitre, nous allons présenter une DHT majeure pour chaque classe[4]. Nous nous attarderons en particulier sur Chord [153] et Pastry [137] qui sont des infrastructures sur lesquelles nous avons travaillé.

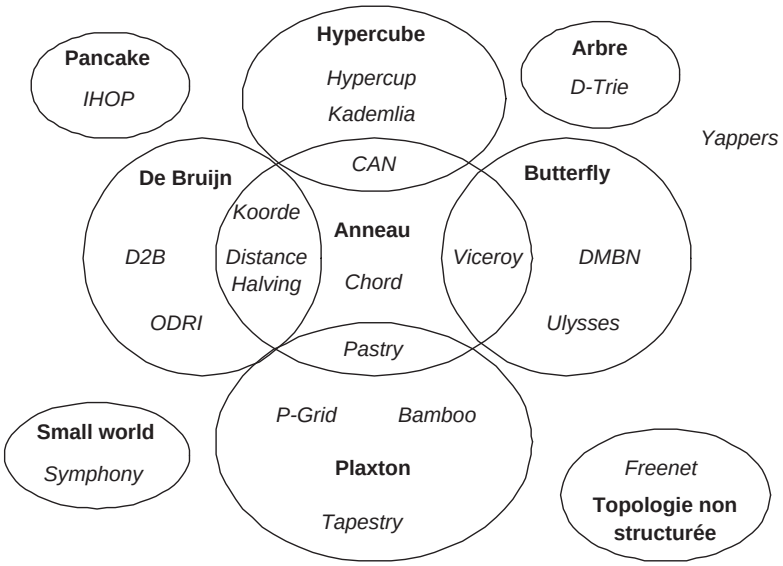


FIG. 3.2 – Taxonomie topologique des différentes DHTs. Les familles sont inscrites en caractères gras et les propositions en italique.

3.3 Propositions à topologie annulaire

Organiser les pairs selon un anneau est envisagé dans plusieurs travaux. Tout d'abord dans Chord [153] qui propose d'organiser les pairs selon un anneau simple. Ensuite, dans Viceroy [103] qui est une version améliorée de Chord, reposant les réseaux Butterfly. Enfin, dans Kademlia [108] qui repose sur la fonction *ou exclusif*. On peut noter que de nombreuses DHTs utilisent de manière plus ou moins directe l'organisation des pairs selon un anneau. C'est par exemple le cas de CAN [124] dont la topologie est un hypercube torique, ou de Pastry [137] qui utilise un anneau pour finaliser son routage. Néanmoins, nous avons choisi de ne pas détailler ces infrastructures

[4]Nous laisserons en marge les propositions qui ne présentent qu'un principe, comme par exemple IHOP [123] ou ODRI [102] et celles qui nous semblaient secondaires comme Ulysses [94] ou Kelips [70].

dans cette section mais plutôt dans celle qui correspond au mieux à leur modèle topologique général. Nous présentons ici Chord [153] et Viceroy [103].

3.3.1 Chord

Chord [153] est un protocole de recherche P2P pour les applications Internet : pour une clé donnée, Chord y associe un pair. Chord est déployé dans plusieurs applications : tout d'abord, dans CFS[5] [30] qui est un système de fichiers distribué à l'échelle de l'Internet, ensuite dans ConChord [6] qui utilise CFS afin de fournir une infrastructure distribuée pour la délivrance de certificats de sécurité SDSI[6] et enfin dans DDNS[7] [29] qui propose une implantation P2P du DNS[8].

Le protocole Chord

Chord repose sur une topologie en anneau ; un pair Chord a la connaissance de son prédécesseur et de son suivant. Une fonction de hachage régulière génère un identifiant pour chaque pair à partir de son adresse IP. Ensuite, chaque pair est placé dans l'anneau de manière à ordonner les identifiants par ordre croissant. Ainsi, chaque pair d'identifiant n est responsable de l'intervalle de clés $]precedent(n), n]$.

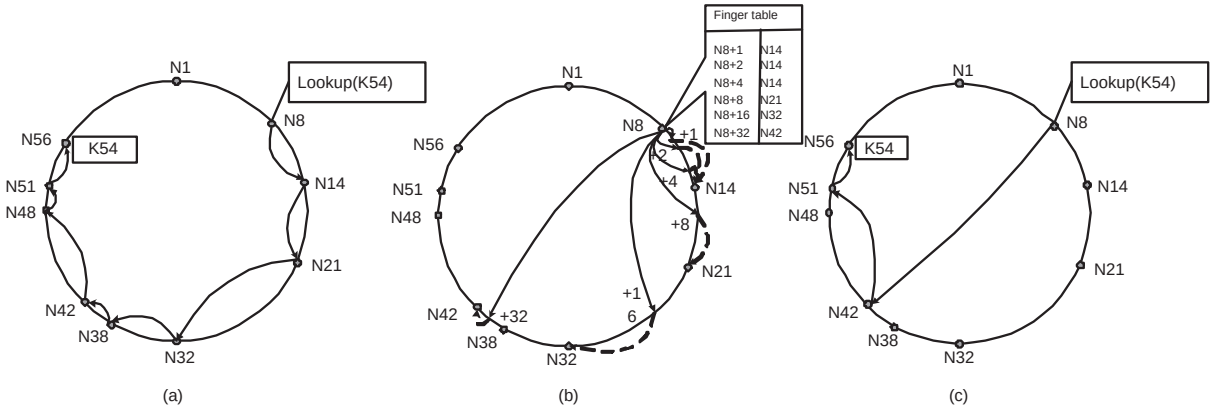


FIG. 3.3 – *Extrait de [153].* (a) Acheminement d'une requête par parcours de l'anneau. (b) Définition des *fingers* d'un pair. (c) Acheminement d'une requête en utilisant les *fingers*.

Pour un pair donné, la simple connaissance de son précédent et de son suivant n'est pas suffisante pour garantir une bonne performance de l'anneau, notamment en termes de nombre de sauts par requête. La figure 3.3.a illustre ce type de problème avec un anneau contenant 10 pairs avec une plage d'adressage comprise dans l'intervalle $[0, 64[$. On peut constater que le routage d'un requête d'un pair de clé N_8 à destination de la clé K_{54} necessitera 8 sauts. Afin de pallier ce problème, pour un espace de clés compris dans l'intervalle $[0, 2^m[$, chaque pair n se voit doté d'une entrée vers les pairs, appelés *fingers*, de clé $suivant(n + 2^{i-1})$ avec $1 \leq i \leq m$. Ainsi, le nombre maximum de pairs parcourus pour acheminer une requête s'exprime en termes de $O(\log(N))$. La figure 3.3.b présente la table de routage du pair N_8 muni de *fingers* et la figure

[5]Collaborative File System
[6]Simple Distributed Security Infrastructure
[7]Distributed Domain Name System
[8]Domain Name Service

3.3.c montre que la même requête de clé K_{54} est cette fois acheminée en 3 sauts. La table de routage d'un pair Chord compte ainsi $O(\log(N))$ entrées.

Insertion, départ et maintenance

L'insertion d'un nouveau nœud dans une communauté se fait par la simple recherche de son suivant dans l'anneau. C'est ensuite le processus de maintenance qui, exécuté régulièrement, va prendre en charge le maintien de la cohérence des tables de routage des pairs concernés par l'arrivée du nouvel élément. En effet, à intervalles réguliers chaque nœud vérifie la validité des informations détenues dans sa table de routage à savoir : son suivant et les *fingers*.

La vérification de son suivant s'effectue en lui demandant quel est son précédent. Si la réponse est soi-même, le suivant est correct. Si le résultat est l'identifiant d'un nœud situé entre soi et son suivant, il devient alors le nouveau suivant, et est notifié que le nœud courant est le nouveau précédent.

La vérification des *fingers* est similaire à la construction de la table et consiste à rechercher pour chaque entrée le nœud correspondant d'identifiant $suivant(n + 2^{i-1})$, avec $1 \leq i \leq m$. La table est mise à jour si jamais un pair différent des pairs actuels est trouvé. Par le biais de ces vérifications régulières, Chord garantit le maintien de la cohérence de l'anneau.

Dans le cas d'un départ de nœud, c'est le processus de maintenance qui permet la mise à jour des informations maintenues par les pairs. En outre, afin de limiter les échecs de requêtes survenant dans l'intervalle de temps situé entre la disparition d'un nœud et l'exécution de ce processus, chaque nœud maintient une liste de suivants, qui peuvent être utilisés alternativement au suivant défaillant.

3.3.2 Viceroy

Viceroy [103] est une proposition de DHT inspirée de Chord mais qui y ajoute de nombreuses améliorations. La principale consiste à construire une topologie à multi-anneaux où chaque pair présente une connectivité constante. En effet, si dans Chord, la connectivité des pairs s'exprime en $\log(N)$, dans Viceroy, elle est constante et égale à 7. Par ce biais, les coûts d'insertion d'un pair dans la communauté, et de mise à jour des informations maintenues sont maîtrisés. En outre, les performances sont similaires à celles proposées dans d'autres DHTs, avec un nombre moyen de sauts qui s'exprime en $O(\log(N))$ pour le routage des messages.

Le protocole Viceroy

Chaque pair Viceroy est muni d'un identifiant aléatoire appartenant à l'intervalle $[0, 1[$ et d'un niveau l tel que l est déterminé aléatoirement dans l'intervalle $[1, \log(N)[$. La figure 3.4 illustre un exemple de topologie Viceroy contenant une trentaine de pairs répartis sur 4 niveaux qui forment quatre anneaux distincts.

Chaque pair d'identifiant id et de niveau l maintient des connexions vers 7 autres pairs. Tout d'abord, vers les pairs précédents et suivants sur l'anneau de même niveau. Ensuite, vers les pairs précédents et suivants sur l'anneau de référence, à savoir celui de niveau 1. Enfin, les trois dernières connexions permettent des passages entre les niveaux et consistent en deux liens vers le niveau inférieur qui référencent les pairs d'identifiant $suivant_{l+1}(id)$ et $suivant_{l+1}(id + \frac{1}{2^l})$ et un lien vers le niveau supérieur vers un pair d'identifiant $suivant_{l-1}(id)$. Sur la figure 3.4, on a représenté les liens maintenus par les pairs d'identifiant $0,14$ et $0,35$. Par souci de clarté, les liens vers le pair de niveau inférieur ne sont pas indiqués. On constate que les liens entre niveaux,

appelés liens Butterfly permettent de parcourir de grandes distances dans la communauté. C'est par exemple le cas du pair $O, 14$ qui référence un pair d'identifiant proche de $0, 75$.

Etant donné cette topologie, le routage d'un message s'effectue en trois étapes. La première consiste à remonter les niveaux jusqu'à atteindre un nœud de niveau 1. Ensuite, la seconde effectue la redescente des niveaux en utilisant les liens Butterfly de courte ou longue distance. Enfin, lorsqu'aucun pair de niveau inférieur ne permet de se rapprocher plus de la cible, les liens intra-anneau sont utilisés jusqu'à l'arrivée à destination. De cette manière, avec une très forte probabilité, les messages sont routés en $O(\log(N))$ sauts.

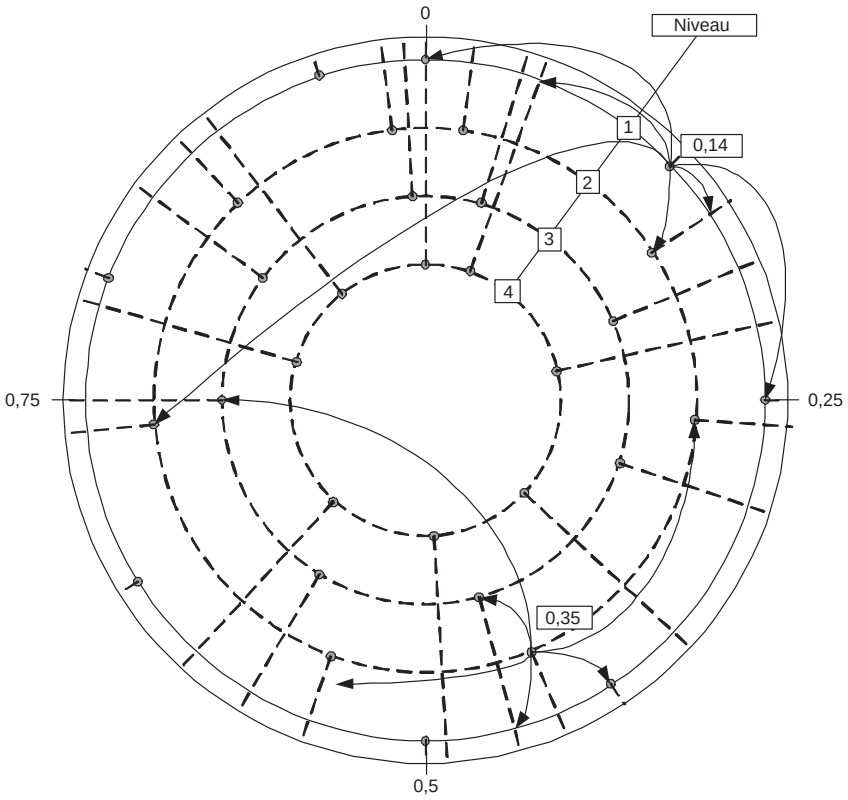


FIG. 3.4 – Exemple de topologie Viceroy

3.4 Propositions fondées sur l'algorithme de Plaxton

Une seconde famille de DHTs repose sur l'algorithme de Plaxton [122]. Parmi les DHTs de cette famille, on retient trois suivantes. Pastry [137] est une DHT sur laquelle nous avons travaillé. Elle est utilisée dans PAST, une application de stockage de fichiers. Tapestry [170, 132] est une autre proposition qui est déployée dans OceanStore [93], une application de stockage de fichier à grande échelle et dans Bayeux [172], une infrastructure de routage multicast applicatif. Enfin, P-Grid [1] est une proposition préliminaire à celles des DHTs actuelle qui n'effectue pas d'association directe entre l'identifiant des nœuds et celui des ressources. Dans cette section, nous introduisons le principe général de l'algorithme de Plaxton et nous présentons Pastry [137].

3.4.1 L'algorithme de Plaxton

Plaxton [122] propose un algorithme de localisation d'objets et de routage pour les environnements de type *overlay* complètement distribués. Une de ses particularités repose sur le fait que la localisation et le routage sont effectués en même temps, réduisant ainsi la charge de ce processus. Dans sa description, Plaxton distingue plusieurs entités qui peuvent être des serveurs, hébergeant des objets, des clients, effectuant des requêtes sur ceux-ci, et des routeurs qui relaient les requêtes. Dans le cadre des réseaux P2P, un pair peut prendre simultanément ces trois rôles de manière indifférente. Ensuite, le nommage des objets et pairs est construit de manière aléatoire ou pseudo-aléatoire, selon une fonction de hachage unique et connue [13]. Chaque identifiant d'objet et de pair est écrit dans une base connue déterminée au préalable et possède une longueur fixe.

Routage

Chaque pair P maintient une table de routage scindée en N niveaux, où N est le nombre de digits des identifiants. Chaque niveau n de la table contient une liste de voisins de P dont les $n-1$ derniers chiffres sont communs à ceux de P. La table de routage possède B colonnes et chaque colonne c représente le digit de rang n. La figure 3.5.b présente un exemple de patron de table de routage du pair 0325 dans le cas où $B = 16$ et $N = 4$. On y voit par exemple que le pair situé dans la deuxième colonne du niveau 3 doit posséder un suffixe commun qui est 25, son deuxième digit doit être égal à 1 et son premier digit, représenté par un x peut être quelconque.

Ensuite, un exemple de routage est présenté sur la figure 3.5.a. Lorsque le nœud d'identifiant 0325 fait parvenir un message à un nœud d'identifiant 4598, il va consulter le premier niveau de sa table de routage, y chercher un voisin dont le dernier chiffre est 8 (ici le nœud $B4F8$) puis lui faire suivre ce message. A son tour, ce nœud va consulter le deuxnième niveau de sa table de routage et chercher un voisin dont l'identifiant se termine par 98 (ici 9098) et lui faire suivre le message. Ce processus se répète ensuite jusqu'à l'atteinte du nœud considéré. Cette méthode de routage garantit qu'un nœud présent dans un système de N pairs peut être joint en $\log_B(N)$ itérations.

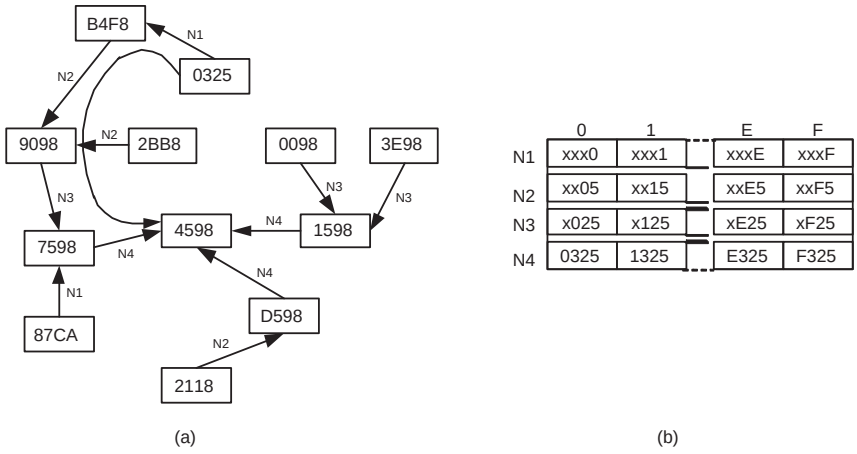


FIG. 3.5 – *Extrait de [93].* (a) Exemple de routage selon l'algorithme de Plaxton. (b) Patrons de la table de routage du nœud d'identifiant 0325.

Localisation

Le mécanisme de localisation de Plaxton permet à un pair de localiser et d'envoyer des messages à un objet particulier situé sur un pair distant. Pour cela, un serveur S publie le fait qu'il possède un objet O au nœud racine de O. Tout au long du chemin emprunté pour acheminer ce message, chaque pair conserve un pointeur vers le serveur de l'objet sous la forme d'un couple `(ObjectId, ServerId)`. Le nœud racine d'un objet est un pair dont l'identifiant est le plus proche numériquement de celui de l'objet et se comporte ainsi comme un simple pointeur vers le serveur de l'objet.

En considérant ce mécanisme de publication, le traitement d'une requête de localisation est effectué de la manière suivante : lorsqu'un nœud désire localiser un objet O, il émet une requête qui est propagée vers le nœud racine. Si un nœud intermédiaire possède un pointeur vers le serveur de l'objet, la requête est redirigée vers celui-ci, sinon, c'est le nœud racine qui se charge de cette opération.

3.4.2 Pastry

Pastry [137] est un protocole de routage et de localisation P2P : pour une clé donnée, Pastry associe un pair dont l'identifiant est le plus proche numériquement de la clé. Son architecture repose sur un modèle P2P décentralisé et utilise une approche hybride pour le routage : un routage grossier est effectué grâce à l'algorithme de Plaxon [122], mais les dernières étapes utilisent la notion de voisins virtuels organisés selon une topologie annulaire [69] similaire à Chord [153]. Pastry est actuellement utilisé dans deux applications P2P qui sont : PAST [50], une application distribuée de stockage de fichiers et Scribe [138] une application P2P de diffusion d'évènements qui peuvent être reçus en souscrivant à un thème particulier.

Le protocole Pastry

Chaque pair Pastry est muni d'un identifiant de 128 bits qui est le résultat d'une fonction de hachage appliquée à l'adresse IP ou la clé publique du pair. Dans un réseau contenant N pairs Pastry, pour une clé donnée, le protocole est capable d'associer un pair dont l'identifiant est le plus proche numériquement en $\log_{2^b} N$ sauts[9], en moyenne.

Une table de routage Pastry contient trois catégories d'entrées. La première contient l'ensemble des voisins virtuels du pair considéré en termes de distance numérique de l'identifiant de pair. La seconde catégorie est la table de routage à proprement parler du pair. Chaque ligne l contient $2^b - 1$ entrées dont les l premiers chiffres sont communs à ceux du pair considéré. On peut noter que chaque entrée de la table est celle dont la distance réelle est la plus petite entre le pair et l'entrée considérée. Enfin, la dernière catégorie de la table contient l'ensemble des voisins réels du nœud. La métrique choisie pour évaluer la distance réelle entre pairs est le nombre de sauts IP. Ainsi, une table de routage Pastry contient $(2^b - 1) * (\log_{2^b} N) + 2l$ entrées. Le tableau 3.1 présente un exemple simplifié de table de routage Pastry. Dans cet exemple, b est fixé à 2 et la longueur des identifiants est de 8 digits. On y voit par exemple que la deuxième colonne de la cinquième ligne (soit $l = 4$) possède une entrée pour un pair possédant un préfixe commun de 4 digits, et un cinquième digit égal à 1.

Le principe de routage de Pastry est le suivant : lorsqu'un message muni d'une clé K arrive dans un pair n, celui-ci vérifie tout d'abord si la clé correspond à un pair d'identifiant voisin virtuellement. Dans ce cas, le message est routé directement vers le pair correspondant. Sinon, le

[9] b est un paramètre de configuration typiquement égal à 4

Pair **10233102**			
Voisins virtuels			
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232
Table de routage			
-0-2212102	**1**	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	**2**	10-3-23302
102-0-0230	102-1-1302	102-2-2302	**3**
1023-0-322	1023-1-000	1023-2-121	**3**
10233-0-01	**1**	10233-2-32	
0		102331-2-0	
		2	
Voisins réels			
13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

TAB. 3.1 – *Extrait de [137].* Table de routage d'un pair d'identifiant 10233102 avec $b = 2$

pair n détermine m, le nombre de chiffres communs à son identifiant et à K, consulte sa table de routage pour trouver un pair qui présente $m+1$ chiffres en commun et lui fait suivre le message. Si aucun pair ne correspond, le pair n cherche un autre pair ayant m chiffres en commun et dont l'identifiant est le plus proche de K. Le message est arrivé à destination lorsqu'aucun voisin virtuel ni entrée de la table de routage ne permet de se rapprocher plus de la clé K.

3.5 Autres propositions

Dans les sections 3.3 et 3.4, nous avons étudié deux familles de DHTs et particulièrement détaillé deux propositions sur lesquelles nous avons travaillé qui sont Chord [153] et Pastry [137]. Dans cette section, nous présentons plus brièvement d'autres familles de DHTs : tout d'abord celle des DHTs construites sur un hypercube, celle des graphes de De Bruijn, puis les améliorations faites sur la famille des graphes Butterfly, ensuite la famille des DHT de type *Small world* et enfin la famille des DHTs construites sur une topologie non structurée.

3.5.1 Propositions fondées sur un hypercube

La topologie de Chord repose sur un espace cartésien circulaire à une seule dimension. CAN[10] [124] étend cette topologie à un espace multidimensionnel et repose sur une topologie torique de dimension d. On parle aussi de topologie en hypercube, car CAN peut être vu comme un cube de dimension d, dont les extrémités de chaque dimension se rejoignent.

Le protocole CAN

Dans une communauté CAN, l'ensemble de l'espace est partitionné parmi l'ensemble des pairs inscrits de sorte que chaque pair soit responsable d'un sous-ensemble unique. La figure 3.6 présente un exemple de partitionnement dans un espace à deux dimensions, mis à plat, munis de coordonnées comprises entre 0 et 1 sur chaque axe. Elle permet notamment de voir que l'ensemble de l'espace est constamment alloué.

[10] Content Addressable Network

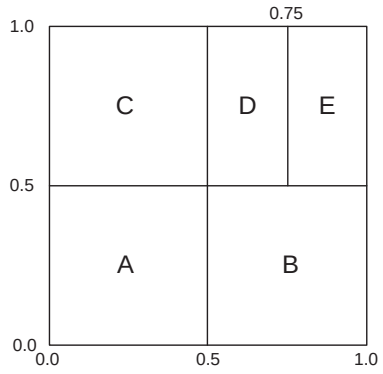


FIG. 3.6 – *Extrait de [124].* Exemple d'espace à deux dimensions avec 5 pairs

Le principe de stockage d'un couple *(clé, valeur)* est le suivant : une fonction de hachage unique, appliquée à la clé, permet de déterminer un point de coordonnées P appartenant à l'espace. Le pair responsable du sous-ensemble dans lequel P est inscrit est alors responsable du stockage du couple *(clé, valeur)*. Ensuite, pour récupérer une valeur stockée, un pair applique à une clé la fonction de hachage, en récupère une coordonnée de point P' et utilise alors le protocole de routage de CAN pour contacter le pair responsable du point P' considéré.

Concernant le routage, un pair CAN possède une table de routage contenant des informations[11] sur chacun de ses voisins. Par exemple, dans le cas de la figure 3.6, le pair responsable du sous-ensemble D a pour voisins C, B et E. Dans le cas d'un espace de dimension d, chaque pair inscrit ainsi dans sa table de routage la liste de ses $2d$ voisins. L'acheminement d'un message vers sa destination s'effectue en faisant suivre le message à des pairs qui présentent des coordonnées qui sont de plus en plus proches de celles du pair destination. Ainsi, on montre que la longueur moyenne du chemin emprunté par un message est de $\frac{d}{4}n^{\frac{1}{d}}$. De plus, augmenter le nombre de pairs CAN n'influe pas sur le nombre d'informations qu'un pair détient à propos de ses voisins, par contre, la longueur moyenne d'un chemin évolue en termes de $O(n^{\frac{1}{d}})$.

3.5.2 Propositions fondées sur les graphes de De Bruijn

Les graphes de De Bruijn ont fait l'objet de nombreuses propositions d'application au routage dans les réseaux [54, 150]. Aujourd'hui, la communauté P2P s'intéresse à ces graphes dans le cadre des DHTs et quatre infrastructures ont vu le jour : D2B [63], Koorde [82], *Distance Halving* [117] et ODRI [102]. Dans cette section, nous introduisons le principe des graphes de De Bruijn et présentons D2B [63].

Un graphe de De Bruijn [35] $B(d, k)$ est un graphe direct dont les nœuds sont des chaînes de longueur k utilisant un alphabet $\{0, \ldots, d-1\}$. Il existe un arc partant d'un nœud $x_1x_2 \ldots x_k$ vers d nœuds $x_2 \ldots x_k\alpha$ avec $\alpha \in \{0, \ldots, d-1\}$. La figure 3.7 illustre un exemple de graphe $B(2, 3)$. On y voit par exemple que le nœud 001 possède un arc vers le nœud 010, car ils possèdent respectivement le suffixe et préfixe commun 01. Ensuite, Le routage d'un message depuis un nœud $x_1 \ldots x_k$ vers le nœud $y_1 \ldots y_k$ s'effectue, sous sa forme la plus simple, en traversant les nœuds d'identifiant $x_2 \ldots x_k y_1$, puis $x_3 \ldots x_k y_1 y_2$, ..., et enfin $x_k y_1 \ldots y_{k-1}$.

Un algorithme de routage optimisé est proposé dans [96]. Celui-ci, appelé *LRL Path*[12] consi-

[11] c'est à dire une association (Adresse IP, coordonnées de l'espace virtuel)

[12] ou symétriquement *RLR Path* selon le sens du routage

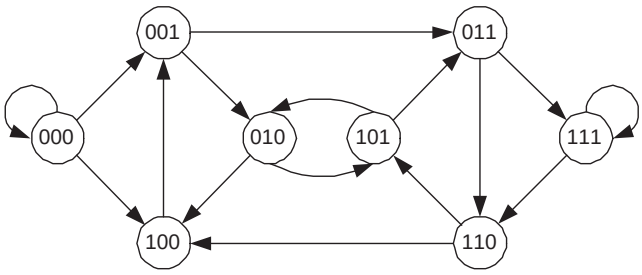


FIG. 3.7 – *Extrait de [62].* Exemple de graphe de De Bruijn $B(2,3)$

dère un graphe de De Bruijn où les arcs sont bidirectionnels, et permet ainsi de réduire le nombre de sauts en recherchant une partie commune entre un identifiant de nœud et de clé. Ce principe de routage est illustré dans [33]. On considère X, un pair source d'identifiant $X_L Z X_R$ et Y, un pair cible d'identifiant $Y_L Z Y_R$, où X_L, X_R, X_L, X_R, et $Z \in \{0,1\}^*$. Le routage de X vers Y s'effectue en faisant $\mid X_L \mid$ sauts à gauche, $\mid Y_L \mid$ sauts à droite, $\mid X_R \mid$ sauts à droite et enfin $\mid Y_R \mid$ sauts à gauche. Le plus court chemin pour acheminer une requête revient ainsi à déterminer la partie commune entre X et Y qui minimise le nombre de sauts.

D2B

D2B [63, 62] repose sur une topologie construite selon un graphe de De Bruijn $B(2,k)$. L'ensemble de clés attribuables appartient à $\{0,\ldots,2^m-1\}$ et chaque clé peut être représentée sous forme d'une chaîne binaire de longueur m. Les nœuds D2B, quant à eux, possèdent aussi un identifiant représentable sous forme de chaîne binaire, mais de longueur inférieure ou égale à m. La valeur d'un nœud u d'identifiant $x_1 \ldots x_k$ est définie par la fonction $val(u) = 2^{m-k}.\sum_{i=1}^{k} x_i.2^{k-i}$ et chaque nœud est responsable des clés comprises dans l'intervalle $[val(u), 2^{m-k}(\sum_{i=1}^{k} x_i.2^{k-i}+1)-1[$. En d'autres mots, une clé de représentation binaire $k_1 \ldots k_m$ est référencée par le nœud d'identifiant $x_1 \ldots x_k$ si et seulement si $x_1 \ldots x_k$ est un préfixe de $k_1 \ldots k_m$. D'un point de vue de la table de routage, chaque pair D2B maintient des entrées vers un ou des fils, un ou des parents et des frères. Le tableau 3.2 présente la description formelle des identifiants pour ces trois types d'entrées. Le principe de routage d'un message est identique à celui décrit dans la section 3.5.2 auquel on a ajouté une optimisation qui prend en compte le suffixe commun de l'identifiant d'un pair avec le préfixe de la clé à atteindre. De plus, on peut noter que les liens fraternels ne sont pas utilisés dans le cadre du routage, mais dans le cadre de la maintenance, pour la redistribution des clés.

Lien	**Identifiant**
Fils	$x_2 \ldots x_j$, avec $j \leq k$ ou
	$x_2 \ldots x_k y_1 \ldots x_l$, avec $1 \leq l \leq m-k+1$
Parents	$\alpha x_1 \ldots x_j$, avec $\alpha \in \{0,1\}$ et $j \leq k$ ou
	$\beta x_1 \ldots x_k y_1 \ldots y_l$, avec $\beta \in \{0,1\}$ et $1 \leq l \leq m-k-1$
Frères v et w,	w grand frère de v s'il a la plus petite valeur $val(w)$ supérieure à v
fils de u	w petit frère de v s'il a la plus grande valeur $val(w)$ inférieure à v

TAB. 3.2 – Les entrées de la table de routage d'un pair D2B u d'identifiant $x_1 \ldots x_k$

3.5.3 Améliorations des graphes Butterfly

Dans la section 3.3.2, nous avons étudié Viceroy [103], une proposition qui repose sur les graphes Butterfly. Depuis, des améliorations ont été apportées à ce travail initial. DMBN[13], une version améliorée reposant sur un graphe de type multi-Butterfly, a été proposée dans [32]. Son principe est équivalent à Chord ou Viceroy, mais cette proposition accroît la tolérance aux fautes face aux attaques qui consisteraient à supprimer une partie des pairs. Ce travail s'inscrit dans la suite de deux travaux [59, 139] qui visent à concevoir des DHTs tolérantes aux attaques ciblées. En effet, la plupart des propositions de DHTs présentent une bonne tolérance aux fautes face à des attaques aléatoires mais ne considèrent pas le cas d'attaques ciblées qui tenteraient de supprimer une fraction de nœuds choisis. Ces dernières propositions permettent donc de construire des DHTs fondées sur des réseaux Butterfly enrichis qui permettent de garantir que : même si la moitié des nœuds disparaît, la majorité des ressources sera toujours accessible à la majorité des nœuds [59]. Le coût de cette tolérance aux fautes se répercute sur le nombre de messages à transmettre qui s'exprime en $O(\log^2(N))$, voire en $O(\log^3(N))$.

3.5.4 Proposition fondée sur le modèle *Small World*

Le modèle *Small world* a été démocratisé par Malgram en 1967 [110] lorsqu'il mena l'expérience suivante aux Etats-Unis : on donne à une personne une lettre à destination d'un tiers qu'elle ne connaît pas. On lui propose de faire passer la lettre à une de ses connaissances qui la fera, à son tour, passer à une de ses connaissances, ... jusqu'à ce que la lettre atteigne sa destination. On a montré alors que la lettre emprunte un chemin constitué de sauts de grande distance qui se rétrécissent au fur et à mesure que l'on se rapproche de la destination. De plus, en moyenne, seul un petit nombre d'intermédiaires est nécessaire pour acheminer la lettre, à savoir 6 à 7 pour l'expérience menée aux Etats-Unis. Le réseau de connaissances mis en œuvre dans cette expérience repose sur le modèle *Small World* [89] et est utilisé dans le cadre des systèmes distribués et particulièrement des DHTs.

Symphony

Symphony [105] est une DHT qui repose sur le modèle *Small World*. La topologie utilisée est un anneau, similaire à Chord, où les identifiants sont rangés par ordre croissant. Ensuite, chaque pair maintient deux types de contacts : des contacts de courte distance avec deux précédents et deux suivants dans l'anneau, et k contacts de longue distance. k est un paramètre fixé, typiquement égal à 4. Les k contacts de longue distance sont choisis de manière aléatoire et sont distincts. De même, un pair compte le nombre de fois où il est référencé par d'autres nœuds dans la limite de $2k$. Ceci permet de référencer les pairs de manière équivalente et d'avoir une topologie équilibrée. Ensuite, le routage s'effectue récursivement en essayant de minimiser la distance entre un pair et la clé requise. La performance induite par ce modèle est similaire aux autres propositions avec une longueur moyenne des requêtes qui s'exprime en $O(\frac{1}{k}\log^2(N))$ et un degré de connectivité constant qui s'exprime en $O(1)$.

3.5.5 DHTs à topologie non structurée

Les DHTs présentées dans les sections précédentes présentent toutes une topologie structurée comme un hypercube, un graphe Butterfly, ou un graphe de De Bruijn. Néanmoins, d'autres

[13] Dynamic Multi-Butterfly Network

propositions reposent sur des topologies non structurées. Nous présentons ici Freenet [26] qui fut la première proposition de réseau à routage par contenu.

Freenet

Freenet [26] n'est pas à proprement parler une infrastructure de routage et de localisation générique pour le P2P, mais plutôt une application de stockage de fichiers qui inclut un mécanisme de routage par contenu. Son objectif est de fournir une infrastructure de stockage de fichier distribuée qui soit anonyme et résistante à la censure et aux attaques qui viseraient à rendre une ressource inaccessible. Son principe est différent de celui des DHTs notamment sur un point majeur : les pairs ne sont pas identifiés par le hash d'un attribut particulier comme l'adresse IP ou le nom de la machine. L'utilisation de l'adresse de niveau transport est suffisante. Par contre, les identifiants de fichiers sont le résultat d'une fonction de hachage appliquée à leur description. Dans ce contexte, il n'est fait aucun rapprochement entre les clés des ressources et les pairs.

La découverte d'une ressource est effectuée comme suit : chaque pair possède des voisins auxquels il publie la liste des identifiants de fichiers qu'il héberge. Ensuite, lorsqu'un nœud désire accéder à un fichier, il regarde tout d'abord dans son cache local pour vérifier qu'il ne possède pas le fichier. Sinon, il envoie la requête à un de ses voisins qui possède un fichier dont l'identifiant se rapproche de celui du fichier requis. Ce routage s'effectue de part en part jusqu'à l'atteinte de la ressource requise. Si jamais la ressource n'est pas disponible, la requête sera stoppée du fait de l'expiration d'un TTL[14] ou par l'absence de nouveaux pairs à contacter. L'identification des messages permet en effet d'éviter les boucles.

L'accès à un fichier se fait en parcourant en sens inverse le chemin emprunté par la requête de découverte. Au sein de chaque pair parcouru, le fichier est copié. Par ce biais, Freenet rend très difficile la suppression d'un fichier stocké et permet d'améliorer la qualité du routage car, si une requête pour le même fichier se reproduit, les nœuds intermédiaires peuvent y répondre sans avoir à router la requête jusqu'au pair qui stocke initialement le fichier requis. De plus, la topologie du réseau tend à s'organiser car, au fil des requêtes, les pairs se spécialisent sur des fichiers de clés proches.

L'anonymat est garanti par deux principes fondamentaux : dans le processus de routage, chaque pair ne connaît ni la source ni la destination d'une requête, et chaque fichier est crypté à l'aide d'une clé publique qui est la description non hachée du fichier. Ainsi, un pair ne peut pas savoir quel fichier il héberge.

3.6 Analyse et améliorations

Face à la multitude de propositions de DHTs, deux questions se posent. *Comment se différencient les DHTs ?* Et, *Quelles sont les limites de ces infrastructures ?* Dans cette section, nous nous attachons tout d'abord à comparer les différentes infrastructures de DHTs et à en souligner les aspects communs aussi bien que les différences. Dans un second temps, nous présentons quelques travaux qui proposent des améliorations pour les infrastructures existantes.

3.6.1 Comparaison des caractéristiques théoriques

Des comparaisons des performances théoriques des différentes DHTs ont été effectuées dans de nombreux articles dont [113, 32, 117, 123, 69]. De même, l'étude du comportement des DHTs

[14] Time To Live

face à la dynamicité est largement étudiée [98, 131, 100, 23]. Le cas des graphes de De Bruijn, fortement controversé, est détaillé dans plusieurs travaux qui, soit justifient son utilisation [102], soit dénotent son inadaptibilité à être utilisé dans le cadre des DHTs [33, 167].

Le tableau 3.3 recense les principales propositions de DHTs ainsi que leurs propriétés théoriques. On y a inscrit le modèle topologique, le degré de connectivité, le nombre de sauts moyen pour acheminer une requête et le taux de congestion des pairs. Cette dernière caractéristique représente le nombre de clés dont un pair est responsable. La première remarque concernant cette vue synthétique des caractéristiques théoriques des DHTs réside dans la similitude des propriétés. En particulier, le taux de congestion des pairs qui, à part pour CAN, est le même pour les différentes approches. De même, le nombre de sauts moyen pour router une requête s'exprime toujours en $O(\log N)$. D'ailleurs, le tableau 3.4 confirme cette homogénéité. C'est un extrait de [69] où, dans une communauté de 65536 pairs statiques, la longueur des chemins empruntés a été mesurée et est quasiment constante pour toutes les approches, sauf pour l'approche Butterfly. En fait, la différence majeure entre les différentes propositions de DHTs porte sur le degré de connectivité. Sur ce critère, on distingue deux types de stratégies : les DHTs qui présentent un degré de connectivité logarithmique, et celles qui présentent un degré de connectivité constant. Pour la première catégorie, l'aspect logarithmique du degré de connectivité assure le passage à l'échelle de ces infrastructures et est donc une propriété satisfaisante pour l'usage dans un environnement P2P à grande échelle. Pour la seconde catégorie, l'aspect constant du degré de connectivité semble très avantageux pour deux raisons [103, 63] : tout d'abord, comme pour le cas de la connectivité logarithmique, il permet d'assurer le passage à l'échelle en assurant que la taille de la table de routage des pairs sera constante quel que soit le nombre de pairs participant. Ensuite, il permet de maîtriser les coûts d'insertion d'un nœud et de maintenance de l'infrastructure, car la mise à jour des entrées de la table de routage n'implique alors qu'un nombre de pairs connus, et cette caractéristique de performances est importante dans le cas de communautés très volumineuses.

DHT	**Topologie**	**Degré de connectivité**	**Nombre de sauts**	**Congestion**
CAN	Hypercube	$O(d)$	$O(n^{\frac{1}{d}})$	$O(d.n^{\frac{1}{d-1}})$
Chord	Anneau	$O(\log_2(N))$	$O(\log N)$	$O((\log N)/N)$
Pastry	Plaxton	$O((b-1)*\log_b N)$	$O(\log_{2^b} N)$	$O((\log N)/N)$
Viceroy	Butterfly	$O(1)$	$O(\log N)$	$O((\log N)/N)$
D2B	De Bruijn	$O(1)$	$O(\log N)$	$O((\log N)/N)$

TAB. 3.3 – *Extrait de [63].* Comparaison des caractéristiques théoriques de différentes DHTs

Topologie	**Nombre de sauts Moyen**	**Nombre de sauts Médians**	**Nombre de sauts pour 90 Percentiles**
Anneau	7.4	7	10
Plaxton	7.7	8	10
Butterfly	21.4	21	28
Hypercube	7.7	8	10
XOR	7.7	8	10

TAB. 3.4 – *Extrait de [69].* Comparaison des nombres de sauts nécessaires au routage d'une requête pour différentes DHTs composées de 65536 pairs

3.6.2 Extensions

La majeure partie des propositions de DHT ont été effectuées entre 2001 et 2003. A ce jour, le domaine des DHTs mûrit et le travail actuel concerne principalement la comparaison des différentes infrastructures, présentée dans la section 3.6, et la manière dont les DHTs peuvent évoluer pour être effectivement déployées dans des infrastructures P2P. Le premier travail que nous présentons concerne la proposition d'une API standard pour les DHTs. Dans un second temps, nous abordons un nouvel axe de recherche des DHTs : la recherche de clés par critère sémantique.

Proposition d'une API commune

Parmi l'ensemble des propositions de DHTs, une partie non négligeable présente une ou plusieurs implantations. Chaque implantation propose une API[15] particulière, incompatible avec les autres. Un travail de Dabek et al. [31] propose de définir une API commune pour l'ensemble des DHTs. Cette contribution est très intéressante, car elle permet tout d'abord, du point de vue du service déployé, d'abstraire les mécanismes de la DHT et de masquer les spécificités propres à une infrastructure particulière. Ensuite, cette API permet d'envisager l'interopérabilité des services construits sur des réseaux P2P structurés, ces derniers pouvant être déployés indifféremment sur CAN, Pastry ou Chord, par exemple.

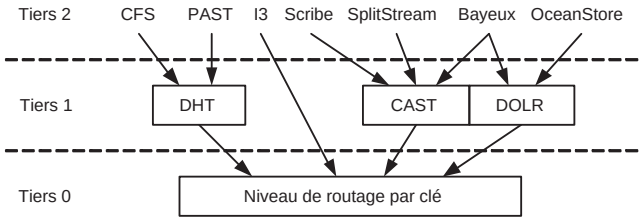


FIG. 3.8 – *Extrait de [31].* APIs et abstractions pour les réseaux P2P structurés

En outre, d'autres infrastructures basées sur des réseaux structurés sont prises en compte dans cette API. Parmi celles-ci, on compte celles de localisation et de routage d'objets (DOLR[16]) et celles de gestion de groupes *anycast* et *multicast* (CAST). La figure 3.8 illustre l'organisation fonctionnelle de cette proposition. Un premier niveau, appelé *tiers O*, propose une API de bas niveau relative aux mécanismes découverte et routage fondés sur des clés. On y trouve des méthodes de routage et d'accès à la table de routage locale. Le niveau supérieur, définit la manière dont les différents systèmes DHTs, DOLR et CAST vont pouvoir utiliser cette API dans un contexte qui leur est propre. Enfin le niveau *tiers 2* se compose des différents services qui peuvent utiliser les méthodes définies aux niveau 1, ou directement au niveau 0, comme c'est le cas pour I3 [152].

Recherche et organisation sémantique

La principale limite des DHTs réside dans l'impossibilité de faire des requêtes complexes. Les deux seules opérations qu'elles autorisent sont `put(key, data)` et `get(key)`. Pour retrouver

[15] Application Programming Interface
[16] Decentralized location and routing

une donnée référencée, il faut donc connaitre sa clé exacte car les requêtes du type `*.mp3` sont impossibles.

Pour améliorer ce fonctionnement, plusieurs travaux [113, 145, 142, 171] proposent des mécanismes d'organisation ou de recherche sémantique. Pour ce faire, deux méthodes ont été envisagées. La première repose sur l'utilisation d'une fonction de hachage sémantique. Celle-ci utilise des mots clés pour générer la clé d'une donnée et elle garantit que, pour deux ensembles de mots clés proches sémantiquement, les clés résultantes le seront aussi. De cette manière, les données stockées par les pairs sont groupées par proximité sémantique. L'inconvénient de cette méthode réside dans le déséquilibre qui peut apparaître au niveau de la charge des pairs. Les pairs responsables de clés qui correspondent à des fichiers populaires seront plus fortement chargés et sollicités. La seconde méthode consiste à utiliser un arbre de recherche construit au-dessus de la DHT. Celui-ci permet de rechercher des données par mot clés en fournissant un niveau d'indirection supplémentaire. Plusieurs organisations d'arbre sont possibles et on retient en particulier celles qui sont fondées sur les ontologies.

3.7 Synthèse

Un des problèmes majeurs posé par la décentralisation et la dynamicité du modèle P2P concerne la manière de découvrir et d'accéder à une ressource. Les modèles à répertoire centralisé et par inondation ont montré leurs limites. Les tables de hachage distribuées sont des infrastructures qui répondent à ce problème en fournissant une infrastructure totalement distribuée qui permet de localiser une ressource dans un environnement P2P. Les DHTs reposent sur une topologie structurée qui est construite selon un modèle particulier comme l'algorithme de Plaxton ou les graphes de De Bruijn. L'intérêt de ces modèles réside dans les bonnes propriétés qu'ilx offrent à la DHT, comme par exemple la maîtrise du nombre de sauts nécessaires au routage d'une requête, ou le degré de connexité du graphe topologique.

Dans ce chapitre, nous avons passé en revue la majorité des propositions de DHT actuelles, en les classant suivant leur modèle topologique. Nous avons particulièrement détaillé Chord et Pastry qui sont deux infrastructures sur lesquelles nous avons travaillé dans le cadre de nos recherches sur la gestion de réseaux et services P2P. Dans un second temps, nous avons montré que la majorité des DHTs présente des performances théoriques assez similaires, et que leurs différences portent surtout sur la taille constante ou logarithmique de leur table de routage. Enfin, nous avons brièvement présenté les nouveaux axes de recherche du domaine des DHTs, qui sont l'interopérabilité, avec la proposition d'une API commune, et la recherche sémantique.

Chapitre 4

Supervision de réseaux et modèle pair à pair

Sommaire

Les notions de supervision de réseaux et de modèles P2P semblent antinomiques. Dans ce chapitre, nous présentons les liens qui existent entre elles. Dans un premier temps, nous présentons le domaine de la gestion des réseaux et services. Ensuite, nous expliquons pourquoi le modèle P2P requiert l'ajout de mécanismes de supervision pour pouvoir délivrer un service de qualité. Dans un troisième temps, nous passons en revue les différentes approches de supervision existantes pour les services P2P. Celles-ci incluent notamment l'utilisation de modèles économiques et l'utilisation de mécanismes de réputation et de confiance. Dans un quatrième temps, nous nous intéressons à l'utilisation du modèle P2P pour la supervision des réseaux et services. Enfin, nous terminons ce chapitre par la proposition d'une classification des applications P2P orientée vers la gestion qui permet de les scinder en différentes classes qui vont influer sur la conception d'une infrastructure de gestion.

4.1 La gestion des réseaux et services

La gestion[1] de réseaux et services est une activité primordiale dans toute infrastructure qui fournit des services au moyen d'un réseau informatique. Elle est la composante qui va permettre

[1]Dans ce manuscrit, on emploie indifféremment les termes de gestion, supervision ou administration

d'assurer les usagers de la garantie d'une qualité de service. Son rôle principal, est de veiller à ce que les objectifs fixés dans le cadre d'une politique ou d'un contrat de service puissent être atteints et stabilisés [58]. Plus formellement, nous proposons de caractériser la gestion de réseaux et services à travers la définition qui suit.

Définition 5 *La gestion de réseaux regroupe l'ensemble des activités qui consistent à mettre en œuvre, maintenir et mettre à jour les infrastructures de réseaux ainsi que les services qu'ils offrent à des usagers.*

Pour illustrer l'activité de gestion de réseaux, on utilise souvent un parallèle avec le monde de l'automatique qui propose de voir les réseaux informatiques et leur supervision comme un système asservi. La figure 4.1 illustre ce parallèle en montrant que la supervision consiste à introduire une boucle de régulation autour d'un système qui est initialement libre et dont l'état n'est donc pas contrôlable. L'entrée d'un tel système asservi est une consigne, qui dans le cadre des réseaux, peut être un objectif à atteindre (dans le cas d'une entreprise, une administration ou une université), ou un contrat de service (dans le cas où le service délivré est payant). Le système administré représente aussi bien une infrastructure matérielle, comme des machines, des routeurs ou des serveurs que des services, tels que la messagerie électronique, un VPN[2] ou un pare-feu. La sortie du système se caractérise par son état, qui inclut par exemple sa performance, sa longévité ou son coût.

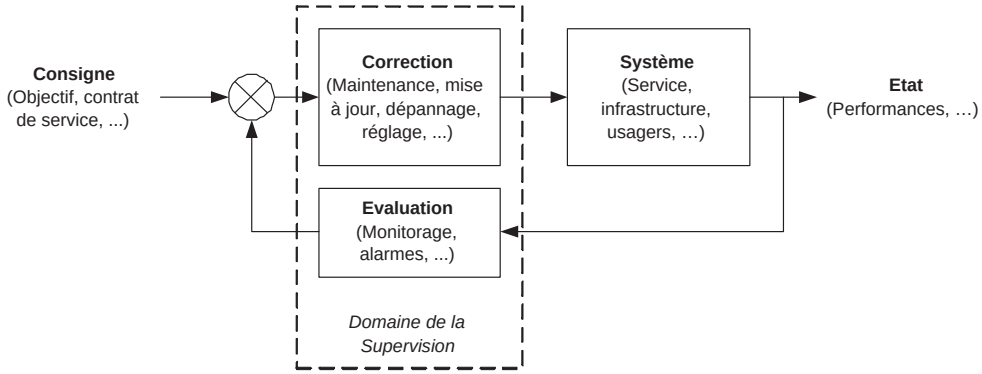


FIG. 4.1 – Analogie entre l'activité de supervision et les systèmes asservis en automatique

L'intégration d'une infrastructure de supervision à un tel système va consister en l'ajout de deux composants. Le premier concerne l'évaluation. Il représente la capacité à observer et obtenir une image abstraite du système supervisé. Différents moyens permettent d'accomplir cette tâche. Les deux principaux sont (1) le monitorage qui consiste à surveiller le système par le biais de rapports réguliers[3] ou de sondage[4], et (2) la mise en place d'alarmes qui informent de l'apparition de problèmes sur le système ou sa dérive hors des limites de fonctionnement fixées. La seconde activité de la gestion concerne le contrôle. Elle représente la capacité à agir sur le système pour le maintenir dans des limites de fonctionnement établies dans les objectifs. De nombreuses actions sont comprises dans le contrôle comme l'installation, la mise en œuvre, la maintenance, la mise à jour ou le réglage[5] du système supervisé.

[2]Virtual Private Network
[3]appelé *reporting*
[4]appelé *polling*
[5]appelé *tuning*

4.1.1 Modélisation de la gestion de réseaux

Une infrastructure de supervision de réseau repose sur quatre modèles qu'elle doit instancier [151]. Pour chacun d'entre eux, nous proposons une définition ainsi que quelques exemples d'utilisation :

Le modèle de l'information : C'est le choix d'un langage commun de description des ressources administrées, exprimées sous la forme d'objets gérés. Actuellement, différents langages existent : L'approche fondée sur SNMP[6] utilise SMIv2[7] [22], WBEM[8] utilise CIM [41] que nous présentons dans la section 5.2, la gestion par politique utilise SPPI[9] [109].

Le modèle de l'organisation : Il détermine le rôle de chaque élément entrant dans le cadre de la gestion. Si les premières infrastructures de gestion de réseau reposaient exclusivement sur une organisation centralisée, on utilise maintenant des approches hiérarchiques [148] ou distribuées [83, 25]. Par exemple, SNMP utilise une organisation constituée de gestionnaires et d'agents. De manière similaire, la gestion par politiques [91], utilise des points d'application des politiques appelés PEP[10] et des points de décision, appelés PDP[11]. Nous détaillons les différentes propositions d'organisation du plan de gestion existantes dans la section 4.1.2.

Le modèle de la communication : C'est le choix d'un protocole de communication entre les différents éléments qui puisse véhiculer les valeurs des objets gérés. SNMP [22] est *le* protocole standard déployé dans les infrastructures actuelles de gestion des réseaux. La gestion par politique utilise COPS[12] [52] et COPS-PR[13] [24].

Le modèle fonctionnel : En termes de critères de fonctionnalité de la gestion de réseaux, les besoins recensés par les utilisateurs sont assez variés. Certains accordent un fort intérêt à l'automatisation de la gestion alors que d'autres se concentrent sur l'aspect sécurité. L'ISO[14] a répertorié et défini cinq aires fonctionnelles pour la gestion des réseaux [78, 151, 58] :

- La gestion des fautes : Elle consiste à détecter, isoler et corriger les anomalies qui se produisent dans l'environnement réseau.
- La gestion comptable : Elle permet d'estimer puis de comptabiliser le coût lié à l'utilisation des infrastructures et services. Dans le cas d'un service payant, elle comprend aussi la facturation du service offert.
- La gestion de la configuration : Elle s'occupe du recensement des différents composants d'un réseau, de leur configuration physique et logicielle. En outre, elle consiste aussi à planifier et mettre en œuvre de nouveaux équipements et mettre à jour les configurations existantes.
- La gestion de la performance : Elle est relative à l'évaluation de la qualité du service délivré par le réseau et aux mécanismes de contrôle qui permettent de la maintenir. Les critères de qualité sont vastes et incluent la rapidité, la disponibilité ou la simplicité de fonctionnement.

[6] Simple Network Management Protocol
[7] Structure of Management Information version 2
[8] Web Based Enterprise Management
[9] Structure of Policy Provisioning Information
[10] Policy Enforcement Point
[11] Policy Decision Point
[12] Common Open Policy Service
[13] COPS for PRovisioning
[14] International Organization for Standardization

– La gestion de la sécurité : Elle permet de garantir la protection, l'intégrité et le bon usage des services. Elle consiste donc à détecter les intrusions, contrôler l'accès aux ressources et vérifier leur intégrité.

4.1.2 Les approches standard pour la gestion des réseaux et services

Actuellement, on recense plusieurs approches pour la gestion des réseaux et services [83]. Parmi celles-ci, nous en présentons ici six qui sont les plus répandues [107].

L'approche centralisée : C'est l'approche qui fut développée dans les premières infrastructures de gestion de réseaux. Initialement composée d'agents et d'une application de gestion (figure 4.2.a) elle a ensuite intégré un élément d'agrégation intermédiaire appelé gestionnaire qui apporte une séparation logique des éléments et une meilleure extensibilité (figure 4.2.b). Les limites de cette approche reposent naturellement sur la centralisation exclusive des opérations effectuées par le gestionnaire et l'application de gestion : si l'une de ces entités disparaît, c'est toute l'infrastructure de gestion qui s'écroule.

L'approche hiérarchique : Pour repousser les limites atteintes par le modèle centralisé tout en conservant la notion d'autorité centrale, l'introduction d'une hiérarchie de sous-gestionnaire [148, 66] fut proposée (figure 4.2.c). Chaque sous-gestionnaire a pour fonction d'agréger les données de gestion issues du niveau inférieur pour en fournir une vue synthétique, accessible par un gestionnaire de niveau supérieur. Un exemple de déploiement sur un réseau local consiste à placer un sous-gestionnaire par sous-réseau IP et un gestionnaire central pour l'ensemble du réseau. L'utilisation de cette hiérarchie introduit un premier degré de décentralisation qui permet de mieux répartir la charge de gestion et d'améliorer la tolérance aux fautes de l'infrastructure.

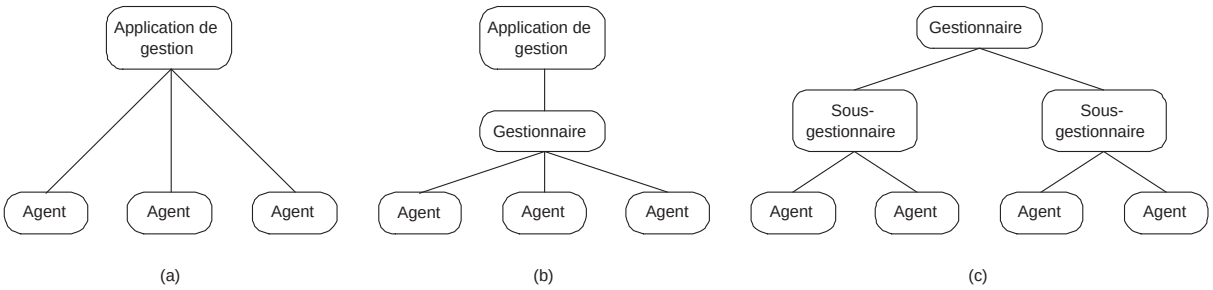


FIG. 4.2 – Différentes approches pour la gestion. (a) L'approche centralisée. (b) L'approche centralisée avec séparation des composants. (c) L'approche hiérarchique.

La gestion par délégation : La gestion par délégation [168] propose de déléguer les opérations de gestion effectuées par le gestionnaire aux agents. Cette idée part de la constatation suivante : dans les approches de gestion centralisée et hiérarchique, chaque opération de gestion nécessite de nombreux échanges entre le gestionnaire et les agents. En outre, l'exécution de plusieurs opérations de gestion simultanées requièrent la mobilisation de ressources importantes chez le gestionnaire qui centralise toutes ces tâches. La gestion par délégation propose de pourvoir les agents de programmes rudimentaires qui puissent être exécutés localement, déchargeant ainsi le gestionnaire de cette tâche. L'installation et l'exécution des programmes chez les agents sont contrôlées par le gestionnaire qui, une fois celles-ci terminées, peut récupérer leur résultat. La gestion par délégation apporte

donc un degré supplémentaire de décentralisation de la fonction de gestion, induisant une augmentation de l'autonomie chez les agents. En outre, comparativement aux approches centralisées ou distribuées, elle accroît considérablement le facteur d'échelle des infrastructures de supervision.

Le code mobile : C'est une généralisation de la gestion par délégation qui consiste à pouvoir dynamiquement télécharger et exécuter un programme sur les éléments administrés [51, 14]. Deux niveaux de mobilité sont recensés. Le niveau *faible* qui représente un programme mobile dont l'exécution complète ne peut être faite que sur un seul hôte à la fois. Le niveau *fort* désigne les programmes mobiles qui peuvent migrer en cours d'exécution et reprendre leur déroulement sans être affectés par leur déplacement.

L'utilisation du code mobile intervient dans trois domaines qui sont : (1) l'évaluation à distance qui consiste, pour un client, à donner le nom d'une méthode, ses paramètres, mais aussi son code au serveur qui va l'exécuter, (2) le code à la demande qui permet à un client, selon les besoins, de contacter un serveur pour télécharger un programme nécessaire à l'accomplissement d'une tâche, et (3) les agents mobiles qui sont des entités autonomes capables de migrer d'un hôte à un autre tout en conservant le cours de leur exécution. Ces approches sont aujourd'hui considérées dans le domaine de la supervision des réseaux et leurs caractéristiques confèrent un fort degré d'autonomie aux éléments qui l'utilisent.

La gestion par politiques : La gestion par politiques [91, 5] est une approche de gestion qui permet à un administrateur de s'abstraire des niveaux bas inhérents aux équipements pour se concentrer sur la définition de règles de la forme `condition -> action` qui spécifient le comportement que les composants du réseau doivent appliquer. Initialement, cette approche avait pour objectif de gérer des protocoles tels que RSVP[15] qui signalent la réservation de QoS[16]. Cette approche attribue de nouveaux rôles aux éléments qu'elle considère. Le PDP est un point de décision de politiques qui a pour rôle de récupérer des politiques de haut niveau, de vérifier leur cohérence, les traduire en commandes interprétables par les équipements et de les distribuer aux entités qu'il administre. Le PEP est un point d'application de politiques qui s'exécute sur les équipements gérés et est sous l'autorité d'un PDP.

On distingue deux modèles de fonctionnement des politiques. Le modèle de délégation stipule que pour tout évènement qui requiert une décision, un PEP va interroger son PDP, qui décidera alors du comportement à adopter. Le modèle par provision, quant à lui, ajoute un degré supplémentaire d'autonomie chez les PEPs. Il consiste à pourvoir les PEPs d'une base de données de politiques appelée PIB[17] qu'ils vont appliquer de manière autonome, sans nécessiter d'intervention du PDP. Le rôle du PDP consiste alors à gérer les bases de données qu'il provisionne dans les PEPs.

L'auto-gestion : C'est l'approche de gestion la plus récente. Le travail de Jeffrey O. Kephart et al. [88] met en évidence l'augmentation de la complexité des infrastructures de gestion actuelles et les limites qu'elles présentent pour une utilisation faite par un opérateur humain. En effet, les infrastructures de réseaux se diversifient, les services disponibles sont de plus en plus nombreux et leurs interactions de plus en plus fréquentes, et la QoS requise par les usagers toujours plus forte. Cette évolution rend la tâche de supervision de plus en plus complexe pour un administrateur et l'utilisation de mécanismes d'auto-gestion devient incontournable. L'auto-gestion consiste à définir des politiques qui spécifient les conditions

[15] Resource reSerVation Protocol
[16] Quality of Service
[17] Policy Information Base

de bon fonctionnement du réseau en fonction des objectifs à atteindre et à laisser ensuite le réseau gérer ces règles de manière autonome.

Une infrastructure d'auto-gestion s'applique à quatre grands domaines qui sont :

- L'auto-configuration : c'est la capacité des équipements et des services à s'auto-configurer en fonction des politiques et du contexte actuel ;
- L'auto-optimisation : elle permet aux équipements d'entreprendre une démarche de recherche continuelle d'optimisation qui accroîsse leur performance et leur efficacité ;
- L'auto-réparation : elle consiste à permettre aux systèmes de détecter, diagnostiquer et réparer par eux-mêmes un quelconque problème de niveau matériel ou logiciel ;
- L'auto-protection : elle permet aux systèmes de se défendre de manière indépendante contre les attaques ou les propagations d'erreurs.

4.2 Besoin de gestion du modèle P2P

Avant de présenter les différents travaux relatifs à la supervision des réseaux P2P, nous exprimons la motivation de notre travail, à savoir, le besoin de gestion relatif à ce modèle de services.

Les applications P2P, de par leur totale décentralisation et leur fonctionnement hors de tout contrôle, semblent ne pas nécessiter de mécanismes de supervision. En particulier, les applications de partage de fichiers semblent satisfaire leurs usagers et laissent croire que de manière générale, le modèle P2P n'a aucunement besoin d'être géré. Cette affirmation est fausse pour deux raisons : la première est que dans une application P2P de type Kazaa, Emule, ..., le service offert ne garantit aucune qualité. On peut le qualifier de service *Best Effort*. L'accès aux données n'est pas fiable et le contenu des données encore moins. Ainsi, ce type d'application fonctionne, mais son niveau de qualité n'est pas garanti. La seconde raison qui contrecarre cette affirmation réside dans l'intégration transparente de mécanismes de gestion minimaux dans les services de partage de fichiers. Lorsqu'on exécute Kazaa, s'il n'existe pas explicitement d'agent de gestion qui se connecte à une infrastructure de supervision pour gérer l'application, on trouve tout de même certains mécanismes qui tendent de manière intégrée à assurer une qualité minimum. Ce sont par exemple les mécanismes incitatifs qui forcent l'usager à participer activement au service en partageant ses propres fichiers, qui peuvent d'ailleurs avoir été préalablement téléchargés. Dans le cas d'une participation active, une recherche plus performante et un téléchargement optimisé sont alors fournis. Ces mécanismes, qui relèvent de la gestion de service, sont complètement intégrés et montrent que même les services *a priori* les plus réticents à la supervision en intègrent certains fonctionnements.

En outre, on peut généraliser le cas des applications de partage de fichiers à l'ensemble des applications P2P. Certains considèrent en effet que les applications P2P n'ont pas besoin de supervision car les mécanismes de la gestion sont intrinsèquement intégrés au modèle. Là aussi, cette affirmation est fausse car s'il est juste de considérer que le modèle P2P assure l'auto-organisation de ses éléments de manière à maintenir la fourniture d'un service, l'auto-organisation n'est pas pour autant synonyme d'auto-gestion. Pour illustrer ce propos, reprenons l'exemple d'une application de partage de contenu. Cette fois, nous imaginons qu'elle soit déployée dans le cadre d'une entreprise où des collaborateurs distants travaillent sur des données partagées. Dans ces conditions, on comprend que l'accès aux ressources doit être garanti à chaque instant. Or étant donné les allers et venues de chaque participant, la disponibilité des ressources va varier, et si l'ajout de mécanismes de redondance va permettre d'augmenter le taux de disponibilité des données, en aucun cas ils vont pouvoir garantir leur parfaite disponibilité.

Ainsi, nous estimons que l'utilisation du modèle P2P, dans des conditions où le niveau de service fourni à l'usager doit être garanti, nécessite l'intégration d'une infrastructure de supervision.

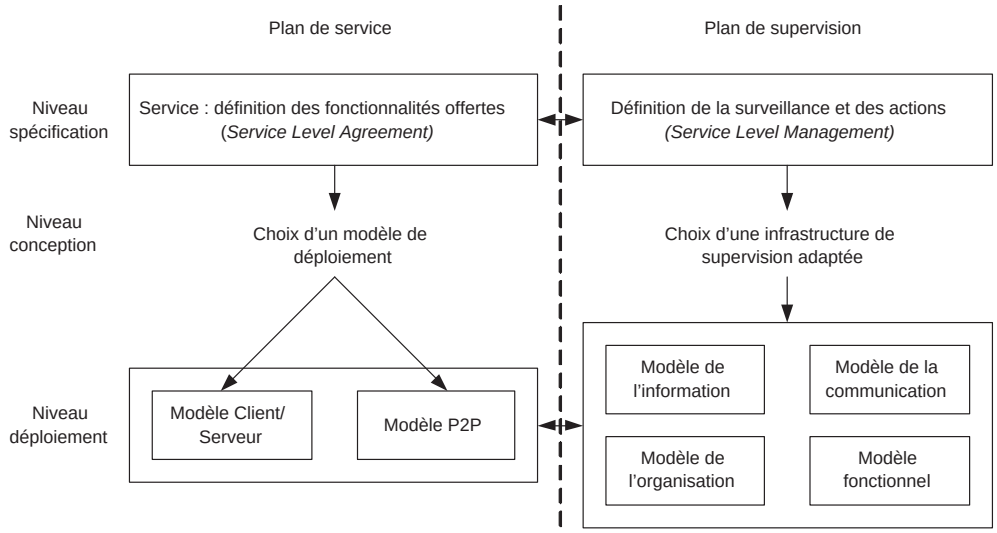


FIG. 4.3 – Etapes de mise en œuvre d'un service et de sa supervision

D'un point de vue de la gestion, la manière dont nous percevons le modèle P2P est celle d'un modèle de déploiement de service qui, tout comme le modèle client-serveur, nécessite une infrastructure de supervision pour garantir le bon fonctionnement du service qu'il offre. La figure 4.3 illustre cette vision. Elle représente les différentes étapes abstraites qui constituent la mise en œuvre d'un service et de sa supervision. La spécification d'un service définit l'ensemble des fonctionnalités offertes à l'usager qui l'utilise. Dans un cadre payant ou à contrainte de qualité, il résulte de cette spécification l'établissement d'un contrat de service[18] entre l'usager et le fournisseur. Sur le plan de la supervision, il faut définir alors les aspects du service à garantir, les indicateurs qu'il faut surveiller, et les actions à entreprendre pour assurer ces garanties. Ces démarches entrent dans le cadre de la gestion de service appelée SLM[19]. Une fois le service spécifié, une phase de conception s'intéresse à la manière de mettre en œuvre ce service. A ce stade, pour les services distribués, deux possibilités d'infrastructure de déploiement sont envisageables : le modèle client-serveur et le modèle P2P. Parallèlement, au niveau de la gestion, on va choisir quatre technologies qui vont instancier les quatre modèles qui forment une infrastructure de gestion. En conclusion, le modèle P2P n'échappe pas à la supervision. Il est un simple modèle de déploiement possible qui peut être considéré parmi d'autres pour la mise en œuvre d'un service.

Etant donné les motivations exposées ci-dessus, notre travail consiste à concevoir et mettre en œuvre une infrastructure de supervision pour les réseaux et services P2P qui repose sur des approches de gestion standard.

[18]Service Level Agreement
[19]Service Level Management

4.3 Approches actuelles

On compte actuellement plusieurs travaux qui portent sur la supervision des réseaux P2P. Ceux-ci sont souvent motivés par le problème de la libre consommation[20] [4] qui montre que 70% des utilisateurs d'applications de partage de fichiers ne partagent aucune donnée. En outre, une étude a montré que, dans un réseau Gnutella, environ 50% des réponses de requêtes de recherche de fichier sont données par 1% des nœuds.

On distingue deux grandes classes d'approches pour résoudre ce problème. La première repose sur les modèles économiques et la seconde, sur des mécanismes de réputation et de confiance. Dans cette section, nous présentons différents travaux-phares de chacune de ces deux approches. Pour terminer, nous présentons une proposition de mise en œuvre d'infrastructure de gestion pour la plate-forme Jxta.

4.3.1 Approches incitatives

Les approches incitatives ont pour objectif de mettre en place un mécanisme qui motive les usagers à participer activement à leur communauté de sorte à entretenir voire accroître la qualité du service offert.

Les modèles économiques

La supervision des infrastructures P2P par l'utilisation de modèles économiques repose sur l'utilisation de modèles de marché issus des domaines de l'économie et des finances. Nous choisissons ici de présenter celui de Golle et al. [67] qui repose sur la théorie des jeux et s'applique à un service de partage de fichiers de type Napster. Durant chaque tour du jeu[21], chaque pair peut effectuer deux actions qui sont le partage de fichiers, noté σ, et le téléchargement, noté δ. Chacune de ces actions peut être faite à différents niveaux qui sont : aucun (noté σ_0 pour le partage et δ_0 pour le téléchargement), modéré (σ_1 et δ_1) et fort (σ_2 et δ_2). A chaque tour, un pair i choisit donc sa stratégie $S_i = (\sigma, \delta)$. On appelle $\Sigma = \{(\sigma, \delta), \ldots, (\sigma, \delta)\}$ l'ensemble des stratégies adoptées par les pairs lors d'un tour.

La stratégie d'un pair dépend du profit qu'il va pouvoir tirer du système. Ce profit, appelé fonction d'utilité, prend en compte six critères de contrainte et de satisfaction. Ceux-ci sont :

- La quantité de données téléchargées (AD) ;
- La diversité des données accessibles (NV) ;
- L'espace disque utilisé pour le partage de données (DS) ;
- La bande passante requise pour le partage (BW) ;
- L'altruisme (AL) ;
- Les compensations financières (FT).

Etant donné ces critères, le profit d'un pair peut se modéliser par la fonction : $U_i = (f_i^{AD}(AD) + f_i^{NV}(NV) + f_i^{AL}(AL)) - (f_i^{DS}(DS) + f_i^{BW}(BW)) + FT$ où chaque fonction f_i est une fonction qui décrit la perception de chaque pair pour le critère concerné. Chaque pair va ainsi essayer de maximiser cette fonction.

En l'absence de mécanisme incitatif, Golle et al. montrent que la stratégie globale $\Sigma = \{(\sigma_0, \delta_2), \ldots, (\sigma_0, \delta_2)\}$ est un équilibre strict et dominant au sens de Nash [118]. Cela signifie que toutes les autres stratégies conduisent forcément à une perte du profit et que dans ces conditions, la stratégie (σ_0, δ_2) est la meilleure pour un agent, quelles que soient les autres stratégies.

[20] on trouve également le terme de pillage pour désigner ce phénomène

[21] qui peut être une heure, un jour, un mois, ... en fonction du contexte

Par contre, l'ajout d'un mécanisme incitatif permet de changer l'équilibre du système. Deux mécanismes sont proposés par Golle et al. Le premier repose sur un système de micro-paiements. Le téléchargement d'un fichier rapporte β Euros au pair qui en est la source et coûte la même somme à celui qui le télécharge. A la fin de chaque tour, si un pair a téléchargé t fichiers et en a fourni u, il reçoit ou paie $|\beta(u-t)|$ Euros. Le second mécanisme incitatif propose non pas de récompenser les pairs dont les fichiers ont été téléchargés, mais ceux qui partagent le plus d'espace mémoire. Dans les deux cas, on montre que le nouvel équilibre strict s'applique à la stratégie $\Sigma = \{(\sigma_2, \delta_2), \ldots, (\sigma_2, \delta_2)\}$ et donc que l'ajout d'incitations accroît la qualité du service.

Le cas d'étude considéré par Golle et al. est celui de Napster, qui repose sur un modèle P2P centralisé où le serveur central dispose d'une connaissance globale des agissements de chacun. Dans le cadre du projet MMAPPS[22], [73] propose d'utiliser des jetons comme monnaie d'échange des fichiers téléchargés. La nouveauté de cette proposition repose sur la nature décentralisée de l'infrastructure de gestion des jetons. D'ailleurs, une étude de Antoniadis et al. [11] montre aussi qu'il n'est pas nécessaire d'avoir une connaissance globale et centrale des agissements des pairs pour mettre en place un modèle économique efficace. Une connaissance partielle est suffisante pour appliquer différentes méthodes de marché et obtenir un résultat satisfaisant sur le plan de l'équilibre entre le coût lié à l'utilisation d'un mécanisme incitatif et la qualité du service offert.

Actuellement, on trouve un nombre assez conséquent d'autres travaux qui proposent d'utiliser des modèles économiques pour inciter les pairs à participer. Parmi ceux-ci, on en retient trois qui considèrent des approches différentes. Le premier [10], propose un modèle économique assez similaire à celui de Golle et al. mais il met en évidence deux stratégies qui sont fondées sur le monopole, pour la première et la concurrence pour la seconde. Les deux autres travaux sont fondés sur la théorie des jeux. Le premièr [56] propose une version généralisée du dilemme du prisonnier. Le second [19] se différencie des approches actuelles en proposant non pas l'insertion de monnaie fictive ou réelle dans une communauté, mais plutôt par la différenciation de la qualité du service offert aux pairs. Par exemple, elle propose d'utiliser une probabilité d'acceptation ou de rejet des requêtes fondée sur le niveau de participation du pair qui les initie.

Les systèmes de confiance et de réputation

Les propositions actuelles de systèmes de confiance et de réputation qu'on trouve dans les infrastructures P2P sont similaires à celles qui sont utilisées dans les services de commerce électronique comme EBay [129]. Ils ont pour objectif de permettre une évaluation du comportement des pairs qui permette à la communauté à laquelle ils appartiennent de voir sa qualité de service améliorée. Par leur utilisation, chaque pair peut effectuer un classement des autres pairs et travailler prioritairement avec ceux qui sont bien évalués. Ainsi, les pairs ayant un comportement qui ne satisfait pas les règles de leur communauté se voient, de fait, exclus.

On trouve actuellement de nombreuses propositions qui permettent d'évaluer le bon comportement d'un pair. Ces propositions utilisent les termes de confiance et de réputation. Pour chacun de ces termes, Wang et al. [164] proposent une définition que nous rappelons maintenant :

Définition 6 *La confiance qu'un pair a envers un autre représente l'estimation personnelle qu'il fait de l'honnêteté, la fiabilité et les capacités de cet autre pair.*

Définition 7 *La réputation qu'un pair a est fondée sur les recommandations, éventuellement basées sur la confiance, que d'autres pairs font sur lui.*

[22] Marker MAnagement of Peer-to-Peer Services - http ://www.mmapps.org

D'après les travaux actuels, on s'aperçoit que la mise en œuvre d'un système incitatif fondé sur la confiance et la réputation doit définir et instancier :

Une métrique d'évaluation de la confiance : Si les propositions actuelles estiment la confiance de manière différente, elles s'accordent toutes sur le fait que la confiance dépend du contexte dans lequel elle s'intègre. Aberer et al. [2] illustrent cette notion avec l'exemple suivant : *"Je fais confiance à mon médecin pour me donner des conseils sur mes problèmes de santé, mais pas sur des questions financières."* Les propositions de métriques sont donc relatives à un certain contexte et sont aussi composées de facteurs qui varient d'une approche à une autre. Par exemple, [71] scinde sa métrique en deux parties. La première est relative au comportement du pair, avec sa contribution au routage et à la découverte, et sa participation en tant que serveur. La seconde concerne ses ressources exprimées en termes de puissance de calcul, bande passante, espace de stockage et mémoire vive. Xiong et al. [166] utilisent trois paramètres qui sont le taux de satisfaction que le pair donne lorsqu'il sert une requête, le nombre d'interactions qu'il a effectuées avec les autres pairs, et la manière dont il est perçu auprès des autres pairs.

Une infrastructure de stockage et d'accès aux données : Elle a pour rôle de stocker l'ensemble des données relatives à l'évaluation des pairs qui entrent dans le cadre de la métrique choisie. Les caractéristiques du modèle P2P impliquent l'utilisation d'une infrastructure distribuée pour assurer cette fonction, qui est souvent fondée sur une DHT. Par exemple, PeerTrust [166] et la proposition de Aberer et al. [2] reposent sur P-Grid [1].

Un mécanisme de sécurité : Face aux mécanismes incitatifs, des pairs mal intentionnés peuvent tenter plusieurs formes d'attaques qui nécessitent la mise en place de mécanismes de sécurité [144]. Une des attaques possibles concerne la corruption des données stockées. Les pairs attaquant l'utilisent pour faire croire que leur réputation est excellente. Un mécanisme de sécurité qui assure l'intégrité des données inhérentes à la réputation est donc nécessaire. Plusieurs solutions sont envisagées pour assurer cette fonction. La plus simple repose sur la duplication des données [166]. Si l'infrastructure utilise une DHT, chaque donnée est hachée plusieurs fois de manière à générer plusieurs clés, qui d'après les propriétés de la fonction de hachage, vont être référencées par des nœuds différents. Ainsi seule la coopération d'un ensemble de nœuds pourrait permettre la corruption des données stockées. Une seconde solution repose sur l'utilisation de mécanismes de cryptage et de signature qui empêchent la lecture et la modification des données.

Un second type d'attaque visant les infrastructures fondées sur la réputation concerne le changement d'identité des pairs. Un pair mal intentionné et ayant une mauvaise réputation peut vouloir changer d'identité régulièrement pour ne pas être pénalisé et continuer à nuire au réseau qu'il rejoint. Un mécanisme d'authentification qui permette de contourner ce changement d'identité doit donc être mis en place. Des solutions centralisées et décentralisées existent et permettent de résoudre ce problème [106].

Les solutions pour la sécurité des infrastructures de réputation qui utilisent des clés posent le problème de leur génération et de leur distribution. Seule une autorité centrale pourrait effectuer ces opérations, mais elle s'apparente à un point central facilement attaquable dans le contexte distribué qu'est le P2P [71, 106]. Cette question reste ouverte.

4.3.2 Gestion de la topologie virtuelle

La majorité des applications P2P définissent une topologie fondée sur un réseau virtuel construit au-dessus des réseaux physiques et logiques. Le maintien de cette topologie en accord

avec les besoins de l'application qu'elle supporte est un élément crucial pour la garantie du service offert aux usagers. Ainsi, l'utilisation d'une infrastructure capable de gérer cette topologie s'impose. Plusieurs travaux proposent des solutions qui accomplissent cette tâche. Ils sont généralement adaptés à un type de service particulier, comme le partage de fichiers ou le calcul distribué. Néanmoins, ils proposent tous des solutions pour des topologies non structurées. En effet, les infrastructures fondées sur des topologies structurées, telles que les DHTs, utilisent directement leur processus de maintenance pour assurer le maintien topologique. Par contre, dans les topologies non structurées, les allers et venues des pairs peuvent conduire à des phénomènes de groupement autour de points centraux[23], le partitionnement d'une communauté en sous-ensembles disjoints ou la dégradation du routage du fait de l'augmentation anormale du diamètre du réseau.

Parmi les différentes propositions existantes, nous en présentons quatre qui reposent sur des principes différents. Nous détaillons particulièrement Cyclon [162], une proposition faite par Voulgaris et al. Cyclon est une infrastructure de gestion de la topologie des réseaux P2P non structurés qui repose sur un algorithme de mélange[24]. Cet algorithme permet à chaque pair d'échanger avec d'autres une partie de sa table de routage de sorte à ne conserver que les entrées les plus récentes. Son utilisation procure de bonnes propriétés, similaires à celles des graphes aléatoires, à la topologie qu'il établit. Parmi celles-ci, on compte un faible diamètre, un faible taux de *clustering*, une répartition équilibrée du degré de connectivité et une très bonne résistance à la disparition massive de nœuds. Ce type de topologie est parfaitement adapté pour les applications de partage de fichiers qui reposent sur la diffusion des informations sous forme de rumeur [80].

Les trois autres approches que nous présentons ont le même objectif que Cyclon. Toutefois, les moyens qu'elles utilisent diffèrent. Par exemple, les travaux de Cooper et al. [27] proposent, au sein d'une communauté, de casser régulièrement les connexions entre les pairs en fonction de leur charge. Pour cela, deux types de relations topologiques ont été identifiées. La première représente un lien de recherche. Entre deux pairs, elle illustre le fait qu'une recherche initiée ou reçue par le premier soit envoyée au second. Le second type de relation topologique représente un lien d'index. Il signifie qu'un pair cache l'index des données accessible sur un autre. Lorsque le réseau évolue, ces deux relations topologiques induisent une charge de travail variable pour chaque pair. Par exemple, un nœud peut avoir à faire suivre beaucoup de requêtes de recherche ou cacher un index de données très important. Cooper et al. proposent ainsi, en fonction de la charge d'un nœud et de ses ressources, de rompre et de reconstruire régulièrement des relations topologiques. Cette idée a été validée par des simulations qui montrent clairement le gain de cette approche par rapport aux topologies construites de manière aléatoire. Elle semble donc propice aux applications de type Gnutella.

Ensuite, une proposition de Di Ferdinando et al. [36] utilise la gestion par politiques pour construire une topologie où les nœuds se regroupent selon leur politique locale. L'idée de ce travail est de définir une ou plusieurs politiques pour une communauté que chaque nœud devra appliquer. Lorsqu'un nouveau nœud rejoint la communauté, il doit être capable d'appliquer la politique en vigueur proposée par le nœud qu'il contacte. Dans le cas contraire, plusieurs tentatives vont lui permettre de contacter d'autres nœuds, qui possèdent éventuellement des politiques différentes. S'il ne correspond à aucune politique, il est alors rejeté. L'objectif de ce travail est de construire une topologie où chaque nœud participe activement, évitant ainsi des situations où un nœud purement consommateur se trouve comme point central de la topologie,

[23] phénomène de *clustering*
[24] *shuffle*

diminuant ainsi les performances du système. Néanmoins, le manque de résultats et d'évaluation du coût de mise en œuvre de cette proposition la rendent peu crédible.

La dernière proposition à laquelle nous nous intéressons utilise une technologie active appelée AVP[25]. Dans leur travail, Kouroulis et al. [92] ne proposent non pas une manière de construire une topologie non structurée efficace, mais plutôt la manière d'utiliser une topologie donnée au mieux. La proposition est intégralement fondée sur Gnutella et pour ce cas, les auteurs proposent plusieurs actions qui permettent d'améliorer son fonctionnement. Elles sont : la segmentation de la topologie virtuelle en sous-domaines qui permet l'optimisation du routage, le contrôle de la topologie qui peut être fondé sur une métrique quelconque définie par l'application P2P, et la gestion des ressources par l'utilisation de caches.

4.3.3 Monitorage de la plate-forme Jxta

Le projet MMP[26] [75] est un projet de la communauté Jxta qui vise à concevoir et mettre en œuvre une infrastructure de supervision pour l'implantation Java de la plate-forme Jxta. MMP comporte trois contributions qui sont : l'instrumentation de la plate-forme, la définition d'un protocole de gestion et la mise en œuvre d'une application de supervision. Un des intérêts du projet MMP repose sur l'intégration totale des fonctions de supervision dans la plate-forme : MMP est écrit en Java, il utilise les mécanismes de Jxta pour la communication et son application de supervision est hébergée par un pair. La gestion se trouve ainsi au coeur du service.

L'instrumentation de Jxta concerne huit services relatifs aux principales fonctions offertes par la plate-forme qui sont la communication (*Endpoint Service*, *TCP, HTTP, Router Modules*, et *Pipe Service*) et la découverte de ressources (*Rendez-vous Service*, *Resolver Service*, *Discovery Service*). Pour chacun de ces services, un ensemble complet de métriques a été défini. Un aperçu de ces métriques est donné dans les figures 8.3 et 8.6 du chapitre 8.

L'architecture MMP [76] telle qu'elle est déployée dans les pairs Jxta est représentée sur la figure 4.4. Le service qu'elle fournit est délivré par le composant appelé *PeerInfoService* qui peut recevoir des requêtes de *reporting* sur un ensemble de services dont la nature est spécifiée à l'aide d'un filtre appelé *MonitorFilter*. A ces requêtes, le service *PeerInfoService* fournit un rapport (*MonitorReport*) qui contient les valeurs prises par les métriques relatives aux services spécifiés dans la requête. Le protocole utilisé pour véhiculer les requêtes et les rapports est une version étendue du *Peer Information Protocol*. Enfin, on peut noter que le *PeerInfoService* répond aussi bien à des requêtes concernant le pair local que des pairs distants. Ainsi, dans un groupe, un pair peut agir comme un *proxy* qui reçoit de la part d'une application de gestion l'ensemble des requêtes de *reporting* et les fait suivre aux pairs concernés.

Le *MonitorManager* est un composant, implanté dans la plate-forme sous la forme d'un module. Sa fonction concerne l'orchestration des différents composants de monitorage des services de la plate-forme (*ServiceMonitor*). En effet, à chaque service de la plate-forme est adjoint un composant *ServiceMonitor* responsable de traiter et de répondre aux requêtes envoyées par le *MonitorManager*.

Enfin, l'application de gestion [77] proposée par MMP offre à un administrateur une interface graphique de monitorage de la plate-forme. Chaque métrique est accessible et surveillable par le biais d'un graphique qui trace son évolution au cours du temps. En outre, des seuils assignables aux métriques permettent de détecter des dysfonctionnements et fixer des limites aux différents composants de la plate-forme. D'un point de vue critique, cette application n'offre qu'une simple interface de monitorage. Aucune action de contrôle, effectuée par l'application de gestion, n'est

[25] Active Virtual Peer

[26] Metering and Monitoring Project - http ://meter.jxta.org

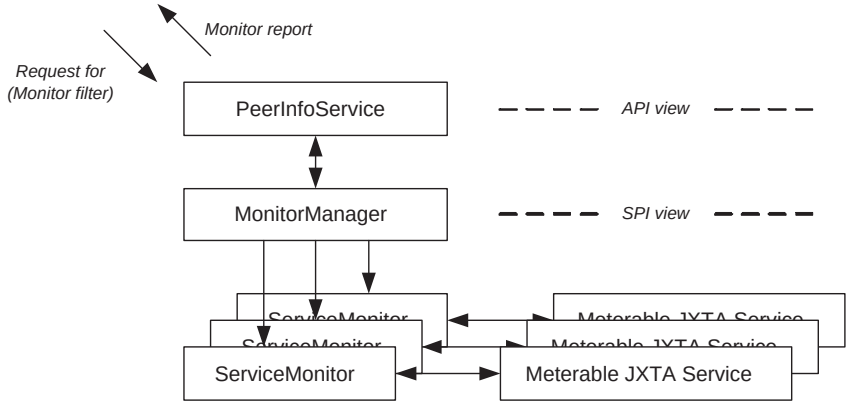


FIG. 4.4 – *Extrait de [76].* L'architecture MMP

possible sur la plate-forme. De plus, elle n'offre pas de fonction d'agrégation ou de synthèse sur un groupe des données récupérées. Elle se contente d'une surveillance individuelle des pairs.

4.4 Utilisation du P2P pour la gestion de réseaux

Dans les sections précédentes, nous nous sommes intéressés à des propositions d'infrastructure de gestion dédiées à des services construits selon le modèle P2P. Nous nous intéressons maintenant à l'idée inverse, c'est à dire, l'utilisation du modèle P2P pour construire une infrastructure de gestion.

Dans son travail de thèse, Alexander V. Konstantinou [90] s'est penché sur la réalisation d'une infrastructure d'auto-gestion reposant sur une infrastructure P2P pour la supervision des réseaux et services. Celle-ci est composée d'éléments autonomes dont le comportement est dicté par des politiques. Les trois contributions qu'il propose sont :

Une architecture de gestion autonome : Elle se scinde en deux niveaux : le niveau inférieur concerne les données de gestion et en particulier leur stockage et leur accès. Pour cela, une interface abstraite appelée *Modeler* permet de stocker et accéder aux données relatives aux équipements et aux services gérés. Les objets gérés représentent des machines ou un routeur, aussi bien qu'un réseau virtuel (VLAN[27]) ou un service de système de fichiers (NFS). Le *Modeler* est en fait une base de donnée distribuée entre les participants de l'infrastructure de gestion. Un service d'abonnement permet aux éléments autonomes de recevoir des notifications sur les évènements qui se produisent au sein du réseau et qui peuvent éventuellement modifier leur comportement. Le niveau supérieur de l'infrastructure est composé des éléments autonomes qui, d'un point de vue de la gestion, agissent comme des agents et des gestionnaires. Cette fusion des deux rôles permet d'accroître la fiabilité du système en supprimant toutes les étapes nécessaires à la remontée d'information auprès d'un gestionnaire centralisé, de distribuer totalement la fonction de gestionnaire, évitant les inconvénients liées à la centralisation, et d'intégrer la gestion au coeur des éléments.

Jspoon, un langage pour l'autonomie : Il se formalise par une extension de Java pour la gestion des éléments autonomes. De nouveaux mots-clés permettent de déclarer des at-

[27] Virtual Local Area Network

tributs et des méthodes comme étant relatifs à la gestion et de les rendre visibles par l'infrastructure. Un compilateur et un environnement d'exécution particulier permettent d'intégrer ces modifications au langage Java. L'intérêt de ce langage réside dans son intégration à Java qui est un langage de développement standard dans les applications réseaux. Ensuite, il permet de généraliser la notion d'exception à des machines virtuelles distantes, permettant ainsi de propager des alarmes sur le plan de supervision. Enfin, son extensibilité permet d'ajouter de nouvelles données et fonctionnalités au langage afin de représenter au plus près les objets gérés.

Un système de propagation des changements : Dans un système autonome, les changements subis par un élément ont une répercussion sur d'autres. Pour effectuer cette répercussion, un langage de définition de règles de propagation appelé OSL[28] a été conçu et mis en œuvre dans les éléments supervisés. Il permet de lier la configuration et l'état de certains éléments à celui d'autres. Un des problèmes de ce type de langage est la convergence vers un état cohérent et stable des éléments sujets à un changement. Cette vérification doit être faite durant les étapes de conception car en cours de fonctionnement, aucun opérateur ne doit intervenir pour stopper par exemple des boucles ou résoudre des états incohérents. Pour aider à cette vérification, une extension d'OSL appelée OPL[29] a été définie. Elle permet de définir des règles d'état que les éléments supervisés doivent appliquer quels que soient les changements.

La validité de ces trois propositions a été prouvée par leur mise en œuvre dans un prototype appelé Nestor. Celui-ci a été appliqué à plusieurs contextes de gestion autonome. Un de ceux-ci est relatif à l'automatisation des changements entre un serveur DHCP[30] et un serveur DNS. Les machines dont l'adresse IP est dynamique et attribuée par le biais d'un serveur DHCP sont difficilement référençables dans un serveur DNS. L'utilisation de la gestion autonome permet de répercuter chaque changement d'adresse d'une machine dans le serveur DNS et ainsi de rendre la machine référençable quelle que soit son adresse IP.

Dans cette section, nous avons présenté les différentes approches existantes pour la supervision des réseaux et services P2P. Dans la section suivante, nous proposons une classification des applications construites selon ce modèle qui est orientée vers la supervision.

4.5 Classification des applications P2P orientée vers la gestion

Une application P2P peut être déployée de différentes manières qui dépendent du service qu'elle offre. On trouve par exemple, dans un contexte d'entreprise, des services P2P qui sont déployés sur le réseau local et sont offerts à une dizaine voire une centaine d'usagers. C'est par exemple le cas des applications de travail collaboratif ou de messagerie instantanée. Pour ce type d'application, il semble envisageable d'utiliser une infrastructure de gestion centralisée pour assurer la supervision du service. D'un autre côté, on trouve aussi des applications P2P déployées à l'échelle de l'Internet avec des millions d'usagers, qui sont des particuliers répartis sur des domaines administratifs différents. On comprend facilement que la supervision de ce type de service ne va pas pouvoir se faire de manière centralisée ni nécessiter l'intervention d'un opérateur humain. Des solutions distribuées et autonomes semblent plus appropriées.

[28] Object Spreadsheet Language
[29] Object Policy Language
[30] Dynamic Host Configuration Protocol

Ces deux exemples, montrent clairement que la supervision d'un service P2P varie fortement en fonction du type de service et de son environnement de déploiement. Il nous a ainsi semblé nécessaire de différencier les applications P2P d'un point de vue de la gestion. Nous proposons donc une classification de celles-ci qui repose sur cinq critères qui sont :

L'ouverture vers la gestion : Cette capacité renseigne sur le fait que des pairs présentent ou non une interface de gestion. Une application P2P peut être déployée dans un environnement où les entités acceptent d'être gérées, comme par exemple dans un réseau d'entreprise, ou dans un environnement non gérable explicitement, comme dans le cas de particuliers exécutant un client de partage de fichiers à l'échelle de l'Internet. Ainsi, dans le cadre de la gestion, si les pairs participant à un service acceptent d'être gérés, il sera possible de déployer une infrastructure de gestion ; dans le cas contraire on pourrait par exemple avoir recours à une approche intégrée au service de type incitative.

La décentralisation : D'un point de vue de la gestion, le niveau de centralisation (pur, hybride ou centralisé) d'un service P2P va influer sur la distribution et le contenu des agents de gestion. Dans le cas d'une application construite sur un modèle centralisé, on comprend aisément que la majeure partie des informations de gestion sera inhérente au serveur central. Par contre, dans le cas d'une application construite selon un modèle pur, ces informations seront complêtement distribuées parmi les pairs. Ainsi, cette caractéristique influence le modèle de déploiement de l'infrastructure de gestion.

Le réseau virtuel : Un service P2P peut être déployé au niveau applicatif et ainsi constituer un *overlay*. Dans ce cas, le nommage des entités et le routage sont effectués à ce niveau. En conséquence, l'infrastructure de gestion devra aussi être capable de fonctionner à ce niveau, en nommant ses entités et en routant ses messages au niveau de l'*overlay*.

L'échelle : L'échelle de déploiement d'une application P2P, au sens du nombre de pairs participants mais aussi des domaines sur lesquels elle est déployée, va conditionner la possibilité d'y adjoindre une infrastructure de gestion et la manière dont elle sera réalisée. Pour un service P2P exécuté au sein d'un domaine particulier, comme une entreprise ou une université, on peut envisager d'avoir recours à une architecture relativement centralisée et interagissant avec une personne physique car l'ensemble des pairs appartient à un même domaine administratif. Dans le cas où une application P2P serait déployée sur plusieurs domaines administratifs connus, sa gestion pourrait être assurée de la même manière, en établissant des accords préalables. Par contre, si elle était déployée sur un ensemble de domaines quelconques, avec un grand nombre de participants, sa gestion centralisée par l'intermédiaire d'une personne semble plus difficile et on aura plutôt recours à des mécanismes d'auto-gestion.

La dynamique : Cette caractéristique exprime le dynamisme de la présence et de l'adressage des pairs participant à une application P2P. Les limites de la dynamique sont, d'un côté, une application P2P constituée de machines toujours connectées munies d'adresses connues et fixes, et de l'autre, une application où les *peers* vont et viennent et changent d'adresse. Dans ce dernier cas, l'application de gestion devra notamment mettre en œuvre un moyen de découvrir les pairs à gérer en fonction de leurs allées et venues ainsi qu'une manière fiable et pérenne de stocker les informations de gestion.

La classification que nous proposons est illustrée par le tableau 4.5 qui recense quelques applications P2P et leur associe les critères décrits ci-dessus. Au niveau de l'ouverture vers la gestion, on peut constater que la plupart des applications actuelles ne fournissent pas une telle interface. Seule Jxta propose une véritable interface avec le projet MMP. En outre, des

opérations de gestion de configuration sont possibles sur le protocole RIP. Ensuite, le niveau de centralisation des applications P2P est souvent faible. Hormis Napster qui repose sur un serveur centralisé, toutes les applications actuelles sont construites selon un modèle pur ou hybride. Troisièmement, la situation des applications P2P en tant qu'*overlay*, définissant un nommage et routage de niveau applicatif est assez courante. Les DHTs reposent sur ce principe et sont maintenant au cœur des services P2P. L'échelle de déploiement des applications P2P actuelles est celle de l'Internet, avec des environnements multi-domaines administrés par des services différents. Enfin, concernant la dynamique, on s'aperçoit que les applications P2P reposent sur une présence et parfois un nommage dynamique des pairs. On trouve à ce jour peu d'applications qui considèrent le cas où les pairs sont statiques, mais c'est le cas de RIP [104] ou d'OceanStore [93] qui utilisent des serveurs dédiés au service.

Application	**Ouverture vers la gestion**	**Décentralisation**	**Réseau virtuel**	**Echelle**	**Dynamique**
Napster	Non	Centralisé	Non	Internet	Présence, adresses IP
Gnutella	Non	Pur	Oui	Internet	Présence, adresses IP
Pastry	Non	Pur	Oui	Internet	Présence, noms
Jxta	Oui (MMP)	Hybride	Oui	Internet	Présence
RIP	Oui	Pur	Non	Domaine (AS)	Aucune
OceanStore	Non	Pur	Oui	Internet	Aucune

TAB. 4.1 – Exemples de classification d'applications P2P

Pour conclure, on voit parfaitement dans l'exemple précédent la diversité des applications P2P actuelles et les différences qu'elles vont induire sur le plan de la supervision. Le travail de recherche que nous avons effectué considère les applications P2P qui sont (1) ouvertes à la gestion, (2) reposent sur un modèlé décentralisé ou hybride, (3) constituent un *overlay*, (4) sont déployées à l'échelle de l'Internet, et (5) comportent des pairs dont la présence et l'adressage est dynamique.

4.6 Synthèse

La supervision des réseaux et des services regroupe l'ensemble des activités qui consistent à mettre en œuvre, maintenir et mettre à jour les infrastructures de réseaux ainsi que les services qu'ils offrent aux usagers. On trouve actuellement différentes approches de gestion qui partent d'une centralisation totale de cette fonction à sa distribution parmi les éléments administrés.

Le modèle P2P, comme toute architecture de déploiement de service, requiert une infrastructure de supervision. En effet, s'il présente des caractéristiques d'auto-organisation, celles-ci ne sont pas suffisantes pour pouvoir offrir aux usagers la garantie de niveaux de services. On trouve actuellement trois grandes approches de supervision du modèle P2P. La première repose sur l'utilisation de modèles économiques issus du domaine des finances et propose de voir une

communauté P2P comme une économie de marché où chaque ressource à un coût et chaque pair cherche à maximiser son profit. Des mécanismes incitatifs permettent d'assurer ce profit tout en garantissant le bon fonctionnement de la communauté. La seconde approche est fondée sur des mécanismes de réputation. Son utilisation conduit à exclure naturellement les pairs dont le comportement n'est pas satisfaisant en renforçant les interactions entre ceux dont la réputation est satisfaisante. La dernière s'intéresse à la supervision de la topologie virtuelle construite par les pairs.

L'utilisation du modèle P2P pour la gestion des réseaux et service a été envisagée et mise en œuvre dans cadre du travail de thèse de Konstantinou. Ce travail s'inscrit dans le cadre de l'auto-gestion et présente des travaux de validation qui montrent sa faisabilité.

Les différentes approches de gestion étudiées dans ce chapitre, montrent que les choix de conception d'une infrastructure de supervision sont fortement liés au service supervisé. C'est pourquoi, nous avons proposé une classification des applications P2P orientée vers la gestion, qui repose sur cinq critères qui sont : l'ouverture vers la gestion, le niveau de décentralisation, l'utilisation d'un réseau virtuel, l'échelle de déploiement et la dynamique des participants et de leur identification.

Deuxième partie

Contributions

Chapitre 5

Modélisation de l'information de gestion

Sommaire

5.1 Objectif

L'objectif du travail présenté dans ce manuscrit est la conception et le développement d'une infrastructure de supervision pour les applications construites selon le modèle P2P. La première étape naturelle de ce travail concerne la modélisation des informations de gestion. Avant de mettre en œuvre une architecture de supervision, un protocole de communication et une application de gestion, il faut tout d'abord définir les informations à considérer dans ce cadre : (1) *Quelles sont les caractéristiques du modèle P2P à intégrer dans le cadre de la supervision ?* (2) *Quelles sont les spécificités introduites par ce modèle par rapport au modèle client-serveur ?* et (3) *Comment représenter ces informations ?*

Dans ce chapitre, nous présentons la manière dont nous avons répondu à ces trois questions. Notre premier travail a consisté à recenser les caractéristiques propres au modèle P2P qui nécessitent la définition d'un nouveau modèle de l'information. Ensuite, nous nous sommes attaché à formaliser ces informations à l'aide de l'approche standard de gestion CIM[1]. Parmi les différents modèles de l'information existants, nous avons choisi ce formalisme pour plusieurs raisons : tout

[1]The Common Information Model

d'abord, CIM est une approche orientée objet et est, par essence, facilement extensible. Ensuite, elle définit un grand nombre de classes qui sont réutilisables, facilitant ainsi notre travail de formalisation en fournissant un cadre général fixe. Enfin, CIM est un standard de la gestion qui est largement reconnu et déployé dans les infrastructures actuelles, ce qui rend notre travail plus facile à intégrer.

La suite de ce chapitre est organisée en deux parties. Dans la première, nous présentons CIM, les différents concepts qu'il propose et quelques éléments du modèle existant. Dans la seconde, nous détaillons chacune des caractéristiques du modèle P2P que nous avons considérées dans notre démarche. Elles concernent les aspects organisationnels, fonctionnels, topologiques, et de communication du modèle P2P. Nous détaillons aussi les propositions de diagrammes de classes que nous avons élaborés.

5.2 Le modèle commun de l'information

Le modèle commun de l'information (CIM) [18] est une approche proposée par le DMTF[2] pour la gestion des stations de travail, des réseaux et des services. CIM est en fait une double proposition qui définit un formalisme d'expression de l'information de gestion ainsi qu'un ensemble d'éléments réutilisables directement dans le cadre d'une infrastructure de supervision [57]. Le formalisme d'expression repose sur le paradigme orienté objet et définit une manière graphique de représenter les éléments modélisés sous une forme assez similaire à l'approche UML[3]. Conjointement à cela, CIM définit le langage MOF[4] comme langage d'expression formel pour les objets gérés.

5.2.1 Les concepts de CIM

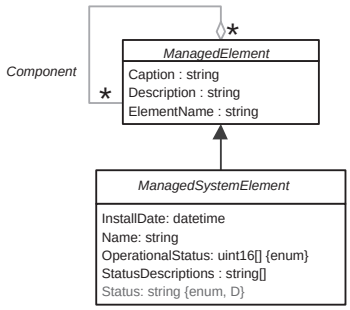
Un des objectifs de CIM est de permettre la spécification et la mise à disposition des applications d'une description unifiée et extensible des ressources gérées [57]. Pour cela, CIM comporte cinq éléments principaux qui sont :

1. Un méta-modèle [39] qui décrit les concepts du modèle qui servent de briques de base à la construction des spécifications. De par sa nature orientée objet, CIM définit notamment les notions de classe, attribut, méthode, association, référence, qualifieur et schéma, dont nous détaillons le sens ci-après ;
2. Un nommage des composants gérés (classes, associations, qualifieurs et instances) qui repose sur deux principes : un schéma et un espace de nommage ;
3. Un langage support pour la spécification des objets gérés. Il définit la syntaxe d'expression des éléments du méta-modèle ;
4. Un modèle de l'information de référence servant de base à la spécification de nouveaux composants gérés, ainsi qu'un sous-ensemble représentant les objets que tout serveur doit implanter ;
5. Un ensemble de recommandations sur l'intégration des modèles de l'information issus d'autres approches. En effet, CIM se présente comme un modèle unificateur capable de prendre en considération les différents domaines et approches de gestion de réseaux. Dans ce cadre, plusieurs méthodes de mise en correspondance et de traduction sont proposées.

[2]Distributed Management Task Force - http ://www.dmtf.org
[3]Unified Modeling Language
[4]Managed Object Format

5.2.2 Illustration de quelques concepts

La figure 5.1 illustre la représentation graphique des classes CIM ainsi que leur traduction dans le langage MOF. Plusieurs points sont remarquables : tout d'abord, une classe CIM possède un nom composé de deux parties. La première identifie le schéma auquel la classe appartient, et la seconde, la classe à proprement parler. Par exemple, dans la figure 5.1.b, la classe CIM_ManagedElement appartient au schéma CIM et possède le nom ManagedElement[5]. Ensuite, une classe est composée d'attributs et de méthodes. Les attributs disposent de types standard (entier, flottant, chaîne de caractères, ...). Les méthodes quant à elles, sont prototypées de manière classique par leur nom, des paramètres d'entrée et/ou de sortie et un paramètre de retour. Dans notre exemple, par souci de clarté, nous avons seulement représenté des classes ne possédant pas de méthode. Par contre, concernant les attributs, on voit par exemple que l'attribut Caption de la classe CIM_ManagedElement est une chaîne de caractères (type string). Le concept suivant que nous introduisons est celui de qualifieur. Un qualifieur est un mot clé qui s'applique à une classe, un attribut ou une méthode, et qui permet d'ajouter une sémantique supplémentaire ou de modifier la sémantique originelle de l'élément auquel il s'applique. On considère à nouveau l'attribut Caption qui possède deux qualifieurs, notés entre crochets. Le premier indique que sa chaîne de caractères ne pourra contenir plus de 64 éléments (MaxLen (64)), le second apporte une description exhaustive de l'attribut (Description).



(a)

```
[Abstract, Version ("2.7.0")] class CIM_ManagedElement {
    [MaxLen (64), Description ("...")] string Caption;
    string Description;
    string ElementName;
};

[Association, Abstract, Aggregation] class CIM_Component {
    [Aggregate, Key, Description ("The parent element (...).") ]
    CIM_ManagedElement REF GroupComponent;
    [Key, Description ("The child element (...).") ]
    CIM_ManagedElement REF PartComponent;
};

[Abstract] class CIM_ManagedSystemElement : CIM_ManagedElement {
    // Attributs de la classe CIM_ManagedSystemElement
};
```

(b)

Tab. 5.1 – Exemple de classes CIM et leur traduction dans le langage MOF. (a) Spécification sous forme de diagramme de classes. (b) Traduction des classes dans le langage MOF.

Nous nous intéressons maintenant à la définition des classes. Comme tout formalisme orienté objet, CIM définit la notion d'héritage, qui dans son cas est un héritage simple. Il est représenté par une flèche dans la figure 5.1.a et par la syntaxe `<nom_classe_fille> : <nom_classe_mère>` dans le langage MOF. Ensuite, CIM distingue plusieurs types de classes : les classes concrètes, les classes abstraites et les classes d'association. Les classes concrètes représentent un élément physique ou logique entrant dans le cadre de l'infrastructure de gestion. Ces classes peuvent être instanciées à travers un objet géré. Ensuite, les classes abstraites, spécifiées à l'aide du qualifieur Abstract représentent elles aussi un élément physique ou logique, mais de manière générique et abstraite. Elle ne peuvent ainsi pas être instanciées directement. Enfin, les classes d'association, mentionnées à l'aide du qualifieur Association, représentent un lien particulier

[5]Par convention, on ne représente pas les noms de schémas dans les diagrammes de classes CIM

entre deux classes concrètes ou abstraites. La spécification de cardinalités affine la sémantique de l'association. Dans notre exemple, la classe CIM_Component, de cardinalité * - * associe la classe CIM_ManagedElement à elle-même et signifie qu'un élément géré peut être éventuellement composé de sous-éléments. De même, un élément géré peut éventuellement appartenir des sur-éléments. Enfin, pour accéder à une instance de classe particulière, CIM définit un schéma de nommage qui contient, en outre, les attributs qualifiés de clé (key).

5.2.3 Le modèle commun de l'information

La proposition d'un formalisme commun pour la spécification des ressources gérées est un premier pas vers l'intégration et l'interopérabilité des différentes approches de gestion de réseaux [57]. Cependant, il est aussi nécessaire de représenter de manière uniforme une même ressource. CIM permet cette uniformisation en proposant un ensemble de schémas qui définissent les classes d'objets gérés inhérentes à différents domaines de la gestion. Cet ensemble se scinde en deux niveaux :

Le modèle *Core* [40] définit un ensemble de classes et d'association génériques à tout domaine de gestion. Il définit une structure de base pour tous les schémas futurs qui concernent des domaines particuliers. Ainsi, le modèle *Core* est stable et comporte des classes, au concept abstrait, dont le nombre est assez limité.

Le modèle *Common* définit des classes et des associations indépendantes de toute implantation, relatives à un domaine particulier de la gestion. Il se compose ainsi de différents sous-modèles qui concernent les applications [47], les systèmes [44], les réseaux [42], les politiques de gestions [43, 116, 115] ou la sécurité [45].

Conjointement à ces deux modèles, des schémas extension permettent d'étendre les classes existantes. Ces derniers sont très utiles pour spécialiser un schéma existant afin de prendre en compte les spécificités d'éléments particuliers ou pour couvrir un nouveau domaine qui ne soit pas pris en compte dans le modèle *Common*.

5.2.4 Eléments du modèle *Core*

Le modèle *Core* définit un ensemble de classes communes à l'ensemble des domaines de gestion. La majorité des classes concrètes qu'il définit héritent de classes abstraites, génériques, représentant la racine du modèle. Comme le montre la figure 5.1, la classe abstraite CIM_ManagedElement représente de la manière la plus générale, un élément géré dans le cadre d'une approche de gestion. Celui-ci est défini par trois attributs qui renseignent sur son nom (ElementName) et deux descriptions, l'une courte (Caption) et l'autre longue (Description). Ensuite, la classe abstraite CIM_ManagedSystemElement représente un élément quelconque distinguable, défini dans le cadre d'un système particulier, comme par exemple, un fichier ou un processeur. Les attributs ajoutés par cette classe renseignent sur le nom de l'élément[6] (Name), un statut opérationnel (OperationnalStatus qui rend obsolète l'attribut Status), une description textuelle de ce statut (StatusDescription) et la date d'installation de l'élément (InstallDate).

A ce niveau d'héritage, CIM effectue une séparation entre les éléments physiques et les éléments logiques. Les premiers, modélisés par le biais de la classe CIM_PhysicalElement, représentent des éléments réels, soumis à la loi de la pesanteur. Ils peuvent être, par exemple, une

[6] Les attributs Name de la classe CIM_ManagedSystemElement et ElementName de la classe CIM_ManagedElement peuvent sembler concurrents, mais souvent, l'attribut Name est surchargé pour devenir une clé, alors que ElementName représente un nom intelligible.

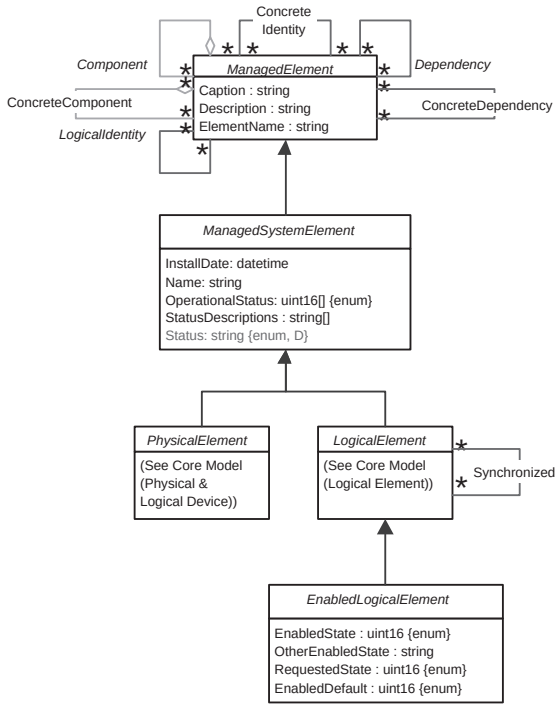


FIG. 5.1 – Les classes abstraites du modèle *Core* qui sont les racines du modèle.

carte réseau, un disque dur ou une alimentation. Par contre, les éléments logiques n'ont pas d'existence concrète et ne peuvent ainsi pas être vus ou touchés. C'est le cas par exemple, d'un système d'exploitation, d'une partition mémoire ou de la pile de protocoles d'une carte réseau.

La dernière classe qui nous intéresse est CIM_EnabledLogicalElement. Cette classe représente un élément logique qui peut prendre différents états. Parmi les quatre propriétés qu'elle définit, l'attribut EnabledState renseigne sur l'état de l'élément considéré qui peut être, en outre, actif, inactif, en cours de démarrage ou d'arrêt.

5.3 Proposition d'un modèle de l'information de gestion

Le modèle de l'information pour les réseaux et services P2P que nous avons conçu, est un schéma d'extension de CIM [173]. Il étend et utilise des classes définies dans le modèle *Core* et dans différents sous-modèles du modèle *Common*.

Un des objectifs que nous nous sommes fixés durant la conception de notre modèle concerne la généricité. Il nous semble important que le modèle de l'information que nous proposons puisse être appliqué à n'importe quelle approche P2P. Par ce biais, une application de gestion peut disposer d'une vue uniforme et cohérente de différentes infrastructures P2P dont elle assure la supervision. Notre travail a donc consisté à inventorier les concepts communs à toutes les approches P2P, qui font sens dans le cadre de la gestion de réseaux et de services. A ce jour, nous en avons recensé cinq qui sont :

L'organisation : Nous avons pris en compte les trois critères d'organisation des pairs présentés dans la section 2.2.2 et qui sont (1) l'organisation fonctionnelle, qui attribue à chaque pair un rôle différent, (2) l'organisation communautaire qui scinde les ensembles de pairs en

groupements d'intérêts communs, (3) l'organisation topologique qui organise le réseau virtuel construit par les pairs.

La communication : Les infrastructures P2P actuelles proposent aux pairs différentes manières de communiquer. Certaines utilisent une simple adresse de niveau transport, d'autres créent une abstraction à travers l'usage de canaux de communication virtuels (*pipes*), et récemment, on a vu apparaître les premières propositions de communication *multicast* et *anycast*. Nous avons pris en compte la manière dont des pairs communiquent à travers une modélisation commune aux différentes méthodes de communication.

Les ressources : Dans un environnement distribué et dynamique tel qu'il est défini par le modèle P2P, il est très difficile d'estimer les ressources dont dispose une communauté. Sur le plan de service, les DHTs apportent un élément de réponse en proposant une infrastructure fiable pour accéder à une ressource donnée, mais elles ne permettent pas d'obtenir de connaissance globale des ressources. Sur le plan de gestion, il est important de pouvoir obtenir une vue des ressources disponibles et de celles qui sont utilisées, mais aussi de pouvoir les localiser et identifier leur propriétaire. Dans ce cadre, nous avons conçu un modèle qui prend en compte ces aspects et fournit une vue générique d'une ressource quelle que soit sa nature (fichier, bande passante, cycles CPU[7], ...).

Les services : La finalité du modèle P2P est de fournir, de manière distribuée, un service particulier. La différence entre un service P2P et un service centralisé tel qu'il est déployé dans le cadre du modèle client/serveur, réside dans les deux niveaux d'abstraction que l'on peut considérer dans sa représentation : un service P2P peut être vu comme une fonctionnalité logicielle exécutée par un pair particulier, ou comme une fonction, commune à un ensemble de pairs exécutant une instance particulière de ce service. Notre modèle de l'information considère cette dualité et propose en outre de localiser les services par rapport aux pairs, et rejoint par ce biais le modèle d'organisation fonctionnelle.

Le routage : Le modèle P2P peut être vu comme la construction d'un réseau virtuel au-dessus d'un réseau physique. Il redéfinit donc un service de routage de calcul de routes qui est étroitement lié à l'approche topologique considérée. L'aspect dynamique du modèle P2P rend la fonction de routage particulièrement sensible et, d'un point de vue la gestion, il est important de pouvoir obtenir une vue des services de routage et de calcul de routes ainsi que des tables de routage. Nous avons ainsi étendu les classes CIM qui sont inhérentes à ce domaine pour prendre en compte les spécificités du modèle P2P.

Ces cinq concepts ont conduit à la conception de cinq sous-modèles CIM, définis dans le cadre du schéma d'extension P2P, dont certains sont de simples extensions de modèles CIM existants. La figure 5.2 illustre la manière dont ces cinq sous-modèles sont en relation. Le modèle[8] organisationnel pour les pairs et les communautés est l'élément central de notre schéma. Les sous-modèles de ressources, communication et services lui sont rattachés. Ensuite, les fonctions de routage et calcul de routes sont des services particuliers ; c'est pourquoi, ils sont représentés par un sous-modèle qui est une spécialisation du modèle de services P2P. On peut noter que le préfixe de nommage de notre schéma d'extension est P2P. Par conséquent, toutes les classes que nous avons conçues portent ainsi le nom P2P_nom-de-classe. Dans la suite de ce chapitre, nous décrivons en détail trois des cinq sous-modèles introduits dans cette section, à savoir le modèle de l'organisation, le modèle de services et le modèle de la communication. Nous présentons les deux autres sous-modèles en annexe A et la spécification MOF des cinq modèles en annexe B.

[7] *Central Processing Unit* : une unité centrale de traitement

[8] Dans la suite, par abus de langage, on utilise indifféremment le terme de modèle et sous-modèle

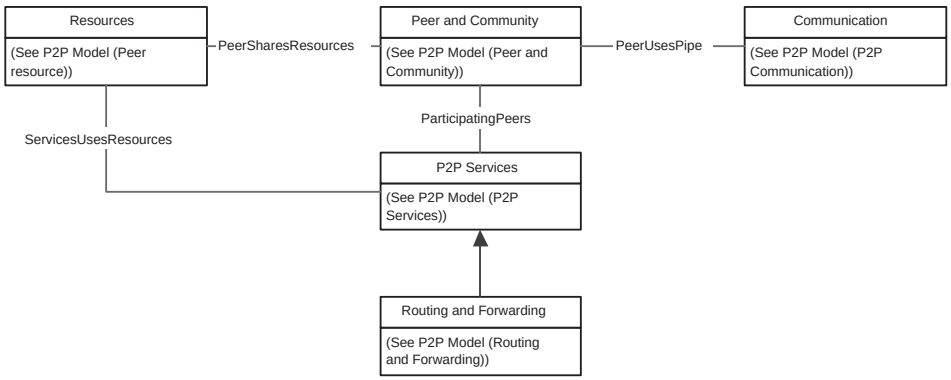


FIG. 5.2 – Vue générale des différents sous-modèles qui constituent le schéma d'extension de CIM pour les modèles et services P2P

5.3.1 Modèle de l'organisation des pairs et des communautés

Dans le domaine P2P, un pair représente un élément, exécuté sur une machine particulière, d'une application distribuée construite selon une approche décentralisée. Une telle application peut couvrir des domaines très variés, du calcul distribué à la diffusion de contenu multimédia. Le service, tel qu'il est offert et consommé, provient de l'agrégation des pairs et de la mise en commun de leur ressources dans ce contexte particulier. Afin de représenter le regroupement de pairs autour d'un intérêt commun, on utilise le terme de communauté [119]. Une communauté est un ensemble de pairs participant à un ou des services communs.

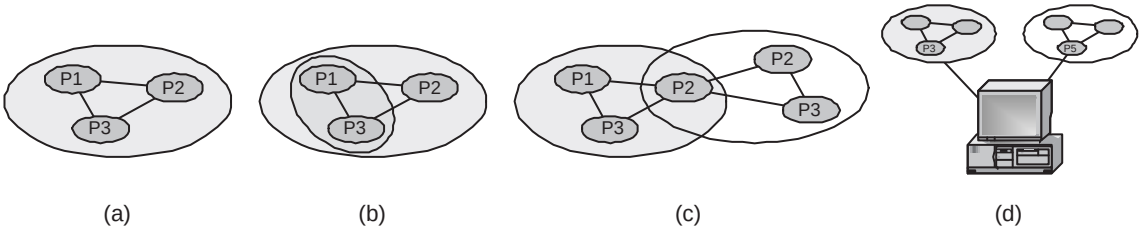


FIG. 5.3 – Exemples de cas représentables avec le modèle organisationnel. (a) Une communauté contient plusieurs pairs. (b) Une communauté contient une sous-communauté. (c) Un pair appartient à plusieurs communautés. (d) Un ordinateur exécute plusieurs applications P2P.

La figure 5.3 représente différents cas de relations qui peuvent apparaître entre des pairs et des communautés. Tout d'abord, la figure 5.3.a illustre le fait qu'une communauté contienne un ensemble de pairs, allant de un à plusieurs. Nous considérons qu'une communauté vide ne peut exister, car le concept de communauté est simplement l'abstraction du regroupement de pairs dans le cadre d'un service. Ainsi, si aucun pair n'existe, aucun groupement non plus, et donc aucune communauté. Ensuite, la figure 5.3.b montre qu'une communauté peut contenir des sous-communautés, et qu'un pair peut appartenir simultanément à une communauté ainsi qu'à ses sous-ensembles. C'est par exemple le cas d'une application de discussion où un pair appartient à la communauté des utilisateurs du client d'une communication et à différentes sous-communautés qui représentent les sujets de discussion auxquels il s'est abonné. La figure 5.3.c montre qu'un pair peut aussi appartenir simultanément à des communautés différentes et

disjointes. Ce cas s'illustre par l'exemple d'une plate-forme de service comme OSGI[9] ou Jxta [119]. Celle-ci peut accueillir et exécuter différents services P2P au sein de son infrastructure, et ainsi représenter un pair appartenant à plusieurs communautés. Enfin, la figure 5.3.d considère le cas où, au sein d'une machine particulière, plusieurs applications P2P sont exécutées, conduisant ainsi à la représentation de différents pairs appartenant à différentes communautés.

Diagramme de classes CIM du modèle de l'organisation

Le modèle de l'organisation des pairs et des communautés que nous proposons est représenté sur la figure 5.4. Ce modèle est la racine de l'ensemble des autres modèles que nous avons intégrés dans notre schéma d'extension.

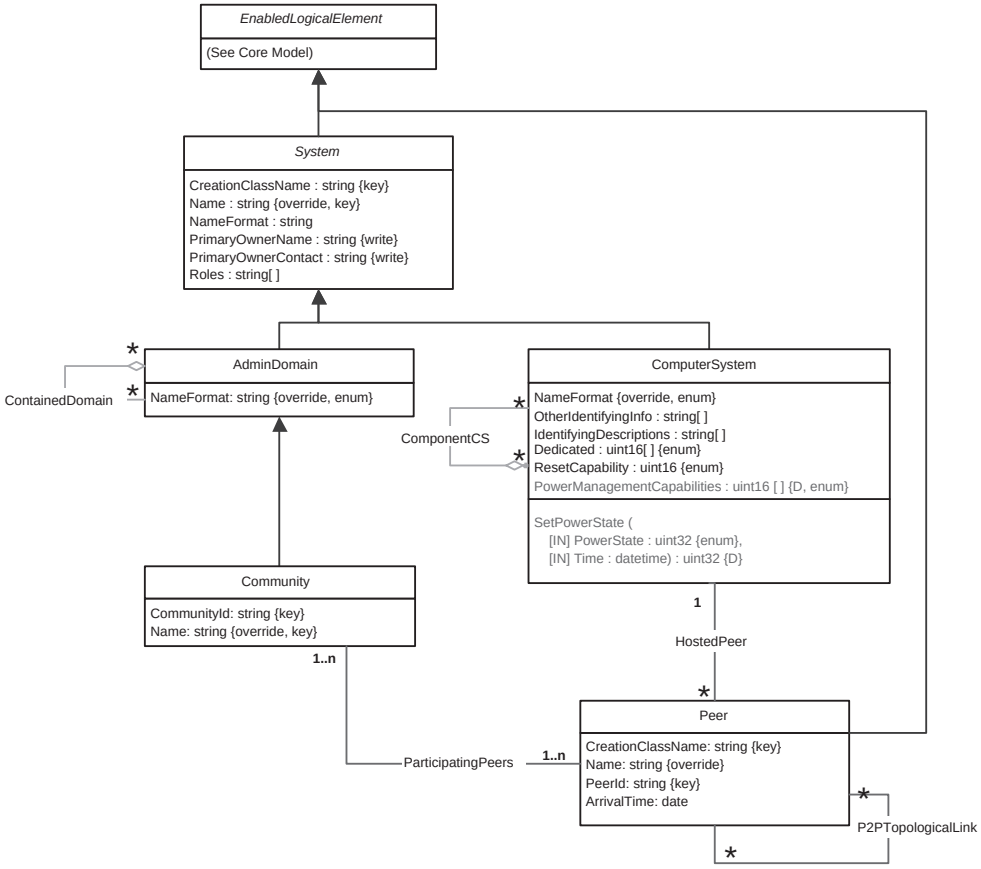


FIG. 5.4 – Diagramme de classes CIM du sous-modèle P2P de l'organisation fonctionnelle et topologique

Un pair est représenté à l'aide de la classe P2P_Peer. Il se caractérise par son nom (Name)[10], son identifiant (PeerId) et sa date d'arrivée (ArrivalTime). L'attribut CreationClassName permet, lors de l'instanciation d'un objet, de renseigner sur le nom de la classe effective de création d'un objet de type P2P_Peer. Il est notamment utile pour distinguer plusieurs objets d'attributs identiques mais de classes, héritant de la classe P2P_Peer, différentes. En outre, un pair est un

[9] Open Service Gateway Initiative - http ://www.osgi.org

[10] par le biais de l'héritage, cet attribut surcharge l'attribut Name de la classe ManagedSystemElement

élément logiciel qui peut prendre différents états. C'est pourquoi, la classe P2P_Peer hérite de la classe CIM_EnabledLogicalElement du modèle *Core*, introduite dans la section 5.2.4.

Une communauté de pairs est représentée à l'aide de la classe P2P_Community. Cette classe présente deux attributs qui sont son nom (Name) et son identifiant (CommunityId). La classe P2P_Community hérite de la classe CIM_AdminDomain, définie dans le sous-modèle de réseaux [42] du modèle *Common*, qui représente un ensemble quelconque d'éléments appartenant à un même domaine d'administration. Afin de couvrir l'ensemble des cas de figure décrits dans la figure 5.3 de la section précédente, nous avons choisi de lier la classe P2P_Peer à la classe P2P_Community par le biais de la classe d'association P2P_PeerParticipating. Sa cardinalité de type 1..n - 1..n indique qu'un pair peut appartenir à une ou plusieurs communautés, et de même, une communauté contient au moins un pair. La notion de sous-domaine est prise en compte par la classe d'association CIM_ContainedDomain qui relie la classe CIM_AdminDomain à elle-même. Sa cardinalité signifie qu'un domaine peut contenir zéro ou plusieurs sous-domaines, de même qu'il peut être contenu dans plusieurs sur-domaines. Ce dernier cas ne semble pas envisageable dans le cadre d'une communauté P2P, mais, par souci de souplesse et de généricité, nous avons préféré conserver la classe CIM_ContainedDomain plutôt que de la sous-classer pour en restreindre la cardinalité.

Pour finir la description de ce modèle, nous présentons la manière dont nous associons un pair à la machine qui l'héberge. Le sous-modèle de systèmes [44] du modèle *Common* propose la classe CIM_ComputerSystem qui représente un ordinateur, et nous avons choisi de la lier à la classe P2P_Peer de notre modèle par le biais de la classe d'association P2P_HostedPeer. Sa cardinalité indique qu'un pair appartient à une machine, mais qu'une machine peut héberger zéro ou plusieurs pairs. On peut noter que nous avons choisi de lier la classe P2P_Peer à la classe CIM_ComputerSystem plutôt que la classe CIM_System, car la classe CIM_ComputerSystem représente un ordinateur, de la station de travail au *cluster* de machines, alors que la classe CIM_System représente simplement l'agrégation d'éléments gérés, qui peut être dans ce cadre un domaine administratif ou une librairie de stockage.

Modélisation de la topologie virtuelle

Les applications P2P sont souvent construites sur un *overlay* de niveau applicatif. La topologie virtuelle que ce réseau engendre peut reposer sur différents critères relationnels. Elle peut être par exemple construite à l'aide des tables de routage des pairs. On considère alors qu'un pair P_1 référencé dans la table de routage d'un pair P_2, est un voisin de ce dernier. Ce concept est celui retenu pour expliquer la construction topologique des DHTs. Cependant, la topologie peut aussi être construite selon un critère de communication : deux pairs qui échangent des données sont considérés comme voisins dans la topologie virtuelle. Enfin, la topologie peut aussi reposer sur la connaissance globale que des pairs ont des autres indépendamment des tables de routage. C'est, par exemple, le cas de Gnutella [84, 133] ou Freenet [26] où le fait de faire suivre un message dans lequel est indiquée la source du message va permettre aux pairs situés sur le chemin d'ajouter le pair source dans leur table de références.

Dans notre modèle, nous avons choisi de représenter les liens topologiques entre les pairs, sans définir de sémantique particulière à cette relation. La figure 5.4 montre que nous avons défini la classe P2P_P2PTopologicalLink qui lie la classe P2P_Peer à elle-même. Sa cardinalité * - * indique que les pairs peuvent posséder zéro ou plusieurs liens topologiques entre eux. Pour remplir notre contrainte de généricité de la relation, nous avons défini les attributs représentés sur la figure 5.5 : Tout d'abord l'aspect asymétrique de la relation topologique est pris en compte par la définition d'un attribut Antecedent et Dependent. L'antécédent de la relation possède la

relation topologique alors que le dépendant est référencé par la relation. Par exemple, dans le cas d'une relation topologique fondée sur la connaissance, l'antécédent est celui qui connaît, et le dépendant est celui qui est connu. Ensuite, la symétrie d'une relation est représentée par l'attribut IsSymetric. Enfin, l'attribut Description permet de décrire la sémantique de la relation.

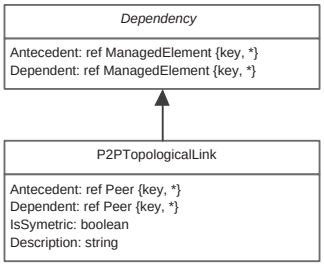


FIG. 5.5 – La classe d'association de lien topologique

5.3.2 Modèle de services P2P

La finalité du modèle P2P est de fournir à des usagers un service d'une manière distribuée. La modélisation des informations de gestion des services P2P est donc un point crucial d'une infrastructure de supervision d'applications P2P. D'après le modèle *Core* de CIM, un service est *une fonctionnalité fournie par un élément logiciel ou matériel*. D'après cette définition, un service peut être, par exemple un logiciel de traitement de texte exécuté sur un ordinateur, une application cliente de courrier électronique ou un service d'impression. Certains services sont locaux à la machine qui les utilise alors que d'autres sont distants et accessibles par le réseau. Pour représenter cette caractéristique, CIM utilise la notion de point d'accès qui représente la manière d'accéder à un service. Par exemple, dans le cas d'une application de messagerie électronique, le point d'accès du service est l'adresse protocolaire utilisée pour retirer les courriers du serveur.

La figure 5.6.a représente les classes CIM du modèle de services tel qu'il est défini dans le modèle *Core*. La classe CIM_Service représente un service défini à travers des fonctionnalités logiques. C'est pourquoi elle hérite de la classe CIM_EnabledLogicalELement. Un service se caractérise principalement par son nom (Name), les coordonnées d'une personne physique ou morale responsable de ce service (PrimaryOwnerName et PrimaryOwnerContact), et surtout deux méthodes qui permettent, par le biais de la gestion, de démarrer (startService()) ou arrêter un service (stopService()) ; l'état résultant étant mentionné dans l'attribut Started. Deux classes d'auto-association permettent de représenter la composition de service (CIM_ServiceComponent) et la dépendance d'un service par rapport à un autre (CIM_ServiceServiceDependency). Les différentes associations entre la classe CIM_Service et CIM_ManagedElement représentent la manière dont l'exécution d'un service dépend et se répercute sur un ou des éléments administrés.

Ensuite, la classe abstraite CIM_ServiceAccessPoint modélise un point d'accès particulier pour un service donné, représenté particulièrement par son nom (Name). Cette classe est générique, et des points d'accès concrets sont décrits dans ses sous-classes, comme c'est le cas des classes CIM_ProtocolEndpoint, CIM_ServiceAccessURI ou CIM_RemoteServiceAccessPoint. Enfin, la classe d'association CIM_ServiceAccessBySAP met en relation un service et ses points d'accès, et la classe CIM_ServiceSAPDependency représente la dépendance entre un service et ses points d'accès. Nous ne décrivons pas plus en détail le modèle de services de CIM. Des informations complémentaires

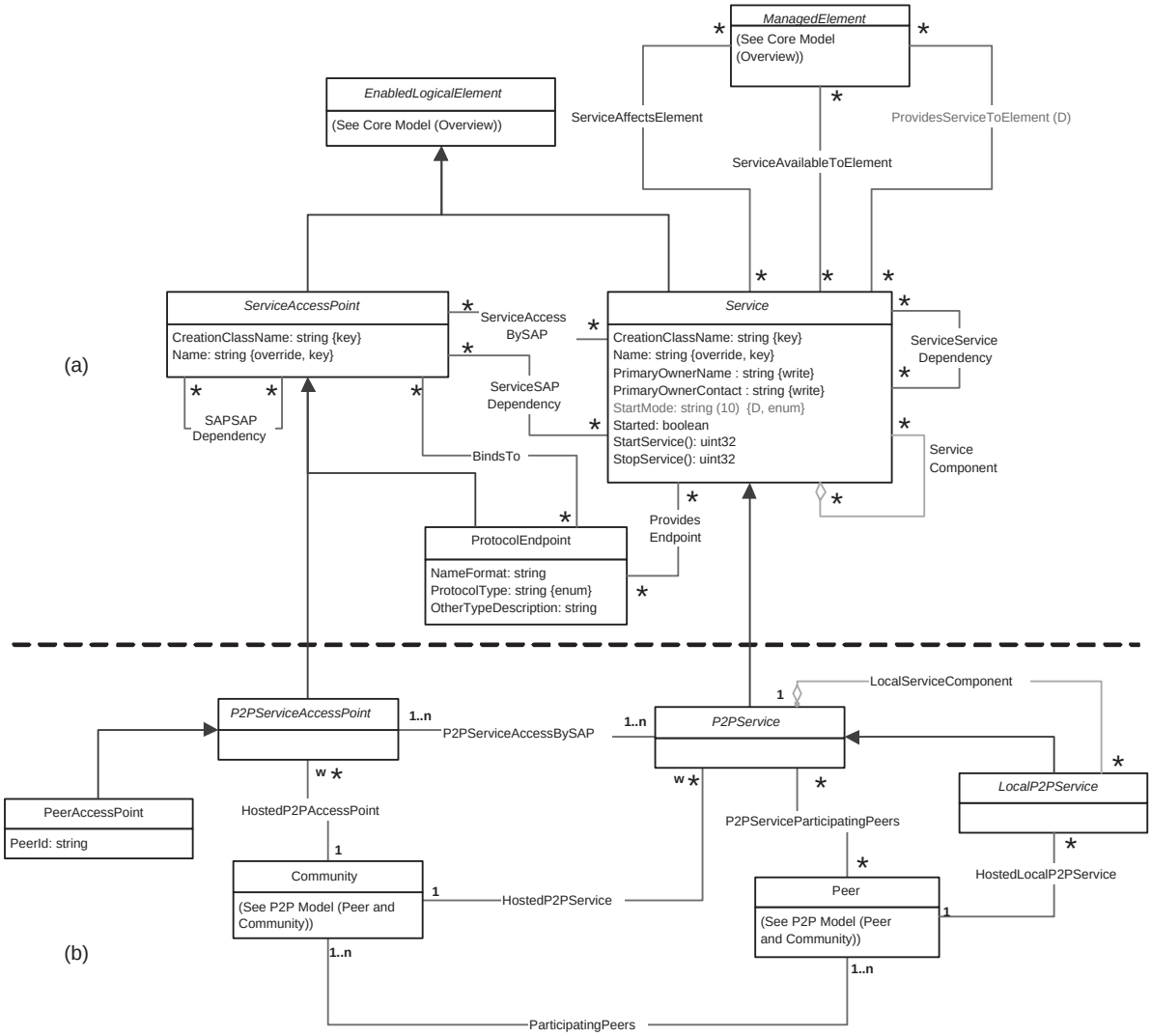


FIG. 5.6 – Diagramme de classes CIM du sous-modèle P2P de services. (a) Classes issues du modèle *Core*. (b) Classes du schéma d'extension P2P.

sont disponibles dans [40, 18].

Diagramme de classes CIM du modèle de services P2P

La définition de service proposée par CIM le décrit comme étant centralisé. Dans le cadre d'un modèle P2P, nous avons étendu cette définition pour considérer l'aspect distribué du modèle et proposons la suivante : *Un service P2P est une fonctionnalité logicielle distribuée fournie et ou consommée par un ensemble de pairs appartenant à une même communauté.*

La définition d'un point d'accès à un service P2P est identique à celle proposée dans le modèle *Core*. Plus précisément, dans le cadre du modèle P2P, un point d'accès représente l'adresse d'un pair qui permet de joindre une communauté. Par exemple, dans le cadre des DHTs, on parle souvent de pair amorce, qui permet à d'autres de s'insérer, et dans le cas d'applications de partage de fichiers ou de Jxta, c'est une liste de pairs connus, maintenue et éventuellement publiée sur le Web.

Etant donnés ces concepts, nous avons conçu le sous-modèle de services représenté sur la figure 5.6.b. Tout d'abord, nous avons défini la classe abstraite P2P_P2PService qui hérite de la classe CIM_Service et qui représente un service P2P dans sa globalité, tel un service travail collaboratif. Nous n'avons pas ajouté de propriété à cette classe car celles définies dans la classe CIM_Service nous ont semblées suffisantes et, l'objectif de cet héritage est, d'un point de vue de l'information de gestion, de différencier la sémantique d'un service CIM standard, d'un service P2P tout en restant générique. De plus, dans le cadre d'une instanciation sur une infrastructure concrète, des attributs spécifiques peuvent être ajoutées à des sous-classes de P2P_P2PService. Ensuite, avec la même démarche, nous avons ajouté la classe abstraite P2P_P2PServiceAccessPoint qui représente un point d'accès à un service P2P. Une de ses sous-classe concrète est P2P_PeerAccessPoint. Elle se caractérise par l'identifiant du pair (PeerId) qui permet l'accès à un service. On peut en outre envisager d'autres méthodes d'accès à un service, comme par exemple une adresse P2P *anycast*. Les classes P2P_P2PService et P2P_P2PServiceAccessPoint sont liées grâce à la classe d'association P2P_P2PServiceAccessBySAP qui associe un service P2P à son ou ses points d'accès ; un point d'accès pouvant être attaché à plusieurs services P2P.

Le second point qui nous a semblé intéressant est la manière dont un service est distribué dans une communauté. Pour cela, nous avons ajouté deux classes d'association à ce modèle. La première, P2P_HostedP2PService permet, en accord avec notre définition de service P2P, d'identifier la communauté à laquelle un service appartient. D'un point de vue des cardinalités, on voit qu'un service appartient à une et une seule communauté, alors qu'une communauté regroupe zéro ou plusieurs services. La seconde association, P2P_P2PServiceParticipatingPeers met en relation les services et les pairs qui y participent. Un service peut être exécuté par zéro et plusieurs pairs et, de même, un pair peut participer à zéro ou plusieurs services. Par ce biais, une infrastructure de gestion peut identifier les différents services d'une communauté ainsi que les pairs qui y participent.

Vue locale et vue globale d'un service P2P

Habituellement, un service est localisé sur une machine particulière. C'est par exemple le cas d'une application de traitement de texte qui réside sur un ordinateur personnel. Au pire, un service est situé sur un serveur distant, accessible par le réseau. Dans tous les cas, le service est une entité fonctionnelle fournie par un élément distinct. Par contre, dans le contexte P2P, un service est une fonction distribuée qui résulte de l'agrégation d'instances particulières exécutées

par des pairs. On voit donc qu'un service P2P peut être vu de manière globale à une communauté, mais aussi de manière locale à chaque pair. D'un point de vue de la gestion, il est important de distinguer ces deux niveaux d'abstraction car les informations qui leurs sont inhérentes sont différentes. Par exemple, considérons le cas d'un service de découverte de fichiers construit sur une DHT. D'un point de vue global, les informations de gestion intéressantes vont porter sur le pourcentage de requêtes accomplies avec succès, ou le temps moyen nécessaire au routage d'un message, le nombre moyen de sauts, ... Par contre, d'un point de vue local, les informations vont porter sur la manière dont un pair particulier exécute le service, comme par exemple, le nombre total de requêtes qu'il a reçues et fait suivre, le nombre de requêtes qui lui ont été destinées, ... Le modèle P2P présente donc une vue à deux niveaux des services et nous avons pris en compte cette spécificité en intégrant la classe P2P_LocalP2PService à notre modèle de services. Cette classe est reliée à la classe P2P_P2PService par le biais de la classe d'agrégation P2P_LocalServiceComponent et représente un service tel qu'il est exécuté sur un pair particulier. A l'inverse, la classe P2P_P2PService représente un service vu dans sa globalité. Les cardinalités de l'association de ces deux classes stipulent qu'un service global est l'agrégation de zéro ou plusieurs services locaux alors qu'un service local n'appartient qu'à une seule vue globale. De cette manière, nous sommes capables de prendre en compte le caractère distribué des services P2P.

5.3.3 Modélisation de la communication

Les applications P2P de type *overlay* utilisent des protocoles de niveau applicatif pour assurer la communication entre les pairs, chaque pair utilisant une adresse de niveau transport pour communiquer. CIM modélise ce type de communication par le biais de canaux de communications virtuels, appelés *pipes*, dont plusieurs types sont représentables. Nous en présentons quelques exemples sur la figure 5.7. Tout d'abord, un *pipe* peut être unidirectionnel (5.7.a) ou bidirectionnel (5.7.b). En outre, CIM autorise deux types de composition qui sont la composition série (5.7.c) et la composition parallèle (5.7.d). Pour chaque composition, un *pipe* de haut niveau permet de représenter de manière abstraite la composition de *pipes* sous-jacents.

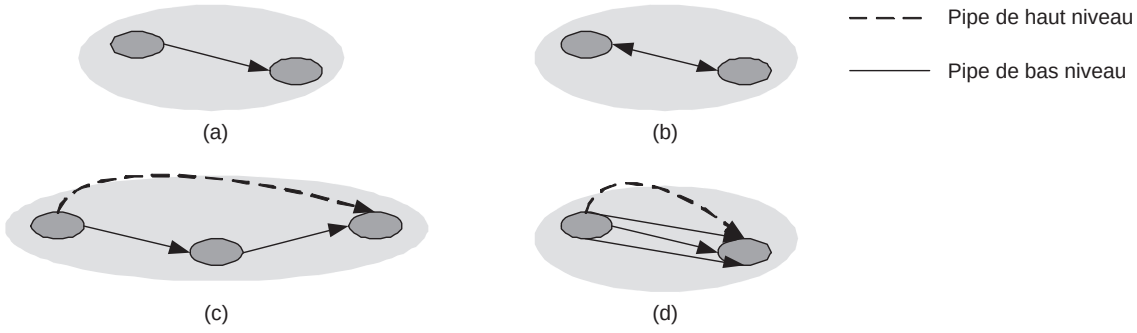


FIG. 5.7 – Exemples de *pipes* : (a) *Pipe* unidirectionnel. (b) *Pipe* bidirectionnel. (c) Composition de *pipes* en série. (d) Composition de *pipes* en parallèle.

Remarque : La notion de *pipe* telle qu'elle est définie dans le modèle *Common* et telle que nous la considérons dans notre modèle de l'information, est similaire à celle de Jxta, en ce sens qu'elle représente un canal de communication virtuel entre plusieurs pairs. Le seul point qui diverge concerne les différents types de *pipes*. Jxta considère des *pipes unicasts* qui correspondent au

cas illustré dans la figure 5.7.a et des *pipes* propagés qui ne sont pas directement représentables à l'aide du modèle *Common*. Ce point est abordé plus en détail dans la section 8.2.2.

Diagramme de classes CIM du modèle de communication

La sémantique de *pipe* proposée par CIM est identique à celle que nous proposons pour le modèle P2P. C'est pourquoi, nous avons choisi de réutiliser directement les classes du sous-modèle de réseaux [42] du modèle *Common*. La figure 5.8 représente l'extension du modèle CIM que nous proposons. Celle-ci est construite autour de la classe CIM_NetworkPipe, qui hérite de la classe CIM_EnabledLogicalElement du modèle *Core*. En accord avec la définition de *pipes* donnée dans la section précédente, cette classe se caractérise par sa directionnalité (Directionnality), son type d'agrégation (AggregationBehavior) et son identifiant (InstanceId). Les extrémités d'un *pipe* sont représentées à travers la classe CIM_ProtocolEndpoint, une sous-classe de la classe CIM_ServiceAccessPoint présentée dans la section 5.3.2. Le lien entre un *pipe* et ses points d'accès est matérialisé par la classe d'association CIM_EndpointOfNetworkPipe, qui stipule qu'un *pipe* est strictement défini par deux extrémités, mais qu'une extrémité peut appartenir à zéro ou plusieurs *pipes*. Enfin, la composition de *pipes* est prise en compte par la classe de composition CIM_NetworkPipeComposition qui relie la classe CIM_NetworkPipe à elle-même.

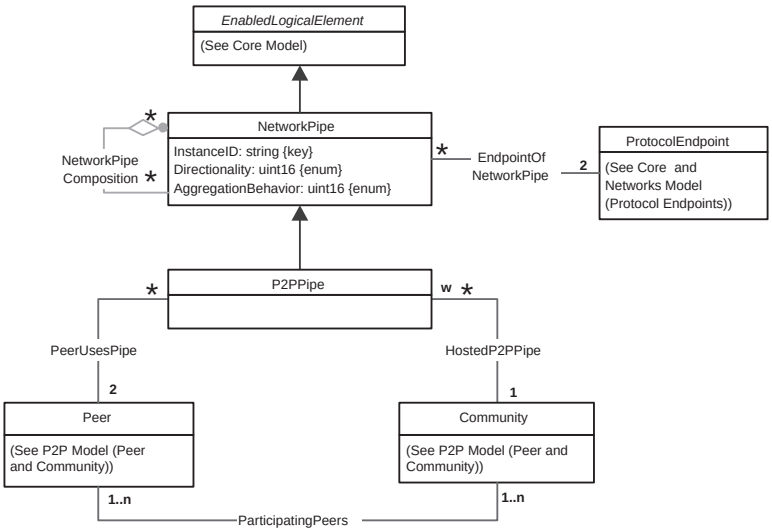


FIG. 5.8 – Diagramme de classes CIM du modèle de communication

Etant donné ce modèle, nous avons simplement étendu la classe CIM_NetworkPipe en y ajoutant la sous-classe P2P_P2PPipe. Cette classe n'apporte aucun nouvel attribut, et possède la même sémantique que sa classe ancêtre, mais elle permet d'ajouter deux liens d'association qui n'influent pas sur le modèle CIM. Le premier, représenté par la classe d'association P2P_PeerUsesPipe, permet de relier un *pipe* P2P aux deux pairs qui l'utilisent. Comme dans le cas de la classe d'association CIM_EndpointOfNetworkPipe, la cardinalité stipule qu'un *pipe* P2P est lié à deux pairs alors qu'un pair peut disposer de zéro ou plusieurs *pipes*. Le second lien que nous avons ajouté exprime la définition d'un *pipe* dans le contexte d'une communauté. Ceci est fait par le biais d'une association faible[11] du côté du *pipe*. Par ce biais, on identifiera toujours un *pipe* dans le cadre d'une communauté.

[11] Au sens du qualifieur weak de CIM

5.4 Synthèse

La première étape vers la construction d'une infrastructure de gestion pour les réseaux et services P2P concerne la modélisation des informations de gestion. Durant cette étape, nous avons recensé les différentes caractéristiques du modèle P2P qu'une infrastructure de gestion doit posséder. Ceci a conduit à la proposition de cinq modèles qui concernent l'organisation fonctionnelle et topologique des pairs, les ressources offertes par une communauté, la manière dont les pairs communiquent, les services qu'ils proposent, et enfin deux services particuliers en rapport avec le routage. Comme formalisme d'expression, nous avons choisi le Modèle Commun de l'Information, CIM. Sa nature orientée objet, son grand nombre de classes existantes et son caractère standard, le placent comme meilleur candidat à la représentation d'informations de gestion pour le modèle P2P. Ainsi, nous avons conçu un schéma d'extension de CIM qui caractérise les réseaux et services P2P du point de vue de la gestion et offre ainsi une vue abstraite à un gestionnaire.

Chapitre 6

Monitorage de la performance des tables de hachage distribuées

Sommaire

6.1 Besoin de monitorer la performance

Actuellement, les tables de hachage distribuées sont intégrées dans de nombreuses applications P2P. Elles sont en effet parfaitement adaptées pour assurer la localisation et l'accès à des ressources dans un environnement dynamique et distribué. Dans le chapitre 3, nous avons présenté quelques études comparatives de DHTs qui identifient leurs différences. D'autres études, relatives à l'évaluation de performance des infrastructures P2P et particulièrement les DHTs, ont été conduites. Un modèle théorique pour les réseaux P2P a été établi dans [85] et la proposition d'un modèle analytique qui permette l'évaluation de performance des applications P2P de partage de fichiers a été faite dans [86]. Conjointement à cela, des travaux de simulation ont été effectués pour comparer les différentes méthodes de recherche et de localisation P2P. Enfin, [130] propose une évaluation de type *benchmarking* pour les implantations des DHTs Chord et Tapestry dans l'environnement de test PlanetLab [97].

Ces différentes contributions, fondées sur des modèles analytiques ou des simulations, ne fournissent qu'une évaluation figée et restreinte à des cas précis, des différentes infrastructures de DHTs. Elles ne permettent pas de considérer l'ensemble des phénomènes qui peuvent apparaître dans le cas d'un déploiement effectif.

La contribution que nous exposons dans ce chapitre consiste à proposer un modèle de l'information, orienté vers la performance, qui permette à un gestionnaire d'estimer la performance d'une DHT dans des conditions de fonctionnement réel. Notre travail n'a pas pour objectif de comparer les différentes propositions de DHTs, mais plutôt de fournir une abstraction, générique à l'ensemble des propositions actuelles, qui puisse être utilisée dans le cadre de la gestion d'un service P2P pour assurer un niveau de performance donné.

Ce chapitre s'organise de la manière suivante. Tout d'abord, nous présentons le sous-modèle de métriques de CIM qui est le formalisme que nous avons choisi pour exprimer notre modèle de l'information. Ensuite, nous présentons notre proposition, qui consiste tout d'abord en une abstraction du fonctionnement d'une table de hachage distribuée, le choix de métriques adaptées, et leur intégration dans un modèle de l'information de gestion. Enfin, nous montrons comment nous avons enrichi ce modèle en l'appliquant à la DHT Chord [153]. Nous décrivons les différentes métriques inhérentes à Chord que nous avons conçues, et nous montrons de quelle manière nous pouvons les intégrer dans une instanciation particulière de notre modèle de l'information générique, décrit dans le chapitre 5.

6.2 Le sous-modèle de métriques de CIM

Le modèle *Common* de CIM propose un sous-modèle dédié aux applications [47]. Il permet de modéliser les applications exécutées sur des éléments administrés, leurs composants ainsi que leurs interactions avec les ressources logicielles et matérielles. Une partie de ce sous-modèle, appelée modèle de métriques [46], est dédiée à la caractérisation des performances des applications. Il est représenté sur la figure 6.1. Deux concepts y sont introduits. Le premier concerne les éléments ou les aspects d'une application dont on veut mesurer la performance. Pour représenter cela, CIM utilise la notion d'unité de travail[1]. Une unité de travail représente par exemple une tâche exécutée par une application, un script interprété ou une transaction entre un client et un serveur. La classe CIM_UnitOfWork modélise ce concept et représente une unité de travail en cours d'exécution. Elle se caractérise notamment par la date à laquelle l'unité de travail a démarré (StartTime), le temps écoulé depuis ce démarrage[2] (ElapsedTime) et son statut[3] (Status). Chaque unité de travail est rattachée à une définition particulière, représentée par la classe CIM_UnitOfWorkDefinition. Cette dernière permet de décrire une unité de travail sous forme textuelle et intelligible et de lui associer un niveau de traçabilité par le biais de son association à la classe CIM_TraceLevelType. Pour atteindre un niveau de granularité suffisant, la décomposition d'une unité de travail en sous-unités est possible grâce à la classe d'association CIM_SubUoW qui relie la classe CIM_UnitOfWork à elle-même.

Le second concept introduit par CIM dans le sous-modèle de mesure de performance d'applications concerne les métriques. Une métrique est une grandeur particulière qui caractérise un aspect d'une unité de travail. Par exemple, dans le cas d'une transaction entre un client et le serveur d'une base de données, une métrique possible est le nombre d'octets transmis par requête ; dans le cas de l'exécution d'un morceau de programme, une métrique peut être le temps nécessaire à son exécution ou la quantité de mémoire utilisée. De plus, comme pour les unités de travail, CIM attache chaque métrique à une définition particulière qui fournit des informations textuelles. La classe CIM_MetricDefinition représente cette définition. Concernant, les valeurs prises par une métrique, CIM propose deux classes pour les représenter. La première,

[1] en anglais *Unit of work*

[2] qui représente le temps total d'exécution si l'opération est terminée

[3] en cours d'exécution, terminée, annulée, ...

CIM_UoWMetric représente la valeur d'une métrique associée à une unité de travail en cours d'exécution. La seconde, CIM_BaseMetricValue représente les valeurs prises par les métriques attachées à des unités de travail précédentes qui sont maintenant achevées. Les valeurs d'une métrique sont liées à une définition par le biais de la classe d'association CIM_MetricInstance qui stipule qu'une définition de métrique peut présenter plusieurs valeurs issues de différentes mesures, et qu'une valeur dépend d'une et une seule définition.

Enfin, les définitions de métriques sont liées aux définitions d'unités de travail à travers la classe d'association CIM_UoWMetricDefinition qui indique qu'une unité de travail peut être caractérisée par zéro ou plusieurs métriques et qu'une métrique est relative à au moins une unité de travail.

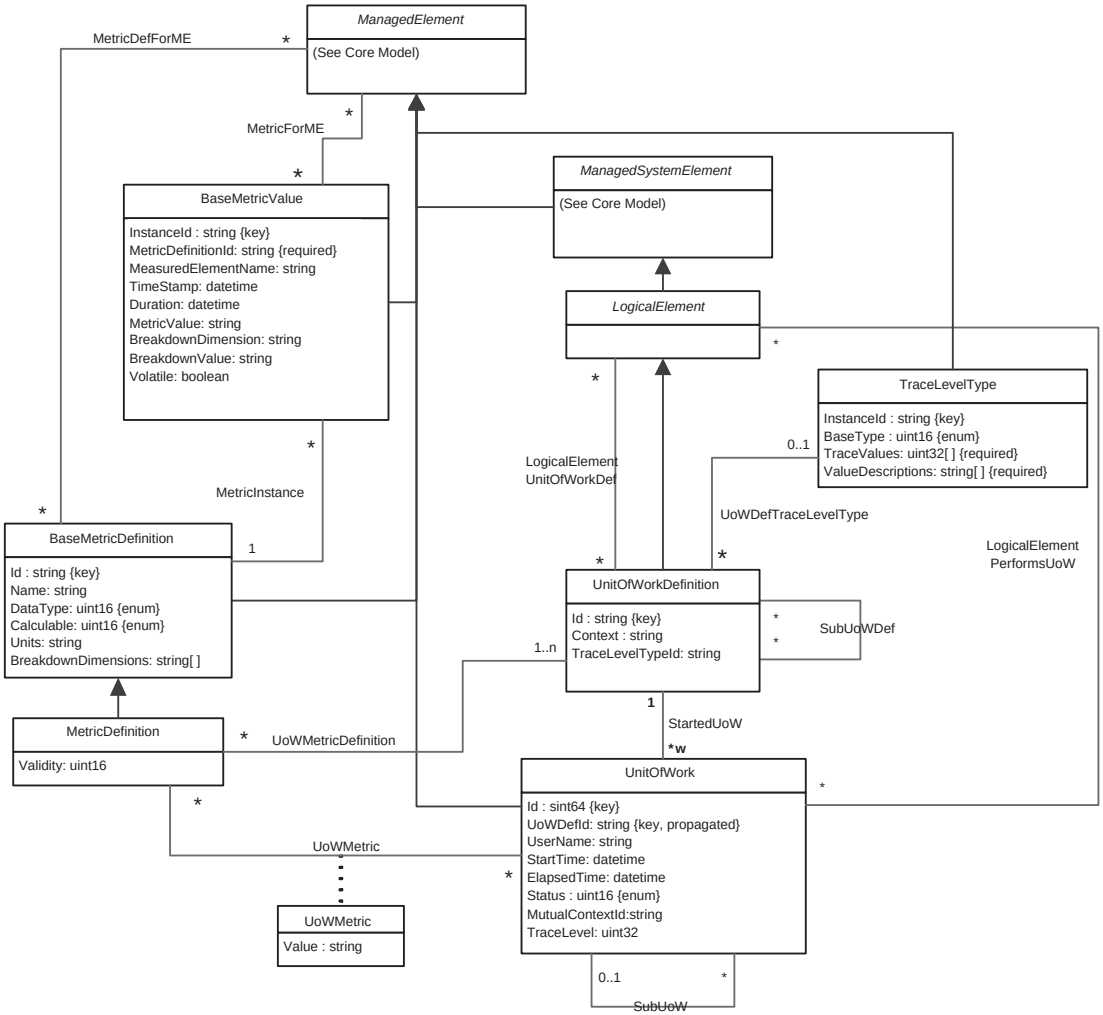


FIG. 6.1 – Le sous-modèle de métrique de CIM

Par le biais de ce modèle, CIM propose une manière standard pour la représentation des informations de gestion inhérentes à la mesure de performance des applications. On peut noter que le modèle proposé par CIM est en fait une adaptation quasi systématique du modèle de classes défini dans l'approche ARM[4] qui propose des concepts équivalents, mais représentés par

[4]Application Response Measurement

des noms de classes différents.

6.3 Notre proposition

Etant donné les motivations exposées dans la section 6.1, nous nous sommes intéressés à la manière dont il serait possible de monitorer la performance d'une DHT [180]. Le travail effectué dans cette voie a tout d'abord consisté à abstraire le fonctionnement d'une DHT. Pour cela, nous avons défini des unités de travail qui représentent les opérations génériques implantées par toutes les DHTs actuelles. Parmi ces opérations, nous nous sommes particulièrement intéressés à celle qui est relative à la localisation des ressources et pour laquelle nous avons établi un modèle d'abstraction. Nous avons adjoint à ce modèle un ensemble de métriques qui caractérisent son fonctionnement. Nous avons exprimé ces différentes abstractions et métriques sous la forme d'un modèle de l'information de gestion qui repose sur le modèle de métriques de CIM.

6.3.1 Définition des unités de travail

Les tables de hachage distribuées reposent sur des modèles topologiques différents et sont constituées de services, dont le fonctionnement diffère d'une infrastructure à une autre. Néanmoins, d'une manière générale, une DHT présente toujours quatre processus, qui sont :

Un processus de localisation : qui assure la localisation des ressources à travers le routage des requêtes vers le nœud racine de leur identifiant ;

Un processus de maintenance : qui met à jour les données inscrites dans les tables de routage des pairs pour assurer la cohérence de la DHT ;

Un processus d'insertion pour les pairs et les clés : qui permet l'insertion d'un nœud ou d'une clé par la recherche du nœud racine ;

Un processus de retrait pour les pairs et les clés : qui permet d'informer les nœuds impliqués du retrait d'un nœud ou d'une clé.

Ces quatre processus sont déployés par toutes les propositions de DHTs actuelles, mais leur fonctionnement varie d'une implantation à une autre. Dans le cadre de notre travail de monitorage de la performance d'une DHT, nous avons pris en compte ces quatre processus. Ceci a conduit à la conception de quatre définitions d'unités de travail telles qu'elles sont représentées sur la figure 6.2. Celles-ci permettent, en accord avec le sous-modèle de métriques du modèle *Common* de CIM, de définir les quatre opérations des DHTs définies ci-dessus. Par la suite, nous décrivons la manière dont nous avons modélisé l'exécution du processus de localisation qui est le processus auquel nous nous sommes particulièrement intéressés.

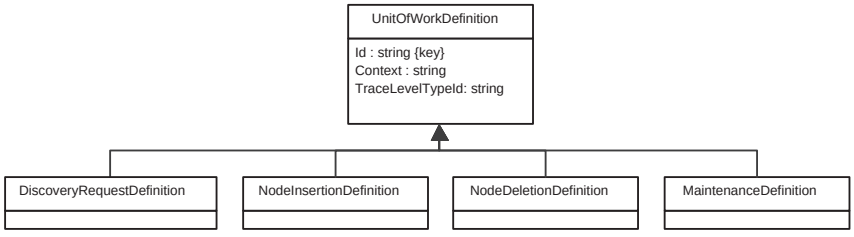


FIG. 6.2 – Définitions des unités de travail des DHTs

6.3.2 Modélisation du processus de localisation et de routage

Le processus de localisation et de routage est le processus qui représente la finalité d'une DHT. Notre travail de modélisation a tout d'abord consisté à analyser le fonctionnement de ce processus pour en concevoir une abstraction applicable à une quelconque DHT. A partir de cette abstraction, nous avons défini les aspects qui nous semblaient significatifs dans le cadre du monitorage de performance et en avons déduit des points de mesure. Afin d'intégrer les données issues de ces points dans notre modèle de l'information de gestion, nous les avons formalisées sous la forme d'unités de travail et de métriques CIM. D'un point de vue de la terminologie, nous introduisons deux termes qui caractérisent le rôle d'un nœud dans le processus de localisation. Nous appelons nœud source, un nœud qui initie une requête de localisation de ressource, et nœud routeur, un nœud qui participe au routage d'une requête initiée par un nœud source. La figure 6.3 illustre ces deux termes à travers l'exemple du routage d'une requête à destination de la clé K_8, initiée par le nœud N_1 et routée par les nœuds N_3 et N_7.

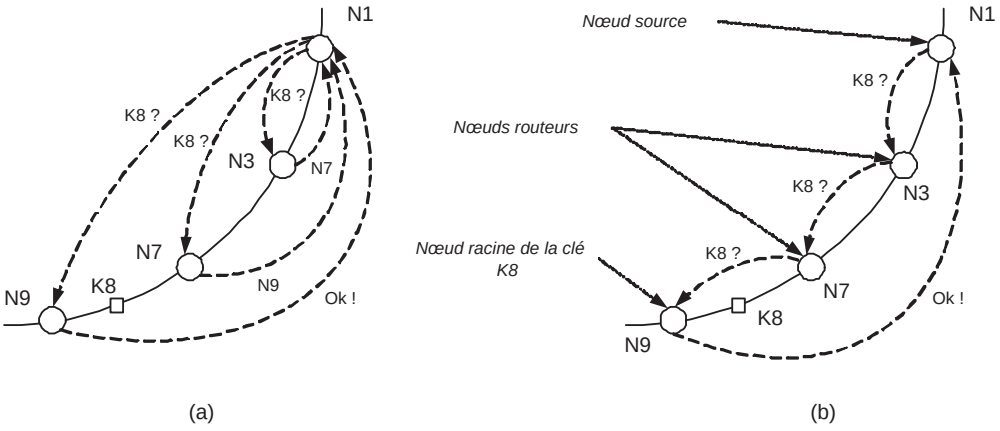


FIG. 6.3 – Les deux méthodes de routage proposées par les DHTs. (a) la méthode itérative. (b) la méthode récursive.

Abstraction du processus de localisation

Le processus de localisation et de routage d'une requête vers une clé donnée s'effectue par approches successives : un nœud source ou routeur d'une requête tente toujours de trouver un nœud plus proche que lui de la clé à atteindre. D'un point de vue de l'implantation, deux méthodes sont proposées dans [153] pour effectuer cette approche. La première, représentée sur la figure 6.3.a est itérative : le nœud source d'une requête contacte, d'après sa table de routage, un nœud plus proche de la clé à atteindre. Le nœud contacté consulte alors sa table de routage et retourne au nœud source l'identifiant d'un nœud plus proche. Le nœud source contacte alors ce dernier nœud, et ainsi de suite. L'exemple donné dans la figure 6.3 illustre le cas d'un nœud N_1 qui désire accéder à la ressource représentée par la clé K_8. On voit que d'après la méthode itérative, N_1 va contacter chacun des nœuds sur le chemin menant à K_8 et attend une réponse de chacun d'eux pour connaître l'identifiant du prochain saut. Cette méthode est assez coûteuse en termes de nombre de messages à échanger. Par contre, la seconde méthode, illustrée sur la figure 6.3.b est récursive. Elle fait passer la requête initiée par un nœud source de nœud routeur en nœud routeur jusqu'à atteindre la clé requise. Nous considérons à nouveau l'exemple illustré

sur la figure 6.3 qui montre que cette méthode est moins coûteuse que la précédente. Elle est ainsi la méthode la plus couramment implantée dans les DHTs actuelles. En outre, c'est celle que nous avons considérée pour modéliser le processus de localisation.

Notre proposition d'abstraction du processus de localisation et de routage est représentée sur la figure 6.4. Elle est valable à la fois pour les nœuds sources mais aussi les nœuds routeurs. Les opérations effectuées par ces deux catégories de nœuds sont représentées à l'aide de traits continus, par contre, les opérations effectuées seulement par les nœuds sources sont représentées en traits discontinus. Nous détaillons maintenant étape par étape les différentes opérations effectuées par les nœuds dans ce cadre.

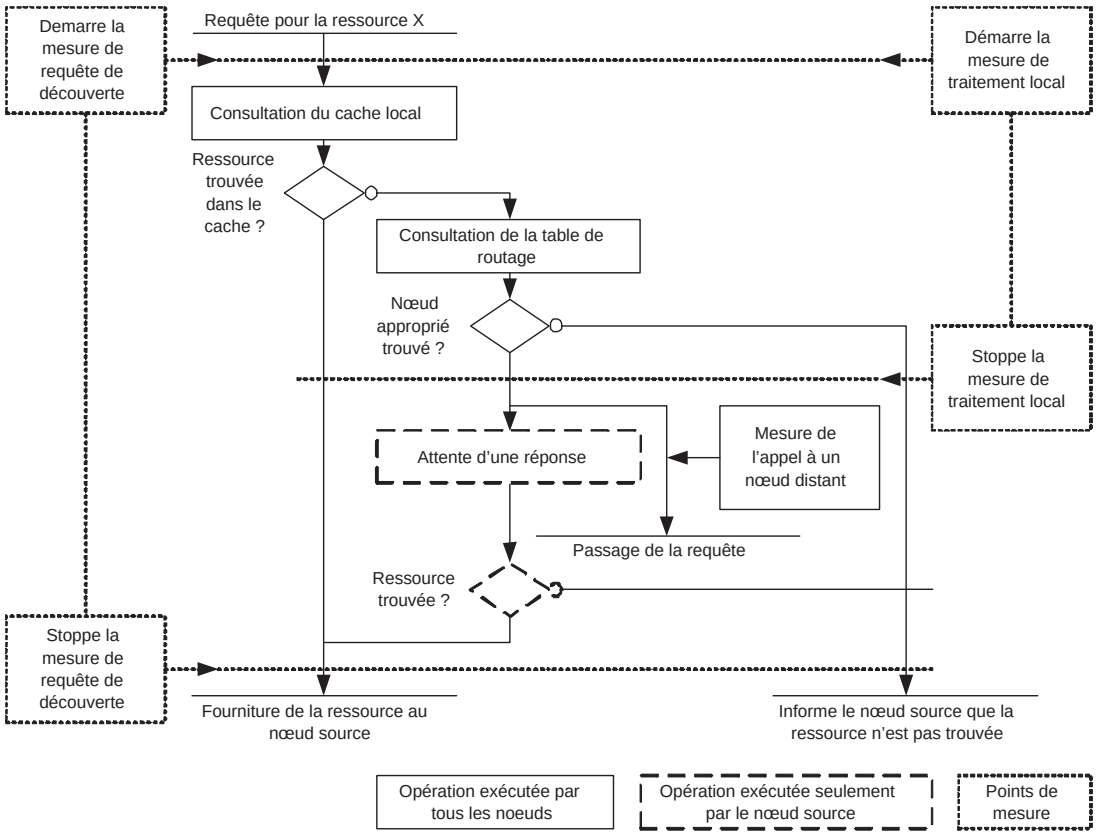


FIG. 6.4 – Abstraction du processus de localisation d'une ressource

Lorsqu'une requête pour une clé donnée est reçue par un nœud, celui-ci va tout d'abord vérifier dans son cache local s'il n'est pas la racine de cette clé. Si oui, il retourne un pointeur vers la ressource à la source qui peut être un nœud distant ou une couche logicielle de niveau supérieur si la requête n'a pas quitté le nœud source. Sinon, le nœud consulte sa table de routage pour trouver un nœud plus proche de la clé requise. Si aucun nœud n'est trouvé, une réponse d'échec est fournie au nœud source. Dans le cas contraire, la requête est transférée à ce dernier nœud. Ensuite, d'après la méthode récursive, si l'on considère l'exécution de ce processus telle qu'elle est faite dans le nœud source, on voit que ce dernier va attendre la réponse à la requête qu'il a soumise, par contre si l'on considère un nœud routeur celui-ci va simplement faire suivre la requête sans attendre de réponse. Pour finir, le nœud source termine le processus de localisation

lorsqu'une réponse positive ou négative est reçue[5].

Maintenant, lorsqu'on examine le processus de localisation tel qu'il est décrit dans la figure 6.4, on constate qu'il se scinde en deux parties. La première représente le traitement local qu'un nœud effectue lorsqu'une requête arrive et consiste en la recherche de la clé requise dans le cache local et d'un nœud approprié dans la table de routage. La seconde représente le passage de la requête à un nœud distant. Nous proposons de caractériser ces deux parties du processus de localisation à travers la définition de métriques qui sont :

Le résultat de la requête : Cette métrique renseigne sur la manière dont une requête s'est terminée. Elle n'est donc considérée que par le nœud source. Ses valeurs indiquent entre autre qu'une ressource a été localisée, ou qu'elle n'a pas été trouvée, ou, si un problème survenait, la nature de problème (nœud inaccessible, requête mal formée, ...).

Le temps total de traitement : Cette métrique présente un sens différent selon que l'on considère le nœud source d'une requête ou un nœud routeur. Pour le nœud source, cette métrique représente le temps écoulé entre le moment où la requête a été initiée et le moment où la réponse est arrivée, soit le temps total nécessaire à la localisation d'une ressource. Par contre, pour un nœud routeur, cette métrique représente le temps total nécessaire au traitement de la requête, qui comprend la recherche dans le cache, la consultation des tables de routage et le passage à un nœud suivant.

Le temps de traitement local : Cette métrique représente le temps nécessaire à un nœud pour consulter son cache local et sa table de routage. Nous considérons que cette métrique est importante notamment pour déceler des problèmes de passage à l'échelle et d'équilibre de la répartition des clés.

Le trafic généré : Cette métrique vise à mesurer la quantité de trafic généré pour passer une requête à un nœud suivant. Elle permet de mesurer la quantité totale de trafic nécessaire au routage d'une requête de localisation.

Les points de mesure correspondant à ces dernières trois métriques sont représentés sur la figure 6.4 par des traits pointillés.

Proposition d'un modèle

Notre proposition de modélisation de la performance du processus de localisation est fortement inspirée d'un exemple, proposé dans [46], de mesure de transactions d'une application distribuée. Celui-ci s'éloigne un peu du modèle général de métriques décrit dans la section 6.2 et propose notamment, par souci de synthèse et de simplicité, d'intégrer les métriques comme attributs des unités de travail.

Ainsi, le modèle que nous proposons pour caractériser la performance des DHTs est représenté sur la figure 6.5. Il comporte trois unités de travail, qui représentent le traitement d'une requête par un nœud dans sa globalité, la partie locale et l'envoi éventuel de la requête à un nœud routeur. La classe DiscoveryRequest représente une tâche de traitement de requête de localisation. Cette classe est instanciée par le nœud source d'une requête et par tout nœud routeur. Pour distinguer le contexte dans lequel elle est instanciée, nous lui avons adjoint les attributs Action et ActionType qui représentent l'action effectuée par le nœud qui l'instancie (à savoir source ou forwarding). Lorsqu'elle est instanciée par le nœud source, elle représente alors la requête dans sa globalité. C'est pourquoi, nous avons défini les attributs NumberOfHops, NumberOfMessages,

[5] Bien sûr dans le cadre d'une implantation, un *timeout* est mis en place pour éviter une attente infinie, mais nous ne le considérons pas à ce niveau d'abstraction

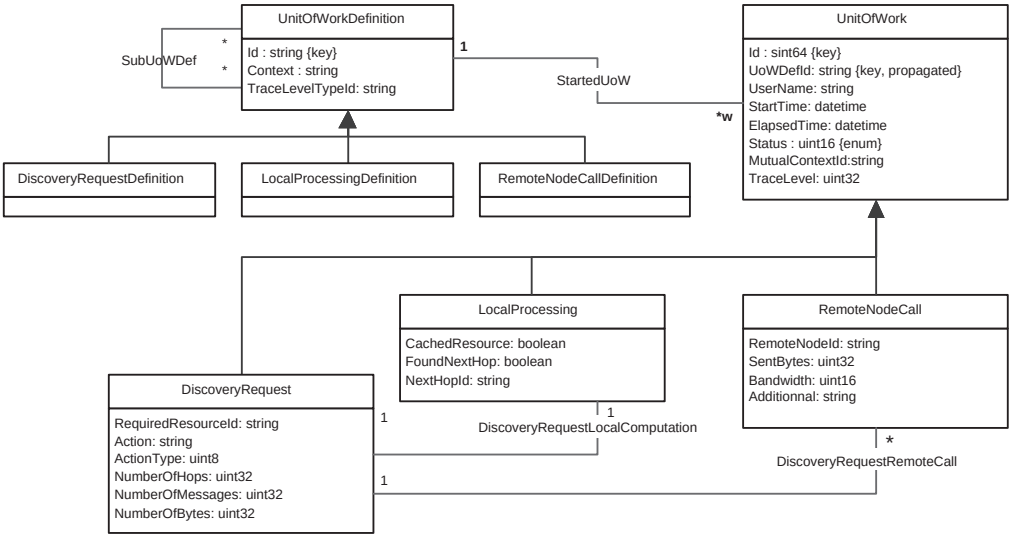


FIG. 6.5 – Modélisation des unités de travail du processus de localisation

NumberOfBytes qui caractérisent respectivement le nombre de sauts effectués, le nombre de messages échangés et le nombre d'octets contenus dans l'ensemble des messages. Dans le cas d'un nœud routeur, ces informations sont hors de propos. Enfin, l'attribut RequiredResourceId représente la clé de la ressource recherchée par la requête. Cette information est intéressante car elle permet d'analyser le taux d'accès à chaque ressource et, dans le cas où un mécanisme de redondance serait déployé, elle permet de détecter un éventuel déséquilibre qui conduirait à la duplication de la ressource concernée.

La classe LocalProcessing représente le traitement local qu'un nœud effectue. L'attribut FoundInCache indique la présence ou non de la clé dans le cache local et les attributs FoundNextHop et NextHopId renseignent sur le fait qu'un nœud plus proche ait été trouvé, et le cas échéant son identifiant. Enfin, la classe RemoteNodeCall caractérise le passage de la requête d'un nœud à un autre. Elle contient notamment l'identifiant du nœud atteint (RemoteNodeId) et le nombre d'octets transmits (SentBytes).

Par le biais de ce modèle, un gestionnaire est capable de monitorer complètement les requêtes de localisation de ressource qui sont propagées dans une DHT.

6.3.3 Corrélation des mesures

Les requêtes initiées par des nœuds conduisent à des transactions impliquant d'autres nœuds. Le modèle de l'information présenté dans la section précédente permet de caractériser les opérations effectuées par chaque nœud participant à une requête. Maintenant, la question induite par ce modèle est : *Comment mettre en relation les instances de classes inhérentes à différents nœuds, mais toutes induites par le traitement d'une unique requête ?* Autrement dit : *Comment corréler les objets gérés dans le contexte d'une requête particulière ?* Pour cela, le modèle de métrique de CIM ne propose pas de solution générale, mais un exemple issu de [46] propose une solution que nous avons utilisée. Celle-ci consiste à utiliser une classe nommée DHT_Correlator qui hérite directement de la classe abstraite CIM_ManagementElement et qui va représenter de manière unique le contexte d'une requête. Pour ce faire, la classe DHT_Correlator est associée à une unité de travail par le biais de la classe DHT_UnitOfWorkCorrelation. Sa cardinalité stipule

qu'un corrélateur est attaché à une seule unité de travail et qu'une unité de travail peut présenter zéro ou un corrélateur. Concernant les attributs, la classe DHT_Correlator se caractérise principalement par un identifiant de transaction (TransactionId) qui identifie de manière unique la requête initiée par un nœud.

Le fonctionnement du corrélateur est le suivant : pour chaque requête initiée, une instance de cet objet est passée à chaque nœud routeur et permet d'associer chaque instance d'unité de travail induite par le traitement de la requête à ce corrélateur. Ainsi chaque objet est mis en relation avec la requête initiée, dont il représente une partie.

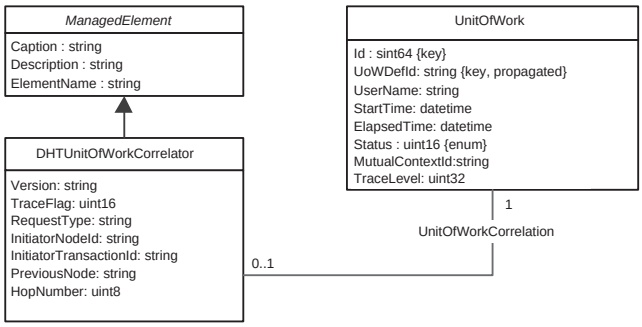


FIG. 6.6 – Modélisation de la corrélation des unités de travail

6.4 Instanciation du modèle de l'information sur Chord

Le modèle de l'information présenté dans la section précédente est générique et est ainsi applicable à n'importe quelle DHT. Pour affiner ce travail, nous avons considéré une infrastructure particulière qui puisse instancier ce modèle générique et pour laquelle nous puissions définir des métriques propres [174]. Nous avons choisi Chord car c'est une proposition reconnue et standard dans la communauté des DHTs. En outre, la simplicité de ses concepts en font un bon candidat dans le cadre de notre travail de monitorage de performance.

Dans cette section, nous présentons donc de nouvelles métriques, propres à Chord, qui viennent compléter les métriques génériques exposées précédemment. Nous détaillons en particulier la manière dont nous proposons de les calculer. Ensuite, nous présentons le modèle de l'information de gestion à caractère statistique qui les intègre et permet, sur le plan de la gestion, de représenter les performances de Chord, mais aussi sa topologie, son organisation et ses services.

6.4.1 Définition de métriques

Les métriques que nous avons définies font appel aux variables suivantes :

N_{MAX} **:** la valeur numérique la plus grande correspondant à un identifiant pour les pairs et les clés. La fonction de hachage de Chord génère des identifiants compris dans l'intervalle $[0, N_{MAX}[$ et toutes les opérations sont effectuées modulo N_{MAX} ;

n_i **:** un nœud d'identifiant i ;

N **:** l'ensemble des nœuds n_i qui sont actuellements présents et joignables dans un anneau, avec $N = \{n_i\}$;

N_T **:** le nombre total de nœuds dans un anneau, avec $N_T = Card(N)$;

k_i **:** une clé d'identifiant i ;

K_i **:** le nombre de clés dont le pair n_i est responsable, avec $K_i = Card(\{k_j\})$ et $id(pred(n_i)) \leq j \leq i$.

K_T **:** le nombre total de clés référencées dans un anneau, avec $K_T = \sum_{i \in N} K_i$.

$\overline{K}$ **:** le nombre moyen de clés par pair, avec $\overline{K} = \frac{K_T}{N_T}$

Mesure de la dynamique de l'anneau

Avant de définir des métriques pour la mesure de performance d'un anneau Chord, nous avons estimé qu'il était important de pouvoir en évaluer l'état. Nous avons ainsi défini des métriques qui caractérisent un anneau Chord du point de vue de la stabilité des pairs et des clés. De la même manière que nous avons défini deux niveaux d'abstraction pour les services P2P, nous proposons de définir ces métriques sur deux niveaux : un niveau local inhérent à un élément, comme un pair ou une clé, et un niveau global qui concerne un anneau dans sa globalité.

La caractérisation de la stabilité de l'anneau est un indicateur qui va permettre de relativiser et expliquer les valeurs des métriques définies par la suite. Par exemple, l'augmentation brutale du nombre moyen de sauts nécessaires au routage d'une requête peut s'expliquer par un fort taux d'arrivée et de départ des nœuds qui perturberait la cohérence des données maintenues dans les tables de routage. La métrique de dynamique de l'anneau permet ainsi de détecter ce fort changement et d'expliquer la baisse de performance du processus de localisation.

Dans un anneau Chord, les deux éléments soumis à des changements sont les pairs et les clés. Les pairs peuvent joindre et quitter la communauté de manière imprévisible, de même que les clés peuvent être insérées ou retirées, ou migrer pour assurer leur persistence. Ainsi, pour ces deux éléments, nous avons choisi de caractériser leur temps moyen de présence et la fréquence à laquelle ils subissent des changements. Pour ce faire, nous considérons que les temps d'arrivée et de départ sont connus et stockés dans un journal. Ceci permet de calculer directement leur temps de présence dans l'anneau et de déduire la fréquence à laquelle ils subissent des changements, considérée comme l'inverse de l'intervalle de temps séparant deux événements. Ensuite, d'un point de vue global, nous proposons d'évaluer le taux d'arrivée et de départ des clés et des nœuds par le biais des relations 6.1 et 6.2. Dans la suite, nous expliquons ces relations en considérant le cas des nœuds. Lorsqu'un nœud joint ou quitte l'anneau, la date courante est relevée et sert à calculer les indicateurs $JoinFrequency$ et $LeaveFrequency$. La valeur de cette date courante est ensuite stockée dans les variables $LastArrivalTime$ et $LastDepartureTime$, selon le cas, pour calculer la fréquence d'apparition de l'événement suivant.

$$JoinFrequency = \frac{1}{CurrentTime - LastArrivalTime} \tag{6.1}$$

$$LeaveFrequency = \frac{1}{CurrentTime - LastDepartureTime} \tag{6.2}$$

Le temps de présence moyen d'un nœud est calculé par l'équation 6.3. On considère une fenêtre temporelle T dont la taille dépend du contexte et peut s'adapter au niveau de dynamisme estimé. Pour tous les pairs qui ont participé à l'anneau durant l'intervalle de temps défini par T, tous leurs temps de présence sont moyennés de manière à fournir une estimation globale.

$$PresenceMeanTime = \frac{\sum_{i \in N} \sum_{t \in T_i} t}{\sum_{i \in N} Card(T_i)} \text{ with} \tag{6.3}$$

T : la fenêtre temporelle de calcul du temps de présence moyen, avec $T = [T_{Begin}, T_{End}]$;

T_i : l'ensemble des temps de présence collectés dans la fenêtre T pour le nœud n_i, avec $T_i = \{t \mid t = (T_{Departure} - T_{Arrival}); T_{Departure}, T_{Arrival} \in T\}$.

Maintenant, concernant les clés, nous avons défini des métriques identiques à celles proposées pour les pairs. Ceci nous permet de représenter les fréquences globales d'insertion, de migration et de retrait des clés.

Par le biais des métriques définies dans cette section nous ne sommes pas capables d'évaluer la performance d'une communauté Chord mais la manière dont la communauté évolue, en termes de dynamisme des clés et des pairs. Cette métrique sert à mettre en relation les métriques que nous définissons dans les section suivantes avec l'état d'un anneau.

Mesure de la performance du processus de localisation

Dans la section 6.3.2, nous avons proposé un modèle qui permet de tracer les requêtes de localisation effectuées au sein d'une DHT. Dans cette section, nous proposons la définition d'une métrique statistique relative à la performance du processus de localisation qui est calculée grâce aux données fournies par le modèle de métriques et d'unités de travail du processus de localisation. La grandeur que nous proposons de monitorer est le nombre moyen de sauts par requête de localisation, pour laquelle nous proposons la définition d'une métrique. Cette métrique est intéressante car elle peut être directement comparée aux performances théoriques attendues, à savoir $\frac{1}{2}\log_2 N_T$ et indiquer le bon état d'un anneau Chord. L'algorithme que nous proposons pour le calcul de cette métrique est décrit dans l'équation 6.4. Pour chaque pair n_i, nous définissons les métriques suivantes :

- $NInitiatedRequests_i$: Le nombre total de requêtes de localisation dont le nœud n_i est la source ;
- $NForwardedRequests_i$: Le nombre total de requêtes pour lesquelles le nœud n_i a été routeur ;
- $NReceivedRequests_i$: Le nombre total de requêtes à destination du nœud n_i.

Ensuite, nous additionnons chacune de ces métriques et déduisons le nombre moyen de sauts nécessaires au routage des requêtes comme suit :

$$AveragePathLength = \frac{\sum_{i \in N} NReceivedRequests_i + \sum_{i \in N} NForwardedRequests_i}{\sum_{i \in N} NInitiatedRequests_i} \quad (6.4)$$

Mesure de l'équilibre des clés

Contrairement au modèle client-serveur qui centralise ses ressources, le modèle P2P les distribue parmi l'ensemble des pairs participants. Les DHTs reposent sur des modèles qui appuient leurs propriétés de performance de localisation sur une répartition équilibrée des clés. La fonction de hachage qui génère les identifiants de nœuds et de clés garantit *avec une très forte probabilité* que les clés seront réparties de manière équitable parmi les nœuds présents. Néanmoins, dans le cas d'un déploiement réel d'une DHT, mettant en œuvre des pairs et des clés qui vont et viennent de manière imprévisible, il est impossible de savoir si cet équilibre sera toujours assuré. Or, un déséquilibre de cette répartition peut fortement dégrader les performances du processus de localisation. A l'extrême, si seuls quelques pairs devenaient responsables de la majorité des clés, la DHT ne fonctionnerait plus selon le modèle P2P mais selon le modèle client/serveur, et elle ne pourrait plus garantir aucune propriété en termes de passage à l'échelle, de tolérance aux

fautes et d'équilibre de la charge et du trafic. C'est pourquoi, dans notre travail de caractérisation de la performance de Chord, nous avons choisi de monitorer la répartition de clés au sein d'un anneau. Pour cela, nous avons établi une métrique qui mesure le niveau de respect de cet équilibre. Dans [153], les auteurs annoncent que chaque nœud Chord est responsable d'au plus $(1+\epsilon)\overline{K}$ clés. Nous proposons de surveiller cette valeur pour chaque nœud. D'un point de vue global, l'équation 6.5 montre la manière dont nous estimons l'équilibre global d'un anneau. Pour chaque nœud n_i, nous considérons la différence entre le nombre de clés hébergées par le nœud et la valeur moyenne théorique estimée pour l'anneau. Nous effectuons ensuite la moyenne de ces écarts pour chaque nœud participant et obtenons un indicateur, compris entre 0 et 1 qui représente le niveau d'équilibre de l'anneau.

$$NodesEquity_{\%} = 1 - \frac{1}{K_T} \sum_{i \in N} \lfloor \ | K_i - \overline{K} | \ \rfloor \tag{6.5}$$

En proposant cette définition de métrique, nous ne faisons aucune supposition sur la manière dont elle est calculée. On voit que son calcul requiert la connaissance du nombre total de clés K_T et du nombre moyen de clés par nœud $\overline{K}$ qui sont des valeurs globales qu'un nœud doit pouvoir déterminer. Plusieurs manières sont envisageables pour les calculer. La première consiste à interroger un gestionnaire qui posséderait cette connaissance par le biais de la gestion. Cette méthode est systématique et adhère au mieux à la réalité. La seconde consiste à procéder par estimation. Cette méthode est utilisée dans [103] pour estimer le nombre total de pairs présents dans une communauté Viceroy. Chaque nœud interroge ses voisins et collecte des informations qui lui permettent de calculer une estimation de $\overline{K}$ et K_T. De cette manière le calcul de la métrique d'équilibre de clés peut être accompli de manière totalement distribuée.

Mesure de la cohérence de l'anneau

Lorsque l'on considère les performances d'une DHT, on se place dans un contexte statique, où les pairs et les clés ne changent pas d'état. Cependant, dans un contexte dynamique, on peut se demander ce que deviennent le bon fonctionnement et les performances du processus de routage. *Comment un pair Chord peut il faire suivre des messages lorsque les entrées de sa table de routage sont erronées ?* Et, *Comment évolue le nombre moyen de sauts des requêtes dans ce contexte ?*

Nous pensons donc qu'être capable, d'un point de vue de la gestion, d'estimer la cohérence d'un anneau est une information intéressante, qui pourrait conduire à différentes actions de gestion. Par exemple, dans le cas où le niveau de cohérence général de l'anneau est trop bas, conduisant à l'effondrement des performances du processus de localisation[6], un gestionnaire pourrait ordonner l'exécution du processus de maintenance sur certains nœuds afin de restaurer un niveau de cohérence correct.

C'est pourquoi, en accord avec les définitions proposées dans [153], nous avons défini trois contraintes qui définissent la cohérence des informations stockées dans les tables de routage de nœuds Chord. Elles sont :

- **Contrainte 1** *Cohérence de l'anneau. Cette contrainte est respectée si, étant donné l'ensemble des nœuds présents dans l'anneau, chaque nœud est effectivement le précédent de son suivant.*
- **Contrainte 2** *Cohérence de la liste de suivants. Chaque nœud maintient une liste de k suivants dans l'anneau. Cette contrainte est respectée si, étant donné l'ensemble des*

[6] en termes de taux de succès des requêtes et de longueur moyenne des chemins empruntés

nœuds présents dans l'anneau, la liste maintenue par chaque nœud contient effectivement les suivants immédiats et si ces suivants sont joignables.

– **Contrainte 3** *Cohérence de la table de routage. Cette contrainte est respectée si, étant donné l'ensemble des nœuds présents dans l'anneau, chaque finger de chaque nœud respecte la relation* $finger_k = suivant(n_i + 2^{k-1})$*, où* $1 \leq k \leq m$ *et est joignable.*

Etant donné les trois contraintes ci-dessus, nous considérons qu'un anneau Chord est cohérent si, pour chaque nœud présent, chacune d'elles est respectée. Ceci nous a conduit à définir, pour chaque nœud n_i, quatre variables booléennes qui renseignent sur le respects de ces contraintes. La première, $IsConsistent_i$ indique si le nœud respecte ou non les trois contraintes. Les trois suivantes, $SuccessorIsConsistent_i$, $SuccessorListIsConsistent_i$ et $FingerTableIsConsistent_i$ renseignent sur le respect de chacune des contraintes respectives 1, 2 et 3.

Les valeurs prises par ces variables sont données par un gestionnaire qui peut les estimer par comparaison entre la vue qu'il possède d'un anneau et les données de gestion issues de l'instrumentation des nœuds, telles que le précédent, les tables de routage et la liste des suivants.

Pour estimer le niveau de cohérence globale d'un anneau Chord, le gestionnaire dispose de deux indicateurs. Le premier, $IsGloballyConsistent$, indique de manière booléenne si l'anneau est dans un état cohérent ou non. Il est calculé d'après la formule 6.6.

$$IsGloballyConsistent = \bigwedge_{i \in N} IsConsistent_i \tag{6.6}$$

Le second affine l'information fournie par le premier en indiquant le niveau de cohérence global de l'anneau. Il est calculé selon l'équation 6.7 et définit, sous forme de pourcentage, le niveau de cohérence global. La fonction $valueOf$ retourne 1 lorsque $IsConsistent_i$ est vrai et 0 sinon.

$$GloballyConsistencyLevel_{\%} = \frac{1}{N_T} \sum_{i \in N} valueOf(IsConsistent_i) \tag{6.7}$$

Enfin, nous avons défini un dernier indicateur qui compte pour chaque nœud le nombre de fois où il est mal référencé par d'autres nœuds. Cette indication aide à localiser les zones de l'anneau qui sont dans un état incohérent et permet par exemple à un gestionnaire de provoquer l'exécution du processus de maintenance seulement sur des nœuds qui sont mal référencés.

6.4.2 Modélisation de l'information de gestion

Sur la base des métriques de performance présentées dans la section précédente, nous avons développé une version étendue de notre modèle de l'information de gestion générique, présenté dans le chapitre 5, qui considère la modélisation d'une communauté Chord orientée vers la performance. Dans cette section, nous présentons ce modèle de l'information, représenté sous forme d'un schéma CIM nommé Chd. Nous détaillons particulièrement les sous-modèles qui concernent les nœuds et la topologie Chord, les clés et les ressources, et enfin, les services de localisation et de maintenance.

Un de nos choix de conception a porté sur la simplicité du modèle. C'est pourquoi, bien que les métriques que nous avons définies précédemment puissent être considérées comme des données statistiques et ainsi intégrées dans le sous-modèle de CIM approprié [40], nous avons préféré les intégrer directement dans les classes de ce nouveau modèle pour en simplifier l'interprétation et l'utilisation.

Modélisation de la topologie et l'organisation

Le modèle topologique et organisationnel que nous proposons est représenté sur la figure 6.7. Chord organise sa topologie selon un anneau ordonné. En ce sens, un anneau représente une communauté, que nous modélisons à travers la classe Chd_ChordRing qui hérite de la classe P2P_Community. Les propriétés de cette classe sont divisées en deux catégories : la première fournit des informations générales sur la communauté, comme le type de méthode de hachage (HashMethod) ou la taille des identifiants (IdentifierLength). La seconde catégorie contient les métriques de performances relatives à la dynamique de l'anneau, comme par exemple la fréquence d'arrivée et de départ des nœuds (NodesJoinFrequency et NodesLeaveFrequency, respectivement).

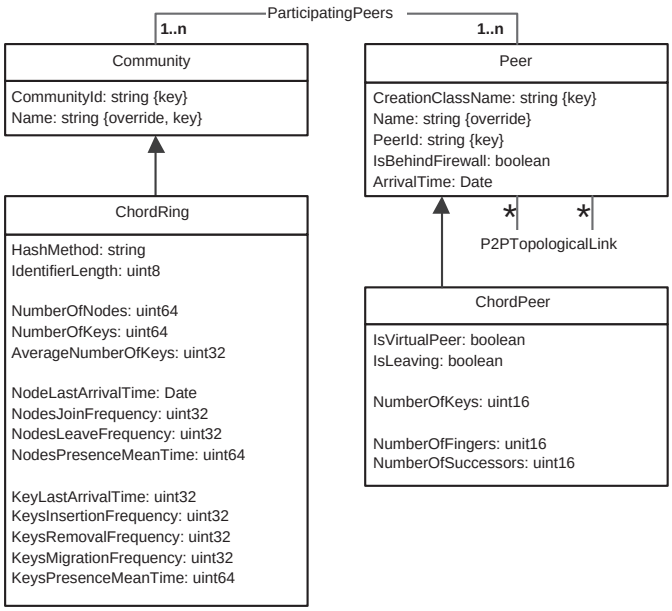


FIG. 6.7 – La spécialisation du modèle de nœuds et de communauté pour Chord

Les nœuds Chord sont représentés par la classe Chd_ChordPeer. Cette classe hérite de la classe P2P_Peer et, en plus des propriétés définies dans cette classe, elle caractérise un nœud Chord à travers : le nombre de clés qu'il héberge (NumberOfKeys), le fait que le nœud soit virtuel[7] (IsVirtual) et le fait que le nœud quitte l'anneau (IsLeaving). Cette dernière information, lorsqu'elle est connue, est utile à un gestionnaire pour planifier une réorganisation de l'anneau et des clés. Enfin, les attributs NumberOfSuccessor et NumberOfFingers permettent de renseigner respectivement sur le nombre de successeurs et le nombre de *fingers* présents dans la table de routage Chord.

Concernant la topologie, nous avons étendu la classe P2P_TopologicalLink et conçu trois nouvelles classes d'association spécialisées, représentées sur la figure 6.8. Ces classes ne définissent pas d'attribut particulier, mais elles permettent, de par leur nom, d'identifier simplement le type de lien topologique qu'elle représentent. La classe Chd_PredecessorLink représente un lien qu'un nœud possède vers son précédent dans l'anneau. La classe Chd_SuccessorLink représente un lien vers un suivant. Plusieurs instances de cette classe sont nécessaires pour représenter l'ensemble

[7]Pour améliorer l'équilibre de l'anneau, plusieurs nœuds Chord peuvent être executés simultanément sur une machine, on parle alors de nœuds virtuels.

des liens vers chacun des suivants d'un nœud. Enfin, la classe Chd_FingerLink représente un lien vers un *finger*. Chaque *finger* d'un nœud obéit à la relation $suivant(n_i + 2^{k-1})$, où $1 \leq k \leq m$ et l'attribut Rank représente k.

A travers ce sous-modèle, nous autorisons un gestionnaire à reconstituer une vue complète d'un anneau Chord, de ses nœuds participants et des relations topologiques qu'il présente.

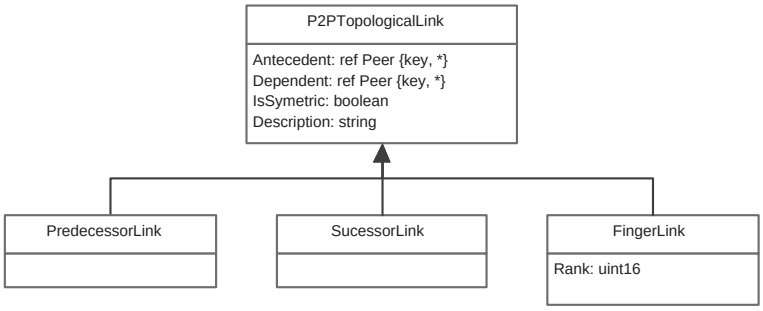


FIG. 6.8 – Les classes d'association qui permettent de représenter la topologie Chord

Modélisation des clés et des ressources

Chaque nœud Chord est responsable d'un ensemble de clés dont chacune représente un pointeur vers une ressource. Néanmoins, la nature des ressources est spécifique à l'application qui utilise Chord. Par exemple, dans le cas de CFS [30] une ressource est le morceau d'un fichier distribué parmi un ensemble de nœuds. Par contre, dans DDNS [29], une ressource représente une entrée d'une table DNS. C'est pourquoi, nous avons choisi de modéliser une ressource Chord par le biais de la classe Chd_ChordKey, représentée sur la figure 6.9. Celle-ci hérite de la classe P2P_PeerResource du sous-modèle générique de ressources P2P. L'attribut LastMigrationTime de la classe Chd_ChordKey indique la date du dernier déplacement de la clé et est utilisé pour estimer la dynamique de l'anneau, d'après la métrique présentée dans la section 6.4.1.

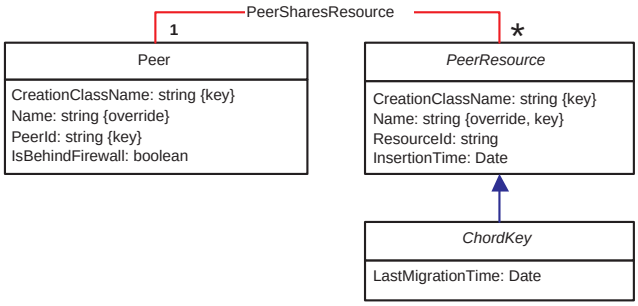


FIG. 6.9 – Le modèle de ressources de Chord

Modélisation des services Chord

Parmi les quatre services de Chord identifiés dans la section 6.3.1, nous nous sommes particulièrement intéressés aux services de localisation et de stabilisation. Pour ces deux services, nous avons conçu un modèle de l'information qui intègre les métriques que nous avons définies.

Ce modèle est représenté sur la figure 6.10. En accord avec notre modèle générique pour les services P2P, chaque service Chord est décomposé en une vue locale à un nœud particulier et une vue globale relative à un anneau.

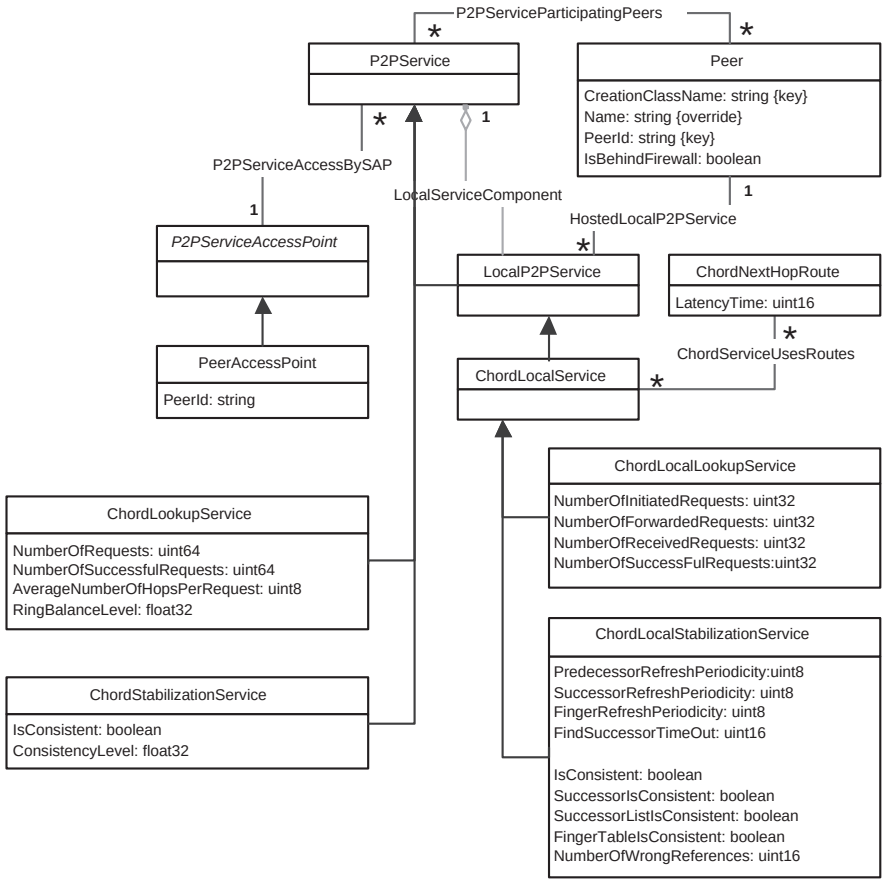


FIG. 6.10 – Le modèle de service de Chord

Le service de localisation est représenté à travers les classes Chd_ChordLookupService et Chd_LocalChordLookupService qui concernent respectivement le niveau global et le niveau local de ce service. Les propriétés de ces classes permettent le calcul des métriques de performance de Chord. Les valeurs globales sont souvent issues d'un calcul effectué d'après les valeurs des attributs locaux. Par exemple, l'attribut AverageNumberOfHopsPerRequest de la classe Chd_ChordLookupService, correspondant à la métrique définie par l'équation 6.4 de la section 6.4.1, est calculé à l'aide des attributs locaux NumberOfInitiatedRequests, NumberOfForwardedRequests et NumberOfReceivedRequests.

Le service de stabilisation est représenté à travers les classes ChordStabilizationService et LocalChordStabilizationService. De même que pour le service de localisation, les attributs des classes qui le représentent autorisent le calcul des métriques de performance et en particulier la métrique de mesure du niveau de cohérence de l'anneau. Au niveau local, nous avons ajouté des informations inhérentes à l'exécution du processus de stabilisation, comme la fréquence d'exécution de ce processus pour les différentes entrées de la table de routage d'un nœud (PredecessorRefreshPeriodicity, SuccessorRefreshPeriodicity et FingerRefreshPeriodicity).

6.5 Synthèse

La performance des DHTs a été évaluée et les infrastructures comparées dans de nombreux travaux reposant sur des approches théoriques ou de simulations. Or dans le cas d'un déploiement effectif, du fait de la nature dynamique du modèle P2P, les performances théoriques ne peuvent pas être garanties naturellement et perpétuellement. Une infrastructure de gestion est nécessaire pour pouvoir assurer un niveau de performance donné. La contribution que nous avons exposée dans ce chapitre est un modèle de l'information de gestion, orienté vers la performance pour les tables de hachage distribuées. Nous avons tout d'abord identifié les différentes opérations effectuées par ces infrastructures et nous nous sommes particulièrement intéressés au processus de localisation de ressources. Pour celui-ci, nous avons conçu une abstraction de fonctionnement et intégré trois métriques qui le caractérisent. Ensuite, à l'aide du sous-modèle de métriques du modèle *Common* de CIM, nous avons conçu un modèle de l'information de gestion pour les DHTs qui intègre les métriques définies.

Comme cas d'application de ce modèle, nous avons considéré Chord. Pour cette infrastructure, nous avons défini des métriques particulières qui affinent le travail générique précédent. Enfin, nous avons à nouveau intégré ces métriques dans une extension de notre modèle de l'information générique dédiée à cette infrastructure particulière. Par ce biais, un gestionnaire dispose d'informations inhérentes au fonctionnement de Chord, aux opérations effectuées mais aussi à l'état dans lequel se trouve l'anneau.

Chapitre 7

Organisation d'un plan de gestion adapté au modèle pair à pair

Sommaire

Dans ce chapitre, nous présentons une proposition d'architecture de gestion pour les réseaux et services P2P. Cette architecture repose sur un modèle gestionnaire/agent et établit une structure de gestion hiérarchique. Dans un premier temps, nous exprimons les motivations qui nous ont conduits à la conception d'une telle infrastructure. Afin de situer notre contribution en regard des travaux actuels, nous passons en revue différentes propositions de construction de structures arborescentes qui présentent des similitudes avec notre travail. Ensuite, nous présentons notre contribution, avec son principe général et sa formalisation à travers un algorithme centralisé [177]. La manière dont nous avons distribué et déployé cet algorithme est ensuite abordée. Nous détaillons les opérations effectuées par plusieurs protocoles qui permettent de gérer l'arrivée de pairs, leur départ, et le maintien de la structure hiérarchique de gestion. Pour terminer, nous présentons les résultats de premiers tests que nous avons effectués pour valider cette proposition.

7.1 Contexte

Cette section situe le cadre de notre travail, avec ses motivations, les travaux relatifs et les objectifs que nous nous sommes fixés.

7.1.1 Motivations

Une fois le travail de modélisation des informations de gestion inhérentes au modèle P2P terminé, nous nous sommes intéressés à la manière dont il serait possible de le déployer. Pour ce faire, nous nous sommes posé plusieurs questions : *Qui crée les instances d'objets gérés ? Où sont stockées ces instances ? Qui y accède ?* Et *Comment y accède-t-on ?* Derrière ces questions relatives au seul déploiement de notre modèle de l'information se trouvent en fait des questions inhérentes au problème plus général de l'organisation d'un plan de gestion adapté au modèle P2P : *Qu'est-ce qu'un agent de gestion ? Qu'est-ce qu'un gestionnaire ?* D'ailleurs, *Peut-on utiliser le modèle gestionnaire/agent dans le cadre du pair à pair ?* Ainsi, plutôt que de travailler sur une proposition d'organisation qui soit avant tout en adéquation avec notre modèle de l'information, nous avons préféré nous concentrer sur une proposition qui safisfasse les propriétés et contraintes induites par le modèle P2P.

Dans la section 4.3, nous avons présenté les différentes approches de gestion de réseaux existantes. Ces propositions ont été faites pour répondre à un problème précis. Notre cas d'étude concerne, en accord avec la classification des applications P2P exposée dans la section 4.5, les applications P2P qui sont (1) ouvertes à la gestion, (2) reposent sur un modèlé décentralisé ou hybride, (3) constituent un *overlay*, (4) sont déployées à l'échelle de l'Internet, et (5) comportent des pairs dont la présence et l'adressage est dynamique. Pour cette classe d'application, nous avons formalisé les contraintes induites sur la conception d'une infrastructure de gestion adaptée. Nous les présentons maintenant.

Le modèle P2P est utilisé dans des applications dont le cadre et le facteur d'échelle varient fortement. D'un côté, on peut trouver des services qui ne comptent que quelques participants situés sur un même réseau local ; c'est par exemple le cas d'applications de travail collaboratif. D'un autre, on trouve des applications qui fonctionnent grâce à des millions d'utilisateurs connectés par l'Internet, comme c'est le cas des services de partage de contenu. D'un point de vue de la gestion, parmi les deux environnements cités, le second est le plus contraint. En effet, il est composé d'éléments hétérogènes, situés sur des domaines administratifs différents et à présence intermittente. De plus, son nombre de participants pose le problème de passage à l'échelle de l'infrastructure qui le supervise. Dans un tel contexte, la gestion, effectuée par un opérateur humain, devient de moins en moins envisageable car la complexité de la tâche est trop importante [88]. Pour faire face à ce problème, l'auto-gestion [90], qui permet aux systèmes d'effectuer les opérations de manière automatique, semble être une solution incontournable. Dans notre cadre de travail, nous estimons qu'une infrastructure de gestion adaptée au modèle P2P doit être ouverte à l'auto-gestion.

Le second point qui nous a semblé important dans le travail de proposition d'une infrastructure de gestion, concerne l'intégration du plan de gestion dans le plan de service. Dans le contexte du pair à pair, fortement hétérogène, cette intégration nous semble cruciale pour obtenir une véritable cohésion entre le service géré et l'infrastructure qui le supervise. Elle simplifie la conception d'applications en permettant à des développeurs d'aborder parallèlement les aspects de mise en œuvre de services et de supervision. Cette intégration peut couvrir plusieurs domaines. Tout d'abord en termes de technologie, il nous semble important d'utiliser des langages et concepts similaires pour ces deux mondes. Ensuite, d'un point de vue de la communication,

la signalisation induite par la supervision doit être intégrée au trafic inhérent au service afin de pouvoir notamment contourner les problèmes de pare-feux. Enfin, elle doit permettre de réduire le coût inhérent à la gestion en simplifiant sa mise en œuvre.

Les motivations exposées dans les paragraphes précédents, relatives à l'ouverture vers l'auto-gestion et l'intégration du plan de gestion dans le plan de service, nous ont conduits à envisager la distribution de la fonction de gestion. Si l'infrastructure de gestion qui gère un service P2P est intégrée à celui-ci, elle doit respecter les contraintes de dynamicité et surtout d'équilibre de la charge et du trafic induites par le modèle P2P. L'utilisation d'un modèle de gestion centralisé intégré à une plate-forme P2P serait un non-sens car tous les efforts de distribution entrepris par le modèle P2P pour accroître le niveau de service seraient anéantis par l'ajout d'une centralisation contraire à ses caractéristiques.

Nous avons donc envisagé d'utiliser le modèle P2P comme modèle organisationnel du plan de gestion. Dans cette idée, chaque pair hébergerait un agent et un gestionnaire autonome qui pourrait, pour exercer sa fonction, consulter d'autres agents à proximité. Cette solution est élégante mais nous a semblé difficilement réalisable, car si la collecte d'information par le gestionnaire est simple, l'entreprise d'actions de contrôle semble beaucoup plus difficle à mettre en œuvre : si chaque pair est un gestionnaire autonome et peut effectuer des actions de gestion, plusieurs actions peuvent être accomplies en parallèle par des gestionnaires différents. Des problèmes de synchronisation des actions et d'influence mutuelle apparaissent alors, et les résoudre pourrait conduire à une complexification ainsi qu'une lourdeur excessive du plan de gestion. C'est pourquoi, nous avons choisi d'organiser les gestionnaires selon une hiérarchie telle qu'elle est définie par exemple dans [148]. Celle-ci permet de conserver l'avantage d'une autorité centrale, tout en distribuant la charge de gestion.

Pour conclure, la proposition de modèle d'organisation du plan de gestion que nous présentons dans la suite de ce chapitre est motivée par le dynamisme et le facteur d'échelle inhérents au modèle P2P, la fusion des plans de service et de gestion, et l'utilisation d'un modèle de gestion simple construit autour d'une hiérarchie de gestionnaires et d'agents.

7.1.2 Travaux relatifs

Il existe actuellement d'autres travaux qui proposent de construire une structure arborescente au-dessus d'un réseau P2P et ce, pour des raisons diverses. Dans cette section, nous présentons les principaux.

Un des domaines qui nécessitent la construction d'une topologie arborescente est celui de la diffusion de contenu. Cette classe d'applications se différencie de la gestion par l'utilisation qu'elle fait de l'arbre sur lequel elle repose. En effet, dans le cas de la gestion, les informations partent des feuilles et remontent à la racine (arbre $n \rightarrow 1$). Par contre dans le cas de la diffusion de contenu, les informations partent de la racine pour aller vers les feuilles (arbre $1 \rightarrow n$).

Le travail de El-Ansaryvet et al. [53] propose une manière de construire un arbre de diffusion, de type *broadcast*, sur une infrastructure P2P. Conceptuellement, la contribution est applicable à n'importe quelle infrastructure P2P qui repose sur une DHT. Néanmoins, dans la présentation de leur travail, les auteurs expliquent le principe de construction de leur arbre sur l'infrastructure Chord [153] et utilisent en particulier les *fingers*[1] qui sont un concept propre à cette infrastructure. Ce travail semble donc difficilement adaptable à une autre infrastructure fondée sur une DHT.

Une autre proposition de construction d'arbre de diffusion, cette fois de type *multicast*, nous

[1] des entrées particulières de la table de routage d'un nœud

semble remarquable. Elle est faite par Ratnasamy et al. [125] qui ont travaillé sur la DHT CAN [124], et présentent un principe de construction d'arbre appliqué à cette dernière infrastructure. Pour ce faire, les auteurs proposent de créer une sous-communauté par groupe de *multicast* et utilisent un algorithme d'inondation amélioré pour effectuer la diffusion de contenu. Cette fois, la généricité du travail est effective car, si l'exemple d'application est fondé sur CAN, le principe de construction et de diffusion ne repose pas sur un concept propre à la DHT qui le supporte.

D'un point de vue théorique, il existe actuellement deux travaux qui sont assez semblables au nôtre. Le premier [17], propose une manière de construire un arbre binaire au-dessus d'une DHT. Le principe de construction et les résultats de simulations sont très intéressants. Par exemple, les auteurs montrent que leur infrastructure permet de réduire le coût d'insertion des nœuds qui s'exprime typiquement en $log^2(N)$ [153, 137] à $log(N)$. Dans notre contexte d'application relatif à la gestion de réseaux, la construction d'un arbre binaire n'est pas la plus adaptée, car l'arbre résultant de cette construction va être très profond et, par exemple, dans le cas du déclenchement d'une alarme sur un agent situé sur une feuille de l'arbre, le nombre de sauts à effectuer pour atteindre la racine sera beaucoup trop important. En outre, ce travail est un cas particulier de la proposition que nous présentons par la suite.

Le travail le plus proche du nôtre est exposé dans [99]. Son objectif est la construction d'un arbre d'agrégation qui puisse être mis en œuvre au-dessus d'une quelconque DHT. L'utilisation de cet arbre est destinée principalement à des opérations de monitorage. Le principe de construction est centré sur la définition d'une fonction *Parent* qui, pour chaque identifiant d'un nœud, retourne celui de son père dans l'arbre. Le choix de cette fonction est primordial car il doit assurer l'unicité du parent de chaque nœud, mais aussi l'équilibre de l'arbre. Notre travail est très similaire à cette proposition car notre organisation hiérarchique du plan de gestion permet aussi de déployer des fonctions d'agrégation. Néanmoins, notre travail présente une portée plus large car il traite aussi de la définition des rôles de gestionnaire et d'agent.

7.1.3 Objectifs

La proposition d'architecture de gestion pour les réseaux et services P2P que nous présentons dans ce chapitre permet d'atteindre plusieurs objectifs qui sont :

La distribution forte de la fonction de gestionnaire : Le modèle P2P est un modèle distribué qui ne présente, dans le cas du modèle pur, aucun point singulier de centralisation. D'un point de vue de la gestion, nous désirons approcher ce même degré de distribution en permettant à chaque pair d'une communauté d'accéder aux fonctions d'agent et de gestionnaire.

L'équilibre de la charge confiée aux gestionnaires : Notre proposition d'architecture de gestion repose sur un modèle hiérarchique. Pour satisfaire la contrainte de distribution du modèle P2P, nous voulons que notre structure soit équilibrée, de sorte qu'aucun pair ne joue un rôle prépondérant et soit de fait un point central de fautes.

La possiblité de choisir les gestionnaires : Nous avons établi que dans notre structure, chaque pair puisse agir comme agent et gestionnaire. Cependant, nous désirons pouvoir choisir les pairs qui accèdent à la fonction de gestionnaire car ce rôle est sensible, et ce, pour plusieurs raisons. Tout d'abord, il confère au pair qui l'héberge une autorité particulière ; les pairs gestionnaires doivent être des pairs bien intentionnés et en lesquels la communauté peut avoir confiance. Ensuite, les fonctions de gestion exécutées par les pairs gestionnaires induisent une surcharge de travail par rapport aux fonctions d'agent et de pair. Les pairs gestionnaires doivent donc présenter les ressources nécessaires en termes de puissance de

calcul, espace de stockage et mémoire de travail qui permettent au pair de garantir le bon déroulement des fonctions de gestion et du service auquel il participe. Enfin, dans un souci de stabilité et de pérennité de l'architecture de gestion et des données qu'elle manipule, les pairs gestionnaires doivent être stables, en comparaison avec les autres pairs de leur communauté. Pour prendre en compte toutes ces contraintes, nous proposons de munir chaque pair d'une métrique, appelée *poids*, qui représente sa qualité. Les critères définis précédemment sont des exemples qui peuvent être pris en compte pour le calcul de cette métrique, mais tout autre critère, défini dans le cadre du service géré peut être considéré. La structure doit s'organiser de sorte que les gestionnaires présentant le poids le plus important soient prioritaires pour l'accès à la fonction de gestionnaire, et que le pair de poids le plus fort soit le gestionnaire racine de la structure.

La maîtrise de la profondeur : Afin de garantir un niveau de performance minimum à notre architecture de gestion, nous voulons pouvoir maîtriser la profondeur de notre architecture qui représente le nombre de sauts maximum qui séparent le gestionnaire racine d'un quelconque agent. Ce critère est important car il dépend étroitement de la fonction de gestion à laquelle on s'intéresse. Par exemple, dans le cas de la gestion d'alarmes déclenchées par des agents, on veut pouvoir notifier le gestionnaire racine le plus rapidement possible, d'où la nécessité de minimiser la profondeur de l'arbre. Cependant, ce tassement de la structure va avoir pour conséquence directe l'augmentation de la charge des gestionnaires, qui, étant peu nombreux, vont devoir prendre en charge un grand nombre d'agents ou de sous-gestionnaires. A l'inverse, dans le cas où des pairs aux ressources limitées, comme des terminaux mobiles de type téléphone portable ou assistant personnel, utilisent cette structure, la charge attribuée aux gestionnaires doit être minimale. Ces derniers ne peuvent alors gérer qu'un nombre très limité d'agents ou de sous-gestionnaires. Cette minimisation de la charge se répercute directement sur la profondeur de l'arbre de gestion qui s'en trouve augmentée.

Un fonctionnement distribué : La construction et la maintenance de notre structure doit s'effectuer de manière distribuée. Chaque pair est muni d'un code identique qui, lorsqu'il est exécuté, conduit à la construction et au maintien de la structure hiérarchique. Aucun pair ne doit jouer de rôle prédondérant ou central. Cet objectif permet de proposer une construction d'une structure hiérarchique par un fonctionnement pair à pair et donc d'adapter au mieux la construction et le maintien de l'architecture de gestion à l'infrastructure de service sous-jacente.

7.2 Notre proposition

Notre proposition repose sur un principe de construction très similaire à l'algorithme de Plaxon et al. [122]. Dans un premier temps, nous exposons l'algorithme général qui permet d'établir cette structure de manière centralisée. Ensuite, nous montrons la manière dont nous distribuons cet algorithme afin de le rendre indépendamment exécutable par chaque pair.

7.2.1 Principe général de construction

La construction de notre structure hiérarchique repose sur cinq axiomes que nous définissons ci-après :

1. Chaque pair est un agent et éventuellement un gestionnaire ;
2. Chaque feuille de l'arbre représente un agent ;

3. Chaque nœud de l'arbre situé à un niveau supérieur aux feuilles représente un gestionnaire ;
4. Chaque nœud est identifié à l'aide de l'identifiant attribué au niveau de l'infrastructure sous-jacente ;
5. Chaque nœud possède un poids qui représente sa qualité en vue de son accession potentielle au rôle de gestionnaire.

Etant donné ces axiomes, la manière dont nous construisons notre arbre est décrite dans les définitions suivantes :

Définition 8 *Chaque gestionnaire de niveau L est responsable des nœuds, agents et gestionnaires, de niveau $L+1$ dont l'identifiant présente un préfixe commun de L digits.*

Définition 9 *Les gestionnaires sont désignés par un processus d'élection qui stipule qu'entre deux ou plusieurs nœuds candidats à la fonction de gestionnaire d'un niveau, celui de poids le plus fort est élu.*

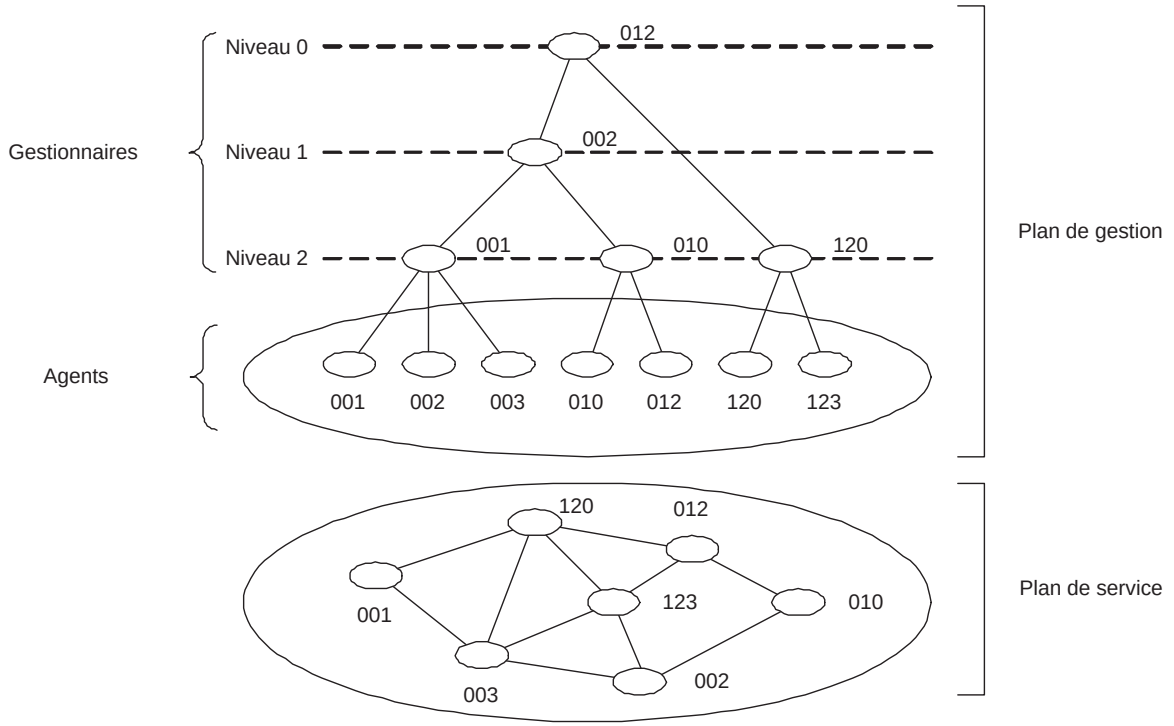


FIG. 7.1 – Exemple d'application de notre hiérarchie de gestion à une communauté de pairs quelconque

La figure 7.1 illustre ce principe de construction. Elle représente une communauté composée de 7 nœuds qui s'organisent selon une architecture hiérarchique qui suit les définitions 8 et 9. On constate tout d'abord que l'arbre résultant compte 7 feuilles qui représentent les agents hébergés sur chaque pair. Ensuite, chaque gestionnaire inscrit dans l'arbre possède un préfixe commun avec ses agents en accord avec son niveau. Par exemple, au niveau 2, le gestionnaire 001 possède le préfixe 00, long de 2 digits, avec ses agents qui sont 001, 002 et 003. De même, au niveau 1, le gestionnaire 002 possède le préfixe commun 0, composé d'un seul digit, avec ses

sous-gestionnaires qui sont 001 et 010. Enfin, on voit que chaque nœud gestionnaire n'apparaît qu'une seule fois dans la hiérarchie. Pour un niveau L donné, le choix du gestionnaire de niveau supérieur n'est pas fait parmi les nœuds de ce niveau, mais parmi les nœuds qui ne sont pas encore gestionnaires et qui remplissement les conditions inscrites dans les définitions 8 et 9.

7.2.2 Définition formelle

Etant donné le principe de construction présenté ci-dessus, nous en avons établi la définition formelle de notre algorithme qui s'exprime comme un invariant du premier ordre. Pour ce faire, nous introduisons trois paramètres qui sont :

B **:** La base numérique utilisée pour la représentation des digits des identifiants ;

D **:** Le nombre de digits des identifiants des nœuds ;

N **:** Le nombre de nœuds dans la communauté.

Ensuite, nous définissons les variables et ensembles suivants :

d_i **:** Le digit de rang i d'un identifiant avec $0 \leq d_i < B$ et $1 \leq i \leq D$;

$d_1 \dots d_D$ **:** L'identifiant d'un nœud ;

L **:** Le nombre de digits d'un préfixe commun ;

λ **:** L'ensemble des niveaux présents dans la structure, c'est à dire, l'ensemble des L ;

$Q_{d_1 \dots d_L}$ **:** L'ensemble des nœuds de préfixe commun $d_1 \dots d_L$;

P_L **:** L'ensemble d'ensemble de nœuds $Q_{d_1 \dots d_L}$ qui possèdent un préfixe commun de L digits ;

G **:** L'ensemble des nœuds gestionnaires.

Nous définissons l'invariant présenté dans l'algorithme 7.1 qui est vrai pour l'ensemble des nœuds inscrits dans notre structure hiérarchique. Cet algorithme est formalisé à l'aide d'une notation orientée objet et il utilise deux méthodes particulières qui sont : $n.Prefix(L)$ qui retourne les L premiers digits de l'identifiant du nœud n et $n.Weight()$ qui retourne le poids du nœud n.

```
TREE DEFINITION ()
1   P_{-1} ← {∅}
2   ∀L ∈ λ
3     ∀n ∈ N
4       Q_{d1...dL} ← Q_{d1...dL} ∪ {n | n.PREFIX(L) = d1 ... dL}
5     P_L ← {Q_{d1...dL} | Q_{d1...dL} ≠ {∅}, 0 ≤ di < B, 1 ≤ i ≤ L}
6     ∀P ∈ P_L \ {P_L ∩ P_{L-1}}
7       G ← G ∪ {n | n ∈ P, n ∉ G, n.WEIGHT() = max(p.WEIGHT(), p ∈ P)}
```

TAB. 7.1 – Algorithme de construction de la structure hiérarchique de gestion

Cet algorithme construit la structure niveau après niveau, en commençant par la racine (lignes 1 et 2). A chaque niveau, il détermine les différents ensembles $Q_{d_1 \dots d_L}$ de pairs qui possèdent L digits en commun (lignes 3 et 4). Chacun de ces ensembles non vides est inscrit dans l'ensemble P_L (ligne 5). Ensuite, pour chaque ensemble de nœuds qui n'a pas été considéré au niveau précédent (ligne 6), on choisit un gestionnaire comme étant le nœud de poids le plus grand et qui ne soit pas déjà gestionnaire (ligne 7). L'algorithme se répète pour chaque niveau de l'arbre.

7.3 Distribution de l'algorithme

L'algorithme de construction de l'architecture de gestion, tel qu'il a été présenté dans la section précédente est de nature centralisée. Il pourrait être exécuté par un serveur central ayant pour rôle d'organiser les pairs en accord avec les définitions 8 et 9 et d'assurer le maintien de la structure selon un état cohérent. Cependant, un des objectifs fixés dans ce travail réside dans la capacité des pairs à organiser la structure de gestion par eux-mêmes. Nous proposons donc de distribuer cet algorithme afin qu'il puisse être exécuté de manière autonome et indépendante par chacun des pairs participant à une communauté. Dans cette section, nous présentons les trois opérations que notre structure supporte pour assurer son bon fonctionnement. Celles-ci sont relatives à l'insertion d'un nœud dans l'arbre, son retrait, et la maintenance de la structure.

7.3.1 Insertion d'un nœud

Le processus d'insertion d'un nœud a pour but d'ajouter un nouveau nœud en le plaçant correctement dans la structure. En d'autres termes, il consiste à rechercher le gestionnaire qui possède le préfixe commun le plus long avec le nouveau nœud. Néanmoins, dans les DHTs actuelles, cette recherche n'est pas simple, car ces infrastructures ne supportent pas directement cette opération. Nous avons envisagé plusieurs manières qui permettent d'accomplir cette tâche : la première consiste à utiliser une méthode d'approches successives : un nouveau nœud cherche successivement les gestionnaires présentant un préfixe commun de $D-1$ digits, puis $D-2$, ..., jusqu'à l'obtention d'une réponse. Cette méthode est coûteuse puisqu'elle peut requérir jusqu'à D messages, chaque message étant routé en $\log(N)$ sauts.

C'est pourquoi nous proposons d'utiliser une autre méthode : lorsqu'un nœud rejoint la structure, il génère une requête dont l'identifiant de destination est aléatoire. D'après les propriétés des DHTs, le nœud d'identifiant le plus proche va alors lui répondre. Le nœud arrivant va alors demander à ce dernier l'identité de son gestionnaire et s'enregistrer comme nouvel agent de ce gestionnaire. Ainsi, le nœud est inséré dans l'arbre, mais à une place qui ne satisfait pas les définitions 8 et 9. C'est le processus de maintenance du gestionnaire qui va détecter ce mauvais placement et tranférer la gestion de ce nœud mal placé à son gestionnaire de niveau supérieur. Par déplacements successifs, le nouveau nœud va être placé correctement dans la structure.

7.3.2 Maintenance de la structure

Le maintien de la structure dans un état cohérent est assuré par un processus de maintenance qui est exécuté dans deux contextes : lorsqu'un nœud arrive, pour s'assurer de son bon placement dans la structure et à intervalles réguliers, pour vérifier que tous les nœuds référencés sont effectivement présents et joignables. Dans le cas où une incohérence est détectée, le processus de maintenance réorganise l'arbre de manière à respecter les contraintes imposées par les définitions 8 et 9.

Ainsi, les différentes opérations effectuées par le processus de maintenance sont les suivantes :

1. **Vérification de présence :** Pour un gestionnaire de niveau L, cela consiste à vérifier la présence de chacun de ses fils et de son père. Pour ce faire, une requête est envoyée à chaque fils. Si un fils ne répond pas à cette requête, il est retiré de la liste des fils dont le gestionnaire à la charge[2]. Ensuite, puisqu'un gestionnaire envoit régulièrement des requêtes de vérification de présence à ses fils, il attend d'en recevoir de la part de son père. Si jamais

[2]Cette vérification part de l'hypothèse que les pairs sont atteignables de manière fiable. Dans le cas contraire, il faudrait envoyer plusieurs messages avant de considérer un pair comme injoignable.

aucune requête n'est reçue, le gestionnaire est considéré comme orphelin et est coupé de la structure. Il va alors utiliser le processus d'insertion pour rejoindre à nouveau l'arbre.

2. **Vérification des préfixes :** Pour un gestionnaire de niveau L, cela consiste à vérifier que chacun de ses fils possède un identifiant qui satisfait la définition 8. Deux cas de réorganisation sont possibles :
   - **Préfixes trop courts :** Si un fils possède un préfixe commun qui est inférieur à L digits, le gestionnaire transfère ce fils à son père.
   - **Préfixes trop longs :** Si deux ou plusieurs fils possèdent un préfixe commun plus long que L, un regroupement est possible et deux cas sont envisageables :
     - **Fils agents et gestionnaires :** Si l'ensemble des fils regroupables est un mélange de gestionnaires et d'agents, le gestionnaire de poids le plus élevé devient gestionnaire des autres éléments de cet ensemble.
     - **Fils identiques :** Si l'ensemble des fils regroupables est composé exclusivement d'agents ou de gestionnaires, le fils de poids le plus fort devient le gestionnaire des autres.
3. **Vérification des poids :** Pour un gestionnaire de niveau L, cela consiste à vérifier qu'aucun fils qui ne soit pas déjà gestionnaire possède un poids plus élevé que celui du gestionnaire. Si jamais ce cas arrivait, le fils de poids le plus fort prendrait alors la place du gestionnaire actuel qui perdrait sa fonction de gestionnaire et redeviendrait un simple agent.

7.3.3 Départ d'un nœud

Le départ d'un nœud de la structure peut se produire de deux manières :

Départ annoncé : Le nœud quitte la structure. Son agent informe son gestionnaire de son départ et dans le cas où le nœud est gestionnaire, ce dernier informe son père qui va alors prendre en charge temporairement chacun de ses fils. Ensuite, le processus de maintenance réorganisera cette partie de l'arbre pour l'amener dans un état cohérent.

Départ non annoncé : Le nœud quitte la structure de manière brutale. Dans ce cas, pour les deux fonctions de gestionnaire et d'agent, le processus de maintenance, par le biais de sa vérification de présence, va détecter ce départ, retirer le nœud des endroits où il était référencé et éventuellement réorganiser l'arbre pour satisfaire cette nouvelle situation.

De par l'utilisation des trois opérations d'insertion, de maintenance et de départ, nous sommes capables de construire notre structure hiérarchique de gestion pour le P2P et de la maintenir d'une manière totalement distribuée et en dépit des allers et venues des nœuds. Dans la suite de ce chapitre, nous montrons la manière dont nous avons déployé et éprouvé cette proposition à travers son implantation dans une DHT existante.

7.4 Implantation et évaluation

Nous avons réalisé un prototype de notre architecture [175] que nous avons déployé dans FreePastry[3], une implémentation Java de Pastry [137]. Dans cette section, nous présentons tout d'abord la manière dont nous avons implanté ce prototype. Ensuite, nous exposons les résultats de premiers tests que nous avons effectués et qui caractérisent son fontionnement.

[3] freepastry.rice.edu

7.4.1 Architecture des nœuds

L'architecture fonctionnelle de notre prototype, tel qu'il est implanté dans les nœuds Pastry, est représentée sur la figure 7.2. Cette figure montre que chaque nœud contient trois composants principaux : un agent, un gestionnaire, qui est activé dans le cas où le nœud accède à cette fonction, et un gestionnaire d'état, appelé *State manager*. Ce dernier a pour rôle de stocker et rendre accessible l'ensemble des données inhérentes à la situation du nœud dans l'arbre. Par exemple, pour l'agent, le gestionnaire d'état référence l'identifiant de son gestionnaire, et pour le gestionnaire, il héberge la liste des agents.

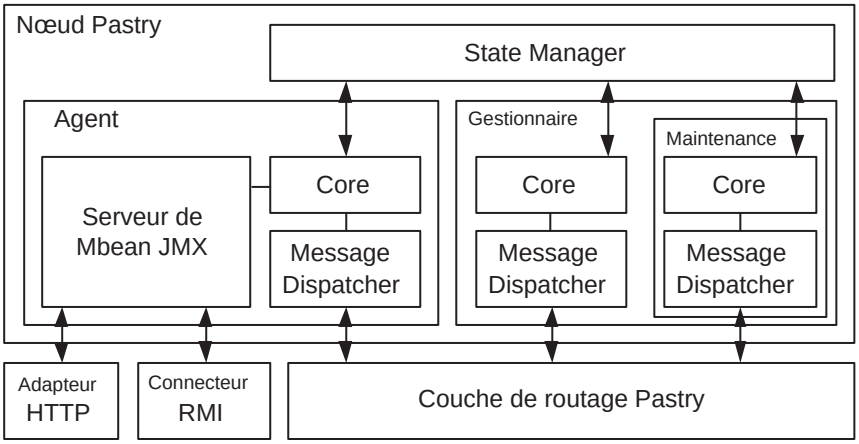


FIG. 7.2 – Architecture fonctionnelle des nœuds

L'agent d'un nœud se scinde en deux blocs fonctionnels. Le premier est un serveur de MBeans JMX[4] [101, 79] qui héberge des MBeans standards issus de l'instrumentation de Pastry, présentée ci-après. Le second concerne le fonctionnement de l'agent à proprement parler, et a pour but de recevoir et aiguiller les requêtes et effectuer les opérations de gestion indiquées par le gestionnaire dont l'agent dépend.

Le gestionnaire hébergé dans un nœud est composé lui aussi de deux blocs fonctionnels : le premier est lié au fonctionnement du gestionnaire et, comme pour l'agent, il effectue le traitement des requêtes et s'occupe des opérations de gestion inhérentes à la fonction de gestionnaire, comme l'agrégation de données de gestion issues de ses fils, le déclenchement d'une alarme sur des données traitées, ... Le second bloc fonctionnel du gestionnaire est relatif à la maintenance de l'arbre. Lorsqu'un nœud devient gestionnaire, il instancie et exécute un processus de maintenance. Ce dernier va alors effectuer les opérations décrites dans la section 7.3.2.

Pour terminer, concernant la manière dont les nœuds communiquent, deux moyens sont utilisés. Tout d'abord, pour construire et maintenir la structure arborescente, les messages sont acheminés par le biais de l'infrastructure de routage de Pastry. Comme conséquence, chaque message échangé entre deux nœuds voisins au niveau de la topologie induite par notre arbre de gestion, nécessite $\log(N)$ sauts. Par contre, les opérations de gestion sont effectuées à travers des agents JMX qui communiquent directement par le protocole RMI[5].

[4]Java Management eXtension - http ://java.sun.com/products/JavaManagement/
[5]Remote Method Invocation

7.4.2 Instrumentation des nœuds

Afin de nourrir de données de gestion les agents de nœuds Pastry, nous avons instrumenté ces derniers. Chaque agent JMX héberge ainsi des instances d'objets CIM qui suivent une vue locale des modèles de l'information présentés dans les chapitres 5 et 6. Pour un nœud donné, nous collectons et enregistrons des informations inhérentes aux tables de routage et aux services de localisation et de maintenance de Pastry.

De manière annexe au travail présenté dans ce chapitre et dans le but de valider l'instrumentation des nœuds Pastry, nous avons conçu un mini-gestionnaire centralisé, ainsi qu'une mini-application de visualisation de topologie. Cette infrastructure est représentée sur la figure 7.3. Le gestionnaire, hébergé sur un nœud Pastry, a pour but d'agréger les données de gestion locales stockées dans les agents. Il fournit ainsi une vue synthétique et globale de l'ensemble de la communauté. Cette vue globale est ensuite utilisée par une application graphique qui permet de visualiser la topologie construite par les nœuds Pastry instrumentés. La communication entre les différentes entités est assurée par le biais du protocole RMI.

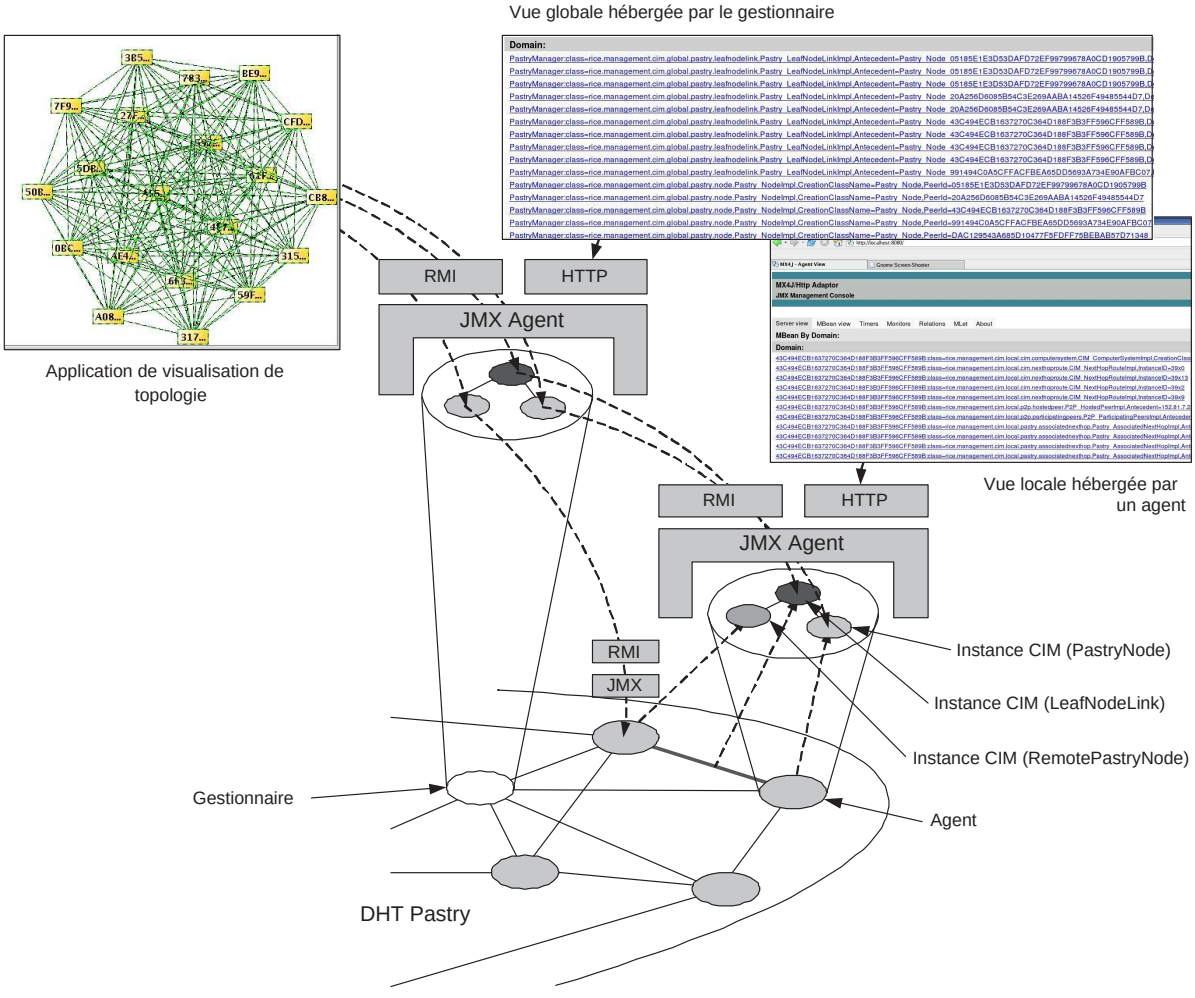


FIG. 7.3 – Mise en œuvre d'une mini-application de visualisation de topologie

La topologie visualisée dans la figure 7.3 illustre le cas d'une communauté contenant vingt

nœuds. Comme critère d'adjacence topologique, nous avons considéré le *leaf set* maintenu par chaque nœud Pastry. Pour rappel, chaque nœud maintient une liste de nœuds, appelée *leaf set*, qui représente les nœuds les plus proches, dans l'espace circulaire des identifiants, du nœud considéré. Sur la figure 7.3 on voit la manière dont un agent, hébergé par un pair, et un gestionnaire vont représenter cette association topologique. Du point de vue de l'agent, l'association à un nœud *leaf* distant est modélisée par trois classes : la classe PST_PastryNode représente le nœud qui héberge l'agent ; la classe PST_RemotePastryNode représente un nœud distant et l'association PST_LeafNodeLink signifie que le nœud distant est un nœud qui appartient au *leaf set* du nœud local.

Du point de vue du gestionnaire, toutes les instances de nœuds, de type PST_PastryNode, de la communauté sont connues et stockées. Donc le gestionnaire peut mettre en relation directement ces classes par le biais de la classe d'association PST_LeafNodeLink. Les agents possèdent donc une vue locale faisant intervenir des objets CIM représentant des nœuds distants alors que le gestionnaire possède une vue globale et héberge l'ensemble des instances qui la composent.

Enfin, par le biais du protocole RMI, l'application de visualisation va pouvoir directement lire les informations hébergées par le gestionnaire et dessiner la topologie résultante. Dans l'implantation FreePastry de Pastry, la taille du *leafset* de chaque nœud est de vingt entrées. La topologie représentée par l'application de gestion contient aussi vingt nœuds, ce qui explique que le graphe de cette représentation topologique soit complet.

Cette étape, annexe au travail d'organisation du plan de gestion, a eu pour rôle de montrer que l'instrumentation des nœuds était valide et exploitable par un gestionnaire qui est ici de nature centralisée, mais qui pourrait être distribué, ou hiérarchique. Nous avons montré que, du point de vue de la gestion, nous étions capable d'agréger une partie du modèle distribué dans les agents des nœuds.

7.4.3 Evaluation de la proposition

Afin de valider notre travail, nous avons effectué un premier test qui consiste à évaluer le coût de construction de l'architecture en fonction du nombre de nœuds.

Conditions de test

Les tests que nous avons conduits se sont déroulés dans les conditions suivantes : Nous avons considéré une communauté Pastry comportant de 1 à 20 nœuds et, pour chaque cas, nous avons évalué le nombre total de messages nécessaires à la construction et au maintien de la structure pour tous les nœuds mais aussi pour chaque nœud. Chaque test a été répété dix foix. D'un point de vue temporel, le taux d'arrivée des nœuds dans la communauté est fixé à un nœud par minute. Pour chaque nœud, le processus de maintenance s'exécute toutes les quinze secondes. Enfin, pour chaque requête envoyée, l'attente d'une réponse expire au bout de trente secondes.

Pour évaluer la surcharge induite par la prise en compte du poids de chaque nœud dans l'organisation hiérarchique, nous avons effectué ce test deux fois. La première fois, nous avons considéré le poids des nœuds qui prend une valeur aléatoire comprise entre 0 et 255. La seconde, nous n'avons pas considéré le poids des nœuds.

Analyse des résultats

Les résultats de notre test sont représentés sur les figures 7.4 et 7.5. Sur chacune des deux figures, nous avons superposé les résultats du test avec et sans considération du poids des nœuds. La figure 7.4 s'intéresse au coût de construction global, et représente la moyenne et l'écart type

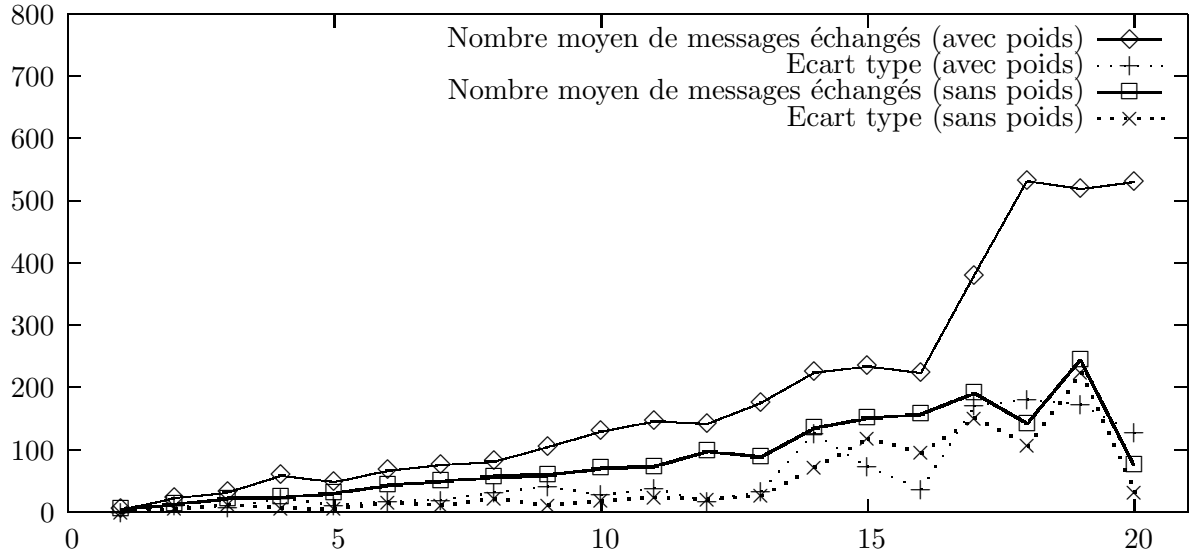


FIG. 7.4 – Evaluation du coût de construction global exprimé en nombre de messages en fonction du nombre de nœuds

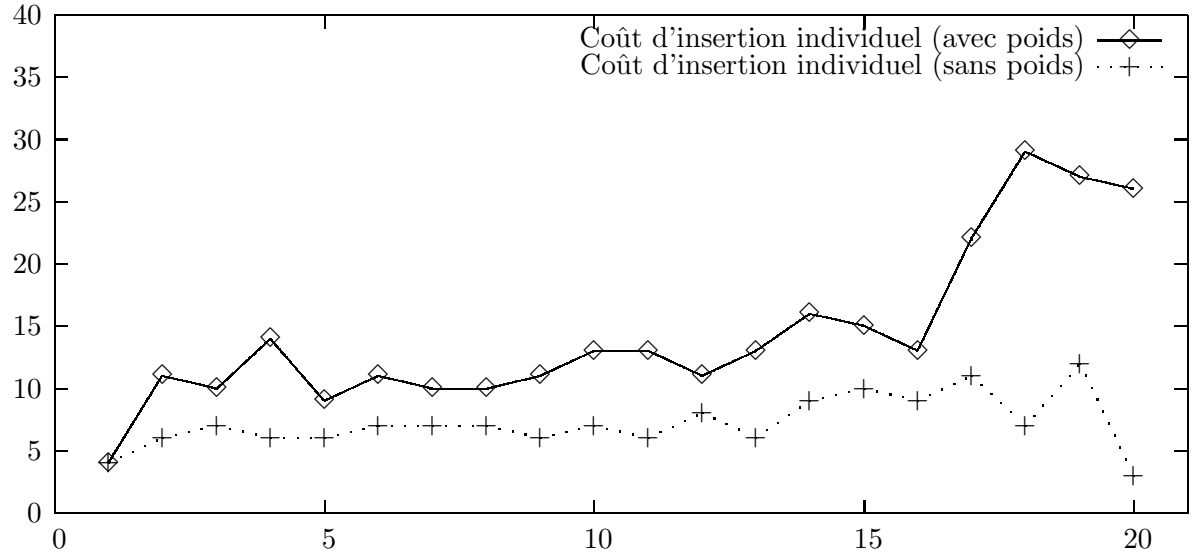


FIG. 7.5 – Evaluation du coût d'insertion d'un nœud exprimé en nombre de messages en fonction du nombre de nœuds

du nombre de messages échangés pour construire et maintenir la structure alors que la figure 7.5 représente la moyenne du nombre de messages échangés pour un nœud.

D'une manière générale, on voit que jusqu'à 16 nœuds, le coût de construction évolue de manière linéaire. A partir de 16 nœuds, le nombre de messages augmente brusquement et semble croître à nouveau linéairement. Cette augmentation brutale s'explique de la manière suivante : les identifiants des nœuds Pastry sont exprimés en base hexadécimale et sont attribués de manière aléatoire. D'après la définition 8, un gestionnaire va donc prendre en charge 16 fils au maximum. Pour une communauté comportant un maximum de 16 nœuds, statistiquement, chaque nœud va présenter un identifiant dont le premier digit diffère, conduisant ainsi à la construction d'un arbre sur deux niveaux : un niveau contenant un gestionnaire racine, et un second contenant les agents. Par contre, à partir de 17 nœuds, la structure va obligatoirement comporter un niveau supplémentaire et le coût de construction, dû à un nombre beaucoup plus grand de réorganisations possibles, va augmenter significativement. Cette augmentation est clairement mise en évidence sur la figure 7.4.

La seconde constatation que l'on peut faire au vu des résultats de ce test concerne la prise en compte du poids des nœuds dans la construction de l'arbre. Il apparaît que le poids des nœuds augmente considérablement le coût de construction et de maintien de la structure. En effet, jusqu'à 16 nœuds, le coût de construction individuel moyen, avec le poids, est de 12 messages, alors que sans le poids, il est de 7,3 messages. De plus, cet écart s'accroît fortement à partir de 17 nœuds, lorsque l'arbre compte deux niveaux de gestionnaires. Cette augmentation, d'autant plus forte que l'on considère le poids, s'explique de la façon suivante : lorsqu'un nœud accède à la fonction de gestionnaire, en fonction de son poids, il va se rapprocher de la racine. Imaginons qu'une structure hiérarchique stable et comportant 20 nœuds soit établie. Si un nœud de poids plus élevé que la racine rejoint la communauté, il va prendre la place d'un gestionnaire de niveau intermédiaire existant et ensuite celle du nœud racine. Cette réorganisation est très coûteuse car les gestionnaires ayant perdu leur fonction vont se retrouver de simples agents, être réélus gestionnaires, et reprendre une place qui correspond à la nouvelle organisation. De plus, on comprend que plus l'arbre comporte de niveaux, et plus cette réorganisation est importante.

La dernière remarque que nous formulons formalise les explications fournies dans les deux précédentes : plus le nombre de nœuds augmente, plus l'écart-type de la moyenne du nombre de messages nécessaire à la construction et au maintien de l'arbre augmente. Comme nous l'avons expliqué précédemment, ceci est dû à l'augmentation des scénarii possibles dans les étapes de construction. Cette augmentation montre que pour avoir des résultats très probants, le nombre de répétitions du test doit augmenter avec le nombre de nœuds.

7.5 Synthèse

La supervision des réseaux et services pair à pair ne peux pas être faite à l'aide des architectures classiques de gestion. Le facteur d'échelle, la dynamicité et l'hétérogénéïté des composants rendent cette tâche de plus en plus complexe pour un opérateur humain et l'auto-gestion semble être une solution incontournable. Ensuite, la simplification et l'efficacité des opérations de gestion passe par leur intégration au plan de service. Ces deux réflexions nous ont menés vers l'utilisation d'une architecture hiérarchique distribuée comme modèle d'organisation du plan de gestion. Celle-ci permet d'organiser le plan de gestion selon une hiérarchie de gestionnaires qui se répartissent équitablement les fonctions de gestion. De cette manière, l'infrastructure de gestion tend à fonctionner selon un modèle pair à pair, dans le sens où chaque pair est potentiellement agent et gestionnaire à la fois, tout en conservant les avantages d'une autorité centrale.

Nous avons proposé un algorithme central qui implante ce fonctionnement et nous l'avons ensuite distribué de manière à ce qu'il puisse être exécuté indépendamment par des pairs participant à une communauté. Pour valider cette proposition, nous avons implanté un prototype sur la DHT Pastry, pour laquelle nous avons aussi réalisé une instrumentation. Afin de tester notre architecture, nous avons conduit un test qui évalue le coût de construction et de maintenance de notre structure en fonction du nombre de nœuds. Ce test a révélé le bon comportement de notre algorithme pour un petit nombre de nœuds. En outre, il a mis en évidence le surcoût induit par la considération d'une métrique qui évalue la qualité des nœuds en vue de leur accession à une fonction de gestionnaire.

Troisième partie

Expérimentations

Chapitre 8

Développement d'une plate-forme de supervision pour Jxta

Sommaire

Dans ce chapitre, nous présentons la plate-forme de supervision pour Jxta que nous avons développée. Deux stages ont contribué à ce développement. Rizi Mohanti [114] a travaillé sur l'intégration d'un agent JMX dans la plate-forme Jxta, et la spécialisation de notre modèle de l'information générique pour cette plate-forme. Julien Braure [16] a implanté ce modèle de l'information, développé un gestionnaire qui agrège les données de gestion issues des agents, et ajouté à l'ensemble une application de supervision qui offre une interface graphique pour un administrateur de la plate-forme. Nous présentons maintenant chacune de ces réalisations.

8.1 Introduction

L'objectif essentiel de ce travail de réalisation est la validation du modèle de l'information pour les réseaux et services P2P présenté dans le chapitre 5. Afin de prouver que notre proposition de modèle de l'information est valide, nous estimons qu'il faut pouvoir l'instancier sur différentes plates-formes. A ce jour, nous l'avons spécialisé pour trois infrastructures P2P qui sont : Chord, Pastry et Jxta. Dans la section 7.4.2, nous avons présenté la mise en œuvre de ce modèle dans le cadre de l'instrumentation de Pastry. Le déploiement sur Jxta constitue une deuxième implantation du modèle qui confirme sa validité.

Le second objectif visé par cette réalisation concerne la mise en évidence du service offert par une plate-forme de supervision dans le contexte du modèle P2P. Comme nous l'avons exprimé dans le chapitre 4, le modèle P2P présente des caractéristiques d'auto-organisation qui ne sont pas synonymes d'auto-gestion. La plate-forme de supervision que nous proposons met clairement en évidence cette différence et montre les nouvelles fonctionnalités qu'elle confère à Jxta.

La figure 8.1 présente une vue générale de notre infrastructure de supervision. Nous avons intégré dans chaque pair Jxta un agent de gestion construit autour d'un agent JMX qui héberge des objets gérés et qui est accessible par le biais du protocole RMI. Un gestionnaire, exécuté lui aussi sur un pair, est élu pour chaque *Peergroup*. Son rôle est d'agréger les différentes données de gestion issues des agents et d'en fournir une vue synthétique qui suit le modèle de l'information que nous avons conçu. Les objets gérés qu'il génère sont aussi enregistrés dans un agent JMX et sont rendus accessibles par RMI. Enfin, une application de gestion qui interagit avec le gestionnaire offre à un utilisateur une véritable interface de supervision pour la plate-forme.

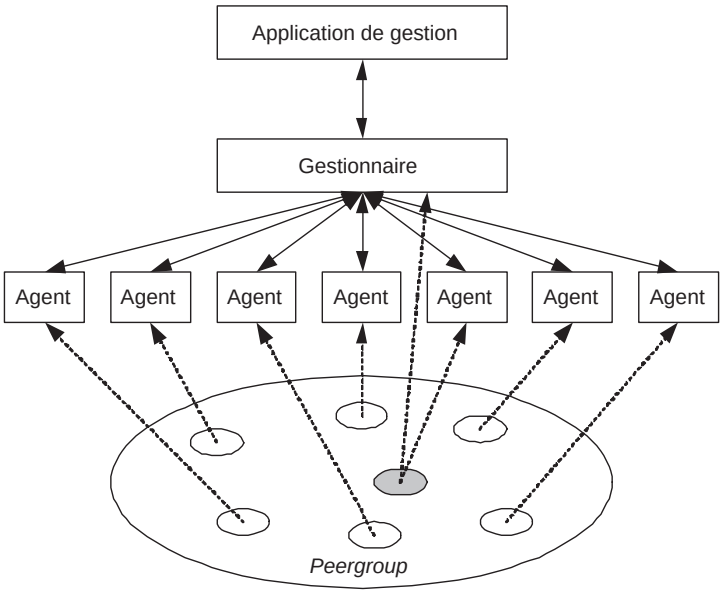


FIG. 8.1 – Vue générale de l'infrastructure de gestion déployée sur Jxta

L'ensemble de cette réalisation repose sur le langage Java. Nous avons choisi d'utiliser ce langage pour plusieurs raisons. La première réside dans sa simplicité et sa portabilité. Etant donné la nature hétérogène des éléments constituant une application P2P, nous estimons qu'un langage portable est nécessaire pour garantir le fonctionnement de notre infrastructure quel que soit son contexte d'exécution. Ensuite, nous estimons que l'utilisation d'un seul langage de programma-

tion pour la mise en œuvre d'un service et de sa supervision est intéressante. Cette intégration permet entre autres d'éviter les traductions dues à des changements de technologies et simplifie le travail des développeurs de services qui voudraient ajouter des fonctions de supervision à leur applications. Les outils que nous avons utilisés sont ainsi un ensemble de bibliothèques Java que nous mentionnons dans le tableau 8.1.

Outil	Version
Java	JDK v1.4.2_08
Jxta	Java Implantation v2.3.3
JMX	Reference Implementation v1.2
JGraph	v5.2
Log4j	v1.2

TAB. 8.1 – Récapitulatif des outils utilisés

Remarque : La conception et la réalisation de la plate-forme de supervision pour Jxta présentée dans ce chapitre s'inscrit principalement dans le cadre du travail de thèse que j'ai effectué. En outre, l'équipe MADYNES participe au projet RNRT[1] Safari[2] qui propose l'étude, la réalisation et l'expérimentation d'une architecture de réseau intégrée pour la conception, le déploiement et l'exploitation optimale de services dynamiques sur un réseau IPv6 hybride ad hoc/filaire. Son principal objectif est de mettre à disposition des fournisseurs de services un environnement logiciel dédié aux services à valeur ajoutée capables d'être exploités sur des infrastructures à forte dynamique, couplant les réseaux ad hoc et les réseaux fixe au travers d'une couche de convergence IPv6 étendue. Un des sous-projets de Safari concerne la conception et la mise en œuvre d'une infrastructure de supervision adaptée aux objectifs du projet. La plate-forme retenue est Jxta. Notre travail s'applique ainsi directement à ce contexte.

8.2 Spécialisation du modèle de l'information générique pour Jxta

Le modèle de l'information générique que nous avons présenté dans le chapitre 5 ne permet pas de prendre en compte toutes les particularités de Jxta. Nous l'avons donc spécialisé afin d'intégrer les concepts relatifs à la plate-forme ainsi que l'ensemble des métriques fournies par le projet MMP qui instrumente la plate-forme. Ceci a conduit à la conception d'un nouveau schéma, nommé Jxta. Par souci de simplicité, nous n'avons pas utilisé le modèle de métriques de CIM [46] mais nous avons inséré les métriques directement dans les classes auxquelles elles se rapportent.

Remarque : La spécification MOF de ce modèle est donnée en annexe C.

8.2.1 Sous-modèle de l'organisation

L'extension du sous-modèle de l'organisation que nous avons conçue pour Jxta est représentée sur la figure 8.2. Tout d'abord, elle montre que nous avons étendu les notions génériques de pair et de communauté à travers les classes Jxta_JxtaPeer et Jxta_JxtaPeergroup qui correspondent

[1] Réseau National de Recherche en Télécommunications
[2] http ://www.telecom.gouv.fr/rnrt/rnrt/projets/res_02_04.htm

aux concepts introduits par Jxta. Le rôle d'un pair dans une communauté est décrit à l'aide de la classe d'association Jxta_JxtaParticipatingPeer, qui hérite de la classe P2P_ParticipatingPeers et dont les attributs sont mentionnés sur la figure 8.3. Parmi ceux-ci, les attributs booléens isRendezvous et isRelay renseignent sur le rôle éventuel de rendez-vous ou relais d'un pair. Ensuite, pour représenter les relations topologiques entre les pairs, nous avons conçu trois classes d'associations qui héritent toutes de la classe générique P2P_TopologicalLink : la classe Jxta_RendezvousConnection représente un lien entre un pair simple et un pair de rendez-vous, la classe Jxta_RelayConnection, un lien entre un pair et un relais, et enfin la classe Jxta_RendezvousPeerView, un lien entre deux pairs de rendez-vous[3]. Par le biais de ce modèle, nous sommes capables de représenter les différents pairs Jxta, les groupes et leur organisation hiérarchique, ainsi que les différentes relations topologiques que les pairs entretiennent.

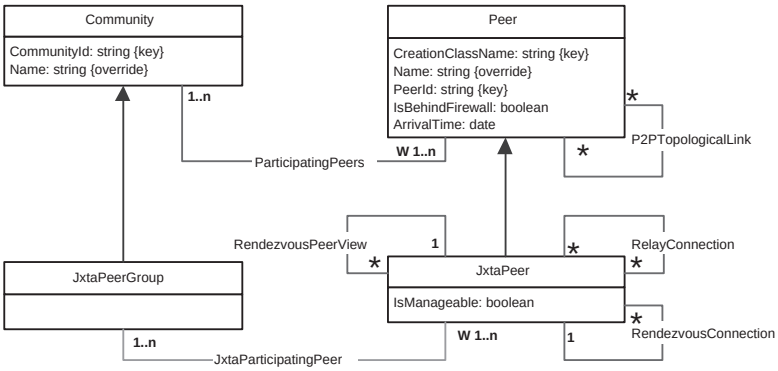


FIG. 8.2 – Le sous-modèle de l'organisation de Jxta

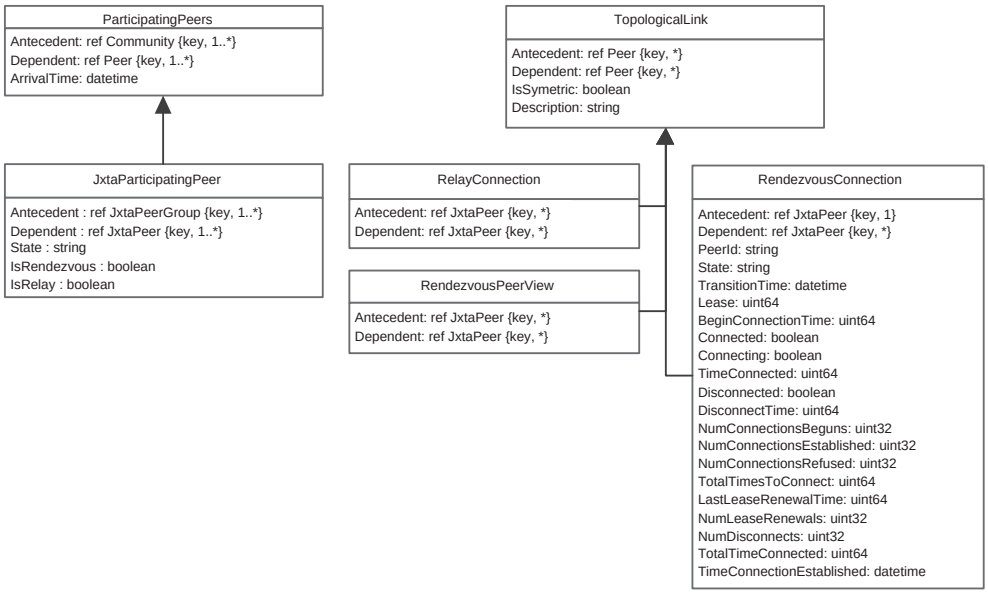


FIG. 8.3 – Les classes d'association du sous-modèle de l'organisation

[3]qui communiquent dans le cadre du service SRDI

8.2.2 Sous-modèle de la communication

La conception d'un sous-modèle de la communication qui représente les *pipes* Jxta à l'aide du formalisme CIM n'est pas directe. En effet, les deux mondes n'ont pas la même conception d'un *pipe*. Jxta, de son côté, stipule qu'un *pipe* peut être *unicast*, et connecter ainsi un pair source à un pair destination, ou propagé et connecter un pair source à plusieurs destinations. La modélisation de ces deux types de *pipes* par le biais de la classe CIM_NetworkPipe du modèle commun pour les réseaux [42], n'est pas directe car dans le formalisme CIM, un *pipe* est lié strictement à deux *endpoints*. La représentation d'un *pipe* propagé pose donc problème.

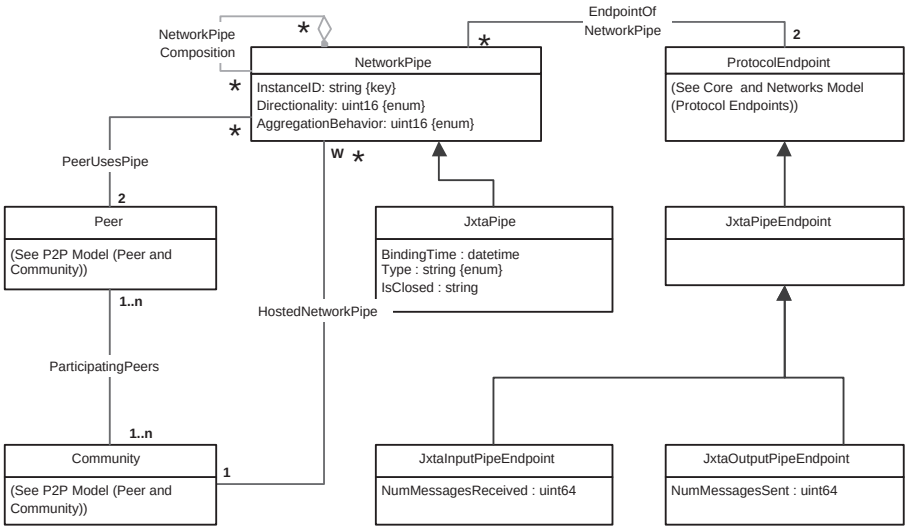


FIG. 8.4 – Le sous-modèle de la communication de Jxta

En outre, après plusieurs tests sur l'implantation Java de Jxta, nous nous sommes aperçu qu'en fait, un *pipe unicast* pouvait relier plusieurs sources à une destination. De la même manière, un *pipe* propagé peut en fait relier plusieurs sources à plusieurs destinations.

Pour résoudre ce problème, nous avons conçu le sous-modèle représenté sur la figure 8.4. La classe Jxta_JxtaPipe représente un *pipe* tel qu'il est décrit dans les documentations fournies par Sun et pas tel qu'il apparaît dans l'implantation de la plate-forme. Les attributs de cette classe permettent de référencer la date de création du *pipe* (BindingTime), de différencier les *pipes unicast* et propagés (Type) et d'indiquer l'état du *pipe* qui peut être ouvert ou clos (IsClosed).

Un *pipe* au sein de la spécification Jxta va donc s'apparenter à la composition de pipes de bas niveau. Ces derniers sont représentés directement à l'aide de la classe CIM_NetworkPipe car ils correspondent bien au concept de *pipe* tel qu'il est présenté dans CIM.

Concernant les *endpoints*, chaque *pipe* de bas niveau, représenté par la classe CIM_NetworkPipe, va bien se trouver lié à deux *endpoints*. Par contre, un *pipe* de haut niveau tel qu'il est conçu par Jxta va présenter plusieurs *endpoints*. Arbitrairement, nous avons choisi de représenter l'ensemble des *endpoints* sources à travers une seule classe, et de même pour l'ensemble des *endpoints* destinations. L'identifiant de ces *endpoints* virtuels étant donné par la concaténation des *endpoints* de niveau sous-jacent.

8.2.3 Sous-modèle de services

Le sous-modèle de services pour Jxta que nous avons conçu a été dicté par les données de gestion fournies par MMP. MMP instrumente huit services du noyau de la plate-forme pour lesquels il propose un large panel de métriques. La figure 8.5 présente notre proposition de sous-modèle de services et la figure 8.6 détaille les métriques présentes dans chacun des services. On constate tout d'abord que nous avons conçu les classes Jxta_PeerService et Jxta_PeerGroupService qui héritent toutes les deux de la classe P2P_LocalP2PService. Celles-ci représentent respectivement les instances locales des services de pair et de groupe Jxta tels qu'ils sont définis par la plate-forme[4]. Les huit services instrumentés par MMP sont des services de groupe qui doivent être implantés par tout pair qui désire joindre le groupe par défaut appelé *NetPeerGroup*.

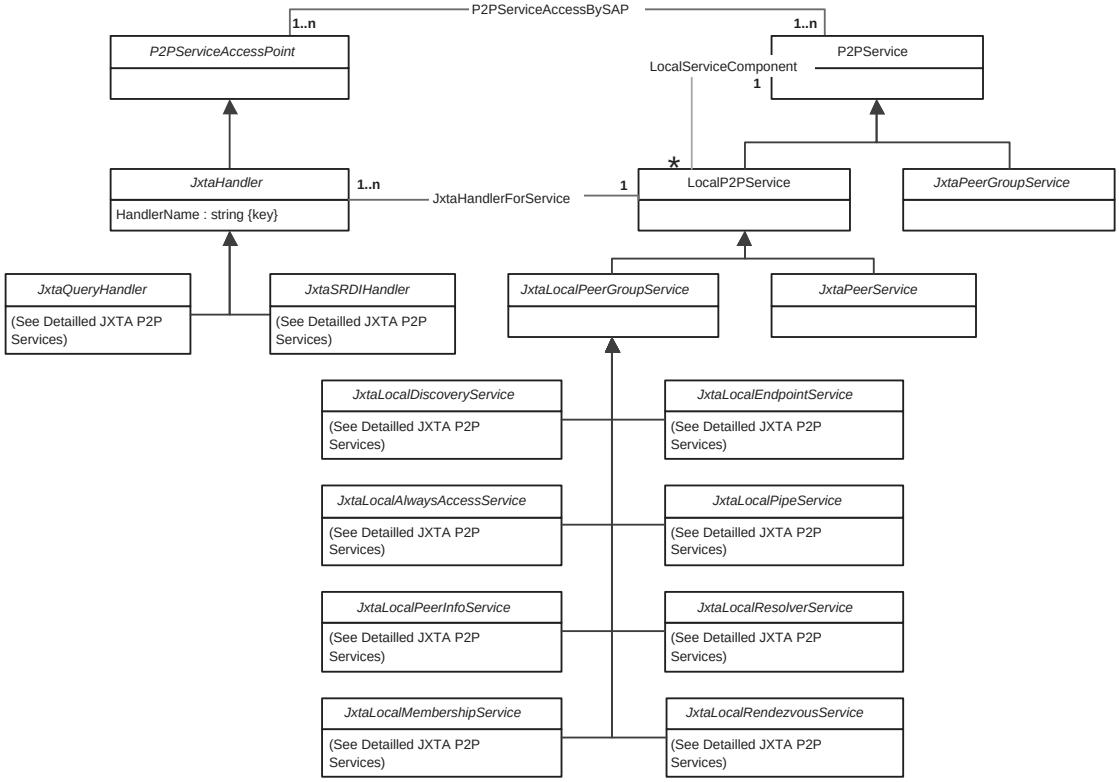


FIG. 8.5 – Le sous-modèle de services de Jxta

Afin de représenter les points d'accès, nous avons conçu la classe abstraite Jxta_JxtaHandler qui hérite de la classe P2P_P2PServiceAccesPoint et qui permet de faire la distinction entre les points d'accès Jxta et un point d'accès générique. Dans la plate-forme, deux types de points d'accès particuliers sont utilisés : l'un est spécifique au service SRDI et l'autre est générique à l'ensemble des autres services. Ces deux points d'accès différents ont conduit à la conception de deux classes qui sont Jxta_JxtaSRDIHandler et Jxta_JxtaQueryHandler. D'après la figure 8.6, on constate que MMP ne fournit pas seulement des métriques relatives aux services Jxta mais aussi des métriques relatives à leurs points d'accès.

Remarque : Nous n'avons pas encore conçu de classes qui représentent une vue globale des

[4]cf. section 2.3.2

instances locales de services. C'est pourquoi aucune classe du sous-modèle de service Jxta n'hérite directement de la classe P2P_P2PService.

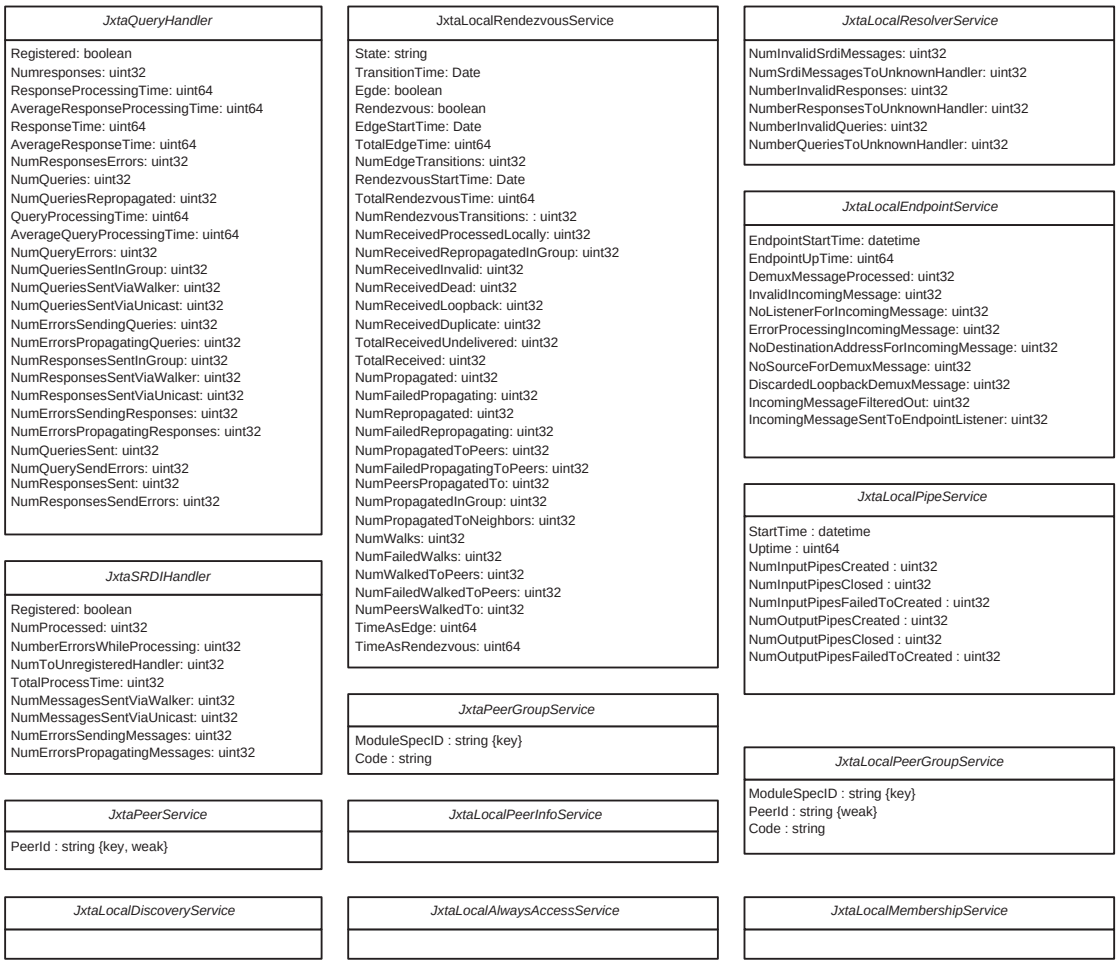


FIG. 8.6 – Détail des métriques intégrées à chaque classe de service Jxta

8.3 Intégration d'un agent de gestion dans les pairs

Le modèle de l'information pour Jxta que nous avons conçu formalise la vue globale dont un gestionnaire dispose. Certaines de ses données sont relatives à un pair et d'autres à plusieurs. C'est par exemple le cas d'un *pipe* qui est, par définition, partagé. Nous nous sommes donc posé la question de la manière dont ce modèle serait distribuable parmi une communauté de pairs. Dans cette section, nous présentons la solution que nous proposons. Celle-ci repose sur deux niveaux de présentation des données. Le premier est local à un agent et le second, utilisé par un gestionnaire, est global à l'ensemble d'une communauté.

8.3.1 Définition d'une vue locale

Nous avons décliné une version locale du modèle présenté précédemment, dont le schéma se nomme JxtaLocal. Nous ne la présentons pas ici car elle est très similaire à la version globale.

Nous soulignons ici simplement la manière dont nous avons traduit des classes partagées entre des pairs en pointeurs. Pour cela, nous utilisons l'exemple de la figure 8.7 qui illustre le cas de la représentation de la connexion d'un pair simple à son pair de rendez-vous. La figure 8.7.a montre qu'au niveau du modèle global, la classe Jxta_RendezvousConnection lie un pair de rendez-vous à lui-même. Au niveau de l'instanciation du modèle en Java, on trouvera donc un objet de la classe Jxta_JxtaPeer représentant un pair simple, un objet de la même classe représentant son pair de rendez-vous, et un objet de la classe Jxta_RendezvousConnection contenant un pointeur vers chacun des deux objets cités précédemment. Les agents JMX du pair simple et du pair de rendez-vous stockeront les instances respectives de la classe Jxta_JxtaPeer. La question est alors : *Où est stockée l'instance de la classe Jxta_RendezvousConnection ?* Pour résoudre ce problème, nous proposons dans notre vue locale de définir la notion d'objet distant. Dans le cas d'un pair lié par le biais d'une association topologique, nous avons défini la classe JxtaLocal_RemotePeer qui représente un pair distant. Le schéma local de cette association est représenté sur la figure 8.7.b. L'ensemble des objets est ainsi hébergeable au sein de l'agent JMX implanté dans le pair concerné.

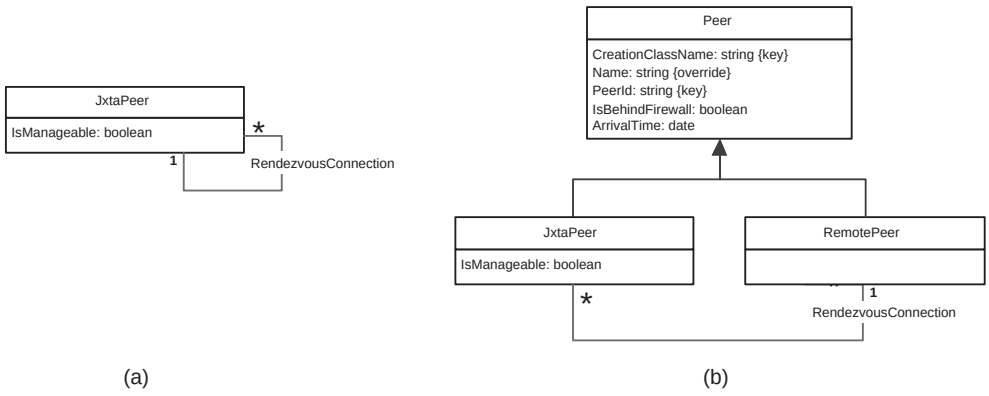


FIG. 8.7 – Exemple d'adaptation de notre modèle de l'information en une vue locale. (a) Vue globale. (b) Vue locale.

8.3.2 Traduction des classes MOF en classes Java

Les modèles de l'information local et global que nous avons conçus sont spécifiés sous forme de diagrammes UML que nous avons traduits dans le langage MOF de CIM. Afin de traduire cette spécification MOF en classes Java, nous avons utilisé l'outil MOF2MBean[5] de Sun. Cet outil permet de générer des paquetages de classes qui sont organisées de manière à respecter les concepts de CIM : chaque classe CIM génère une classe Java, l'héritage est respecté et certains qualifieurs sont adaptés à Java[6]. Par exemple, le qualifieur `Max(n)` qui permet de limiter la longueur d'une chaîne à n caractères est traduit par un test de type `if(attr.length > n){...}` effectué dans la méthode `set` de l'attribut correspondant. Par ce biais, chaque classe CIM est traduite en une classe Java.

Ensuite, MOF2MBean implante dans chaque classe Java une interface de type MBean standard qui permet de les intégrer dans un agent JMX. Un schéma de nommage qui référence de

[5]http ://java.sun.com/products/JavaManagement/JMXperience.html
[6]les autres sont simplement ignorés

Listing 8.1 – Extrait de la spécification MOF des pairs Jxta

```
// Peer
[Version ("1.0.0"), Description (...) ]
class P2P_Peer : CIM_EnabledLogicalElement {

    [Key, MaxLen (256), Description (...) ] string CreationClassName;
    [Key, MaxLen (256), Description (...) ] string PeerId;
    [Override ("Name"), MaxLen (256), Description (...) ] string Name;
    [Description (...) ] datetime ArrivalTime;
};

// JxtaPeer
[Version ("1.0.0"), Description (...) ]
class JXTA_JxtaPeer : P2P_Peer {

    [Description (...) ] boolean IsManageable;
};
```

manière unique chaque instance de classe dans l'agent est aussi proposé. Il se compose d'un nom de domaine qui peut être choisi arbitrairement et d'une série de couples `attribut=valeur` qui est constituée des attributs clés de la classe CIM. Un couple supplémentaire `class=...` qui renseigne sur la classe Java correspondant à l'objet est ajouté. Au sein de l'agent JMX, une instance de cette classe se nomme de la manière suivante :
`<PeerId>:class="Jxta.JxtaPeer",CreationClassName="Jxta_JxtaPeer",PeerId=<PeerId>`

Le listing 8.1 montre un extrait de la spécification MOF de la classe Jxta_JxtaPeer qui représente un pair Jxta. Sa traduction dans le langage Java est représentée dans le listing 8.2. On peut remarquer que chaque classe CIM est traduite par une classe Java abstraite dont le développeur doit donner une implantation. L'héritage CIM n'est donc pas direct et passe par une classe d'implantation dont le nom se termine par convention par `Impl`.

8.3.3 Architecture et fonctionnement

L'architecture fonctionnelle des agents de gestion que nous avons insérés dans les pairs Jxta est représentée sur la figure 8.8. Le coeur de l'agent a pour rôle principal de détecter les groupes auxquels le pair participe et pour chacun de ces groupes, de lancer un processus autonome, appelé *PeerGroupTimerTask*, qui se charge de collecter et mettre à jour les informations de gestion relatives au groupe dont il s'occupe.

Chaque processus *PeerGroupTimerTask* rapatrie ses informations de gestion par une méthode de *polling* qui consiste à interroger régulièrement les services MMP. Ensuite, afin de limiter les opérations d'enregistrement et de suppression d'objets de l'agent JMX, à chaque nouvelle scrutation, le processus *PeerGroupTimerTask* compare la vue qu'il possède avec celle qu'il vient d'obtenir. Seuls les objets qui diffèrent entre ces deux vues conduisent à des opérations d'insertion et de suppression dans l'agent. Enfin l'ensemble des objets gérés, enregistrés dans l'agent JMX, sont rendus accessibles par le biais du protocole RMI.

Remarque : La fréquence de *polling* de l'agent peut être paramétrée par le biais de l'interface de supervision. Pour cela, nous avons intégré dans l'agent JMX un objet géré dédié à la configuration de l'agent. Cet objet contient plusieurs attributs qui sont accessibles à un gestionnaire en lecture et écriture.

Listing 8.2 – Extrait de la classe Java représentant un pair Jxta

```
// Class P2P_Peer
package cim.p2p.peer;

public abstract class P2P_Peer extends CIM_EnabledLogicalElementImpl
                               implements P2P_PeerMBean, MBeanRegistration {
    public P2P_Peer() throws InvalidAttributeValueException {
        super();
        className = new String("P2P_Peer");
        ...
    }

    public String getCreationClassName() throws ImplementationException,
                                                InvalidAttributeValueException {
        String result = _do_getCreationClassName();

        if (result != null)
            if (result.length() > 256)
                throw new MaxLenConstraintViolationException(result.length(),256);
        return result;
    }

    public String getPeerId() throws ... { ... }
    public String getName() throws ... { ... }
    public CIMDateTime getArrivalTime() throws ... { ... }

    protected abstract String _do_getCreationClassName() throws ... ;
    protected abstract String _do_getPeerId() throws ... ;
    protected abstract String _do_getName() throws ... ;
    protected abstract CIMDateTime _do_getArrivalTime() throws ... ;

    ...
}

// Class JXTA_JxtaPeer
package cim.jxta.jxtapeer;

public abstract class JXTA_JxtaPeer extends P2P_PeerImpl
                                    implements JXTA_JxtaPeerMBean, MBeanRegistration {

    public JXTA_JxtaPeer() throws InvalidAttributeValueException {
        super();
        className = new String("JXTA_JxtaPeer");
        ...
    }

    public Boolean getManageable() throws ImplementationException {
        Boolean result = _do_getManageable();
        return result;
    }

    protected abstract Boolean _do_getManageable() throws ... ;
}
```

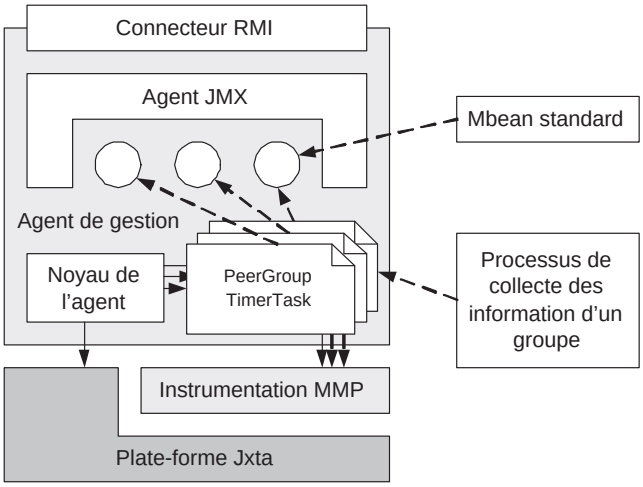


FIG. 8.8 – Architecture fonctionnelle des agents de gestion Jxta

8.4 Développement d'un gestionnaire

Nous avons développé un gestionnaire dont la fonction est d'agréger les données de gestion issues des agents pour fournir une vue globale d'une communauté. La vue qu'il offre suit le modèle de l'information présenté dans la section 8.2. Afin de respecter le modèle d'organisation de Jxta fondé sur des groupes, nous avons choisi de désigner, pour chaque groupe Jxta, un pair qui se voit confier le rôle de gestionnaire.

Dans la suite de cette section, nous présentons l'architecture du gestionnaire, la manière dont il découvre ses agents, et le processus d'élection du gestionnaire d'un groupe.

8.4.1 Election d'un pair gestionnaire et découverte des agents

Lorsqu'un pair rejoint une communauté, il n'a aucune idée sur la présence d'un éventuel gestionnaire et le cas échéant, sur son identité. La première opération qu'il effectue consiste donc à demander aux pairs présents qui est gestionnaire de ce groupe. Chaque pair du groupe reçoit cette requête et y répond seulement s'il est le gestionnaire. Si un gestionnaire existe, à la réception de ce message, il enregistre le nouveau pair dans la liste de ses agents. Si au bout d'un temps fixé par un *timeout*[7] le nouvel agent ne reçoit aucune réponse, il se proclame gestionnaire de la communauté.

Nous avons implanté ce protocole sous la forme d'un service Jxta, enregistré dans la plate-forme. Ce service possède ainsi un *handler* spécifique.

8.4.2 Architecture et fonctionnement général

L'architecture fonctionnelle du gestionnaire que nous avons développé est représentée sur la figure 8.9. Pour chaque groupe où le pair qui héberge ce service est élu gestionnaire, un processus appelé *PeerGroup Manager* est créé. Ce processus a pour fonction de collecter les données de gestion fournies par les différents agents du groupe et d'en fournir une vue agrégée. Pour cela, le *PeerGroup Manager* utilise trois composants. Le premier est le noyau du gestionnaire,

[7]La valeur du *timeout* est paramétrable par le biais du MBean de configuration de l'agent.

appelé *Core*. Il répond aux requêtes émises par les pairs qui recherchent un gestionnaire, et gère l'enregistrement de nouveaux agents et leur suppression. Un agent peut en effet être supprimé du gestionnaire pour deux raisons : la première est sa notification au gestionnaire qu'il quitte la communauté ; la seconde est l'impossibilité pour le gestionnaire de se connecter à l'interface RMI qu'offre son agent JMX. Le deuxième composant du gestionnaire est le *Remote Agent*, un composant chargé de rapatrier les données de gestion fournies par un agent particulier. Le *PeerGroup Manager* instancie autant de *Remote Agents* qu'il supervise d'agents. Le dernier composant, intitulé *Shared Data Processor* a pour rôle de gérer les données partagées entre plusieurs agents. C'est par exemple le cas des *pipes* qui sont des données de gestion que le gestionnaire intègre dans son modèle de l'information, mais qui ne sont générés par aucun des agents, ceux-ci ne disposant que de *endpoints*.

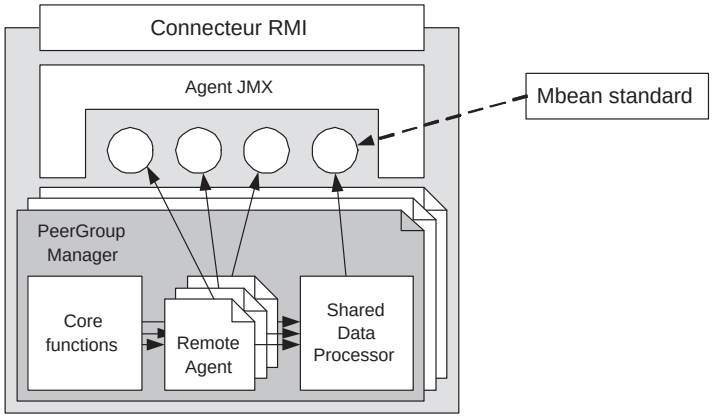


FIG. 8.9 – Architecture fonctionnelle du gestionnaire

8.4.3 Rapatriement des données de gestion

Pour construire la vue globale d'une communauté, le gestionnaire repose sur une approche hybride composée de *polling* et de délégation. Le *polling* est utilisé pour connaître la nature des objets de gestion qu'un agent héberge. Le gestionnaire peut ainsi créer ses propres objets, à l'image de ceux qu'il a découverts dans les agents. La délégation est utilisée pour récupérer la valeur des attributs de chacun des objets de gestion. En effet, les attributs des objets hébergés par le gestionnaire ne contiennent aucune valeur. Ils sont simplement des pointeurs vers les attributs des objets correspondants dans l'agent à qui ils font référence. Ainsi, lorsque qu'une application interroge l'agent JMX contenu dans le gestionnaire pour récupérer la valeur d'un attribut, celui-ci va déléguer cette requête à l'agent correspondant. Les valeurs des attributs sont ainsi les plus proches possibles de la réalité.

8.5 Développement d'une application de gestion

Le travail d'agrégation des données de gestion effectué par le gestionnaire nous a permis de développer une application de gestion qui, comme le montre la figure 8.10, offre à un administrateur une véritable interface de supervision de la plate-forme. Dans cette section, nous détaillons chacune des fonctionnalités qu'elle offre.

8.5.1 Découverte des gestionnaires

L'application est conçue pour fonctionner de manière indépendante des groupes de pairs qu'elle supervise. Elle peut en effet superviser simultanément plusieurs groupes, chacun ayant son propre gestionnaire. Pour pouvoir accéder aux données hébergées par un gestionnaire particulier, l'application doit se connecter à lui. Nous proposons deux manières d'effectuer cette connexion. La première est manuelle et nécessite de connaître l'adresse RMI du gestionnaire. La seconde est effectuée automatiquement par la découverte de l'ensemble des gestionnaires des groupes visibles et accessibles. Pour ce faire, durant cette phase, l'application de gestion exécute un pair Jxta qui va scruter les différents groupes accessibles et chercher au sein de chaque groupe l'existence d'un gestionnaire. Chaque gestionnaire présent répond à cette requête de découverte en indiquant son adresse RMI. Une fois l'opération de recherche effectuée, le pair est arrêté.

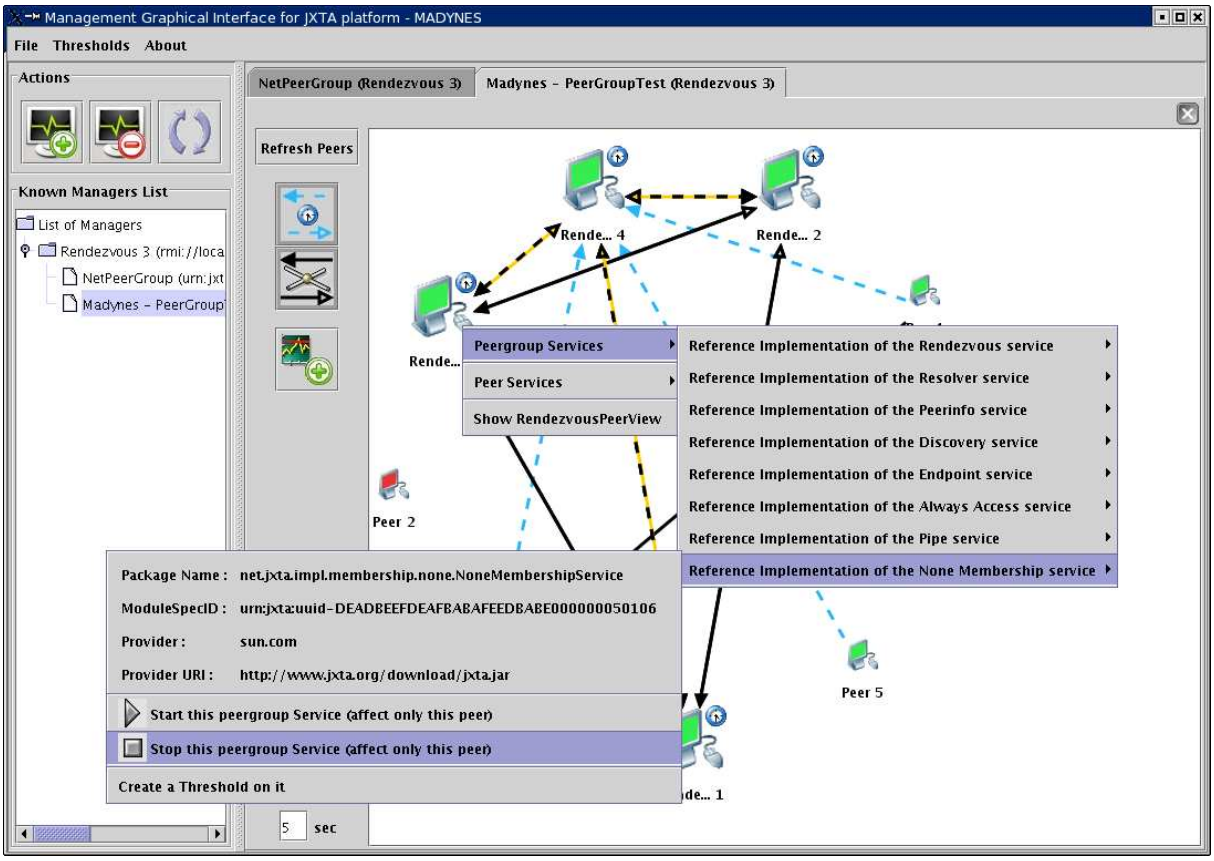


FIG. 8.10 – Un aperçu de notre application de gestion pour Jxta

8.5.2 Tracé d'une vue topologique

Les fonctions offertes par notre application s'articulent principalement autour d'une vue topologique des pairs de la communauté supervisée. Les données de gestion récupérées auprès du gestionnaire nous autorisent à distinguer les différents types de pairs et les différents liens topologiques. Concernant la nature des pairs, l'application fait ainsi la distinction entre les pairs simples, de rendez-vous et relais, et, du point de vue des relations topologiques, elle distingue

les liens de connexion d'un pair à son pair de rendez-vous, les liens d'un pair à un pair relais et les liens entre pairs de rendez-vous.

8.5.3 Monitorage de la plate-forme

Les fonctions de surveillance proposées par notre application permettent de s'abonner à une des métriques[8] proposées dans les données de gestion et de surveiller sa valeur. Le tracé de son évolution au cours du temps est possible mais l'aspect le plus intéressant concerne la possibilité de fixer un seuil qui, lorsqu'il est franchi, déclenche une alarme. Cette alarme peut être de différentes natures : l'inscription dans un fichier journal, l'apparition d'un message dans l'application ou l'envoi d'un mail à l'administrateur.

D'un point de vue de la réalisation, nous avons confié l'intégralité de la tâche de surveillance à notre application de gestion. Ainsi, elle interroge régulièrement le gestionnaire pour récupérer la valeur courante de la métrique et, si un seuil est fixé, elle compare cette valeur avec celui-ci.

8.5.4 Interaction avec la plate-forme

Enfin, la dernière fonctionnalité offerte par notre application concerne le contrôle de la plate-forme Jxta. A ce jour, la seule action que avons rendue possible concerne, pour un pair donné, le démarrage et l'arrêt de services à distance. Cette fonctionnalité, combinée à celle de surveillance d'une métrique, permet de créer des règles de fonctionnement que la plate-forme va devoir appliquer. Par exemple, imaginons qu'un pair mal intentionné rejoigne une communauté et tente une attaque de déni de service en saturant de trafic les *pipes* qu'il a établis avec ses voisins. La surveillance de la métrique NumMessagesSent de la classe JxtaOutputPipeEndPoint va permettre la détection de cette saturation et l'application de gestion va alors provoquer l'arrêt du service de *pipe* du pair concerné pour mettre en déroute cette attaque.

8.6 Synthèse

La réalisation d'une infrastructure de supervision pour Jxta s'inscrit dans le cadre de la validation du modèle de l'information générique que nous avons proposé, et dans le cadre du projet Safari. La spécialisation de notre modèle de l'information et sa mise en œuvre dans la plate-forme Jxta a montré que notre modèle pouvait parfaitement intégrer les spécificités de cette plate-forme.

L'infrastructure que nous avons mise en œuvre se compose de trois éléments : un agent de gestion intégré aux pairs Jxta qui héberge une vue locale du modèle de l'information ; un gestionnaire, lui aussi hébergé dans un pair dont le rôle est l'agrégation des données fournies par les agents ; et une application de gestion qui interagit avec le gestionnaire et qui offre une interface de supervision à un administrateur.

L'ensemble de cette réalisation a été développée en langage Java. Nous avons choisi ce langage pour sa portabilité et pour garantir l'intégration des fonctions de supervision dans les pairs Jxta. En outre, nous avons utilisé la technologie JMX comme cadre de développement de nos agents et du gestionnaire. Cette technologie, dédiée à la supervision des applications Java s'est montrée parfaitement appropriée à notre travail.

[8] à valeur numérique

Conclusion

Chapitre 9

Conclusions et perspectives

Sommaire

La thématique de recherche relative à la supervision du modèle P2P par l'utilisation d'approches standard était peu explorée lorsque nous avons commencé nos travaux. Cependant, l'utilisation actuelle du modèle P2P dans des environnements contraints comme des entreprises ou des administrations, nécessite l'intégration de mécanismes de supervision adaptés. Comme nous l'avons vu dans la première partie du manuscrit, il existe très peu d'approches de gestion explicites pour le modèle P2P. Le travail effectué durant ma thèse n'a donc pas consisté en l'amélioration d'approches existantes mais plutôt en l'exploitation d'une voie nouvelle qui a permis de dégager des premiers objectifs et d'initier plusieurs travaux novateurs. Ces derniers ne se sont donc logiquement pas concentrés sur un point particulier de la supervision des réseaux et services P2P, mais sur des aspects différents de la gestion.

Ces travaux de recherche ont porté sur trois des quatre modèles qui définissent une infrastructure de supervision et qui sont : le modèle de l'information, le modèle fonctionnel de la performance et le modèle organisationnel. Pour chacun d'entre eux nous proposons un résumé, une analyse et quelques perspectives de travaux futurs.

9.1 Un modèle de l'information de gestion

Ce modèle définit et formalise les données de gestion qui entrent dans le cadre de la supervision du modèle P2P. Nous avons recensé les différents aspects du modèle P2P qui sont communs à toute application. Ceci a conduit à la conception de cinq sous-modèles relatifs à l'organisation, à la communication, aux ressources, aux services et au routage. Le formalisme que nous avons retenu pour exprimer ce modèle est CIM. Nous l'avons choisi pour sa nature orientée objet qui facilite son extension, la richesse des éléments existants et son caractère standard. Son instanciation sur Chord, Pastry et Jxta a montré sa généricité et sa capacité à prendre en compte les spécificités de chaque plate-forme.

Sur la base du modèle que nous avons conçu, de nombreuses pistes de travaux futurs sont possibles.

Concernant le sous-modèle de services, les différents éléments que nous avons identifiés, comme la décomposition d'un service en une vue locale et une vue globale, ou la manière dont un service est distribué au sein d'une communauté, se justifient, mais la formalisation que nous en avons faite est repensable. Le modèle d'applications de CIM [47] propose maintenant un ensemble de classes qui représentent des logiciels, composés de fonctionnalités générales, elles-mêmes composées d'éléments logiciels. Le modèle d'applications concerne aussi bien les applications centralisées que les applications distribuées et chaque élément logiciel peut être rattaché à un service et un point d'accès particulier. Par le biais de ces classes, les caractéristiques de service du modèle P2P sont prises en compte, et il ne manque que l'ajout de quelques classes d'associations vers des pairs et des communautés pour rattacher ce modèle à une vue organisationnelle du modèle P2P. La proposition de modèle de service que nous avons faite n'est donc pas la seule possible.

Le modèle de routage, qui hérite du sous-modèle de service peut aussi être formalisé autrement. Dans le sous-modèle de réseaux du modèle *Common*, CIM propose des classes de modélisation des services de routage et de calcul de routes. Celles-ci héritent de la classe de service et représentent ces services, de manière locale à un routeur. Cet aspect purement local ne satisfaisant pas notre contrainte de vue locale et globale, nous avons redéfini deux classes de même sémantique mais héritant de notre classe de service P2P. Cette double définition ne correspond pas à la philosophie de CIM et, étant donné le modèle d'applications, il aurait aussi été possible de définir ces classes dans ce cadre.

Ensuite, le travail de déploiement de notre modèle de l'information sur Jxta a mis en évidence la lourdeur de CIM : au sein d'un agent, le nombre d'instances de classes est très important, (environ 170 objets) et l'héritage induit par les différentes classes du modèle *Core* rendent les objets assez lourds.

Concernant les perspectives, la question de la manière de déployer efficacement un tel modèle dans un contexte P2P reste ouverte. Dans nos travaux de mise en œuvre sur Jxta et Pastry, nous avons utilisé un gestionnaire centralisé qui agrège les informations fournies par des agents. Cette solution n'a eu pour objectif que de montrer la validité de notre modèle en laissant de côté les aspects de distribution de l'information de gestion. Nous avons conscience que le passage à l'échelle d'une telle infrastructure n'est pas des meilleurs. Pour remédier à ce problème nous envisageons d'utiliser une solution distribuée qui repose par exemple sur une DHT. De cette manière, les informations de gestion seraient réparties équitablement entre les différents pairs.

9.2 Instanciation du modèle de l'information sur Jxta

Le modèle de l'information de gestion que nous avons conçu a été instancié sur Chord, Pastry et Jxta et déployé sur ces deux dernières plates-formes. En particulier, le travail effectué sur Jxta a consisté à (1) intégrer des agents de gestion construits autour d'un agent JMX dans les pairs Jxta, (2) construire un gestionnaire qui agrège les données de gestion fournies par les agents et les synthétise sous la forme d'objets qui suivent notre modèle de l'information, et (3) développer une application de gestion qui propose des fonctions de monitorage et de contrôle de la plate-forme.

A partir du travail existant, nous envisageons plusieurs pistes de travaux futurs qui portent sur :

Le protocole de découverte : Nous avons conçu ce protocole d'une manière très simple car le travail de conception et de mise en œuvre d'un service de découverte d'agents et de gestionnaires dans un environnement P2P représente un travail à part entière et nous avons préféré concentrer notre travail sur la validation de notre modèle de l'information.

Néanmoins, nous sommes conscients que notre protocole, fondé sur un *timeout* doit être fiabilisé. Lorsqu'un agent recherche un gestionnaire, s'il ne reçoit aucune réponse durant le *timeout*, il se proclame gestionnaire. Si un gestionnaire est présent mais répond après l'expiration du *timeout*, la communauté va se trouver avec deux gestionnaires concurrents et ce cas de figure n'est pas pris en compte dans notre version actuelle du protocole.

La gestion de la liste d'agents maintenue par le gestionnaire : Un agent est supprimé de la liste maintenue par le gestionnaire s'il est impossible d'accéder à son connecteur RMI. Or, dans un environnement P2P, il faut prendre en compte les déconnexions intermittentes des pairs qui peuvent se produire, surtout dans le cas où les communications reposent sur un réseau sans-fil. Ainsi, comme le fait Tapestry [170], lorsqu'il est impossible d'établir une connexion avec un agent, il faudrait effectuer plusieurs tentatives et ne supprimer effectivement l'agent qu'au bout d'un nombre d'essais donné. Pendant ce laps de temps, toutes les données de gestion enregistrées dans le gestionnaire devraient être conservées et leur potentielle obsolescence signalée.

Le maintien des informations par le gestionnaire : Pour connaître la nature des différents objets de gestion à instancier dans son agent JMX, le gestionnaire interroge régulièrement les agents qu'il supervise. L'avantage de cette méthode réside dans la simplicité de sa mise en œuvre. Néanmoins, son fonctionnement limite le passage à l'échelle de l'infrastructure de gestion. Pour pallier ce problème, nous proposons d'utiliser le service de notification de JMX qui permettrait d'alléger fortement ce fonctionnement. Lorsqu'un changement intervient dans les objets de gestion d'un agent, une notification serait émise vers le gestionnaire qui pourrait adapter sa vue en conséquence.

La délégation des fonctions de monitorage : La tâche de surveillance d'une métrique est intégralement prise en charge par l'application de gestion qui interroge régulièrement le gestionnaire pour récupérer sa valeur. Il nous semble maintenant plus efficace de déléguer cette tâche au gestionnaire ou même directement aux agents. Le service de monitorage offert par JMX est parfaitement adapté à l'exécution de cette fonction.

Concernant la valorisation de ce travail, nous comptons exposer notre réalisation à la communauté Jxta et leur proposer de créer un projet centré sur notre infrastructure qui permettrait à d'autres développeurs de prendre une part active à son amélioration, et de l'intégrer aux futures versions de la plate-forme.

9.3 Un modèle fonctionnel orienté vers la performance pour les DHTs

Les DHTs sont des infrastructures de découverte et de localisation de ressources qui forment actuellement le coeur d'un bon nombre d'applications P2P. Leur bon fonctionnement est crucial à celui des applications qui les utilisent et, d'un point de vue de la gestion, il est important de pouvoir surveiller et contrôler leur performance. Nous avons donc abstrait le comportement générique d'une DHT, défini un ensemble de métriques qui la caractérise et étendu notre modèle de l'information générique pour intégrer ces nouvelles données. Un travail d'instanciation sur Chord nous a permis d'appliquer cette proposition à une infrastructure concrète tout en spécialisant notre proposition initiale.

Le monitorage et le contrôle de la performance des tables de hachage distribuées constitue un sujet de recherche à part entière. Le travail que nous avons effectué ouvre une voie dans cette

direction, avec la proposition d'une formalisation du processus de découverte, la définition, pour ce processus, de métriques, et le choix d'un modèle de l'information.

Nous envisageons trois étapes pour poursuivre ce travail. La première consiste à mener un travail identique à celui effectué sur le processus de découverte sur les processus d'insertion, de retrait et de maintenance. Par ce biais, nous disposerions d'une abstraction totale du fonctionnement d'une table de hachage distribuée. La seconde consiste à implanter ces modèles au sein d'une ou plusieurs DHTs pour valider l'ensemble de cette proposition. La troisième porte sur les actions de contrôle qu'un gestionnaire pourrait effectuer à partir des données collectées. En effet, si le monitorage d'une table de hachage permet d'estimer sa performance, il n'est pas suffisant pour en garantir un certain niveau. Nous avons envisagé plusieurs types d'actions qui seraient entreprenables par une infrastructure de gestion. Parmi celles-ci, on compte : le démarrage et l'insertion de nouveaux nœuds qui pourraient résoudre des problèmes de charge, l'exécution forcée du processus de maintenance sur certains nœuds pour résoudre des problèmes d'incohérence des méta données, et le changement des identifiants des nœuds pour équilibrer les nœuds dans l'espace de nommage.

9.4 Un modèle d'organisation hiérarchique du plan de gestion

Les architectures de supervision actuelles s'adaptent mal aux caractéristiques du modèle P2P. Elles sont souvent centralisées et tolèrent difficilement le dynamisme induit par la présence intermittente des pairs d'une communauté. Nous avons proposé une nouvelle architecture de gestion pour le modèle P2P qui repose sur une hiérarchie de gestionnaires. L'utilisation des préfixes des pairs permet d'équilibrer la structure et d'impliquer un nombre conséquent de pairs dans le rôle de gestionnaire, conduisant ainsi à assigner fréquemment la double fonction de gestionnaire et d'agent aux pairs. Nous avons déployé une application prototype de cet algorithme sur Pastry. Les tests à petite échelle que nous avons conduits sont concluants et nous incitent à poursuivre dans cette voie. Cependant, il semble que l'utilisation d'une métrique de poids qui permette de choisir les pairs qui accèdent au rôle de gestionnaire soit trop coûteuse pour être utilisée. Actuellement, nous conduisons une série de tests à plus grande échelle sur un *cluster* de 200 machines, qui nous permet de simuler une communauté comptant jusqu'à 4000 nœuds. Ce travail est en cours de réalisation.

La prochaine étape de ce travail consiste à définir le rôle de chaque gestionnaire et à implanter des fonctions de gestion dans cette hiérarchie. Une piste intéressante consisterait à intégrer à cette architecture la proposition de modélisation de la performance des DHTs présentée ci-dessus. Ceci conduirait à une infrastructure de supervision orientée vers la performance complète pour les tables de hachage distribuées.

La thématique relative à la supervision des réseaux et services P2P est très ouverte. Comme nous l'avons expliqué, les propositions que nous avons faites méritent d'être poursuivies et peuvent conduire, pour chacune d'entre elles, à une thématique de recherche à part entière. En outre, il existe d'autres voies qui pourraient être explorées. Je pense en particulier à l'utilisation du modèle P2P pour sa propre supervision. De la même manière qu'un pair joue le rôle de client et serveur d'un service, il pourrait aussi jouer le double rôle d'agent et gestionnaire sur le plan de la supervision. Chaque gestionnaire ne serait responsable que du pair qui l'héberge, conduisant ainsi à des communautés exemptes de toute hiérarchie. La collecte des données de gestion pourrait s'effectuer par la scrutation du voisinage et les actions de gestion de manière indépendante et concurrente. Cette approche s'apparente à un mécanisme d'auto-gestion

et permettrait à un administrateur de travailler exclusivement avec des politiques qui règlent le comportement et les objectifs d'une communauté.

Références

Bibliographie

[1] K. Aberer. P-Grid : A self-organizing access structure for P2P information systems. In C. Batini, F. Giunchiglia, P. Giorgini, and M. Mecella, editors, *Proceedings of the 9th International Conference on Cooperative Information Systems - CooplS'01*, number 2172 in LNCS, pages 179–194. Springer-Verlag, 2001.

[2] K. Aberer and Z. Despotovic. Managing trust in a peer-2-peer information system. In *Proceedings of the tenth international conference on Information and Knowledge Management - CIKM'01*, pages 310–317. ACM Press, 2001.

[3] K. Aberer and M. Hauswirth. Peer-to-peer information systems : concepts and models, state-of-the-art, and future systems. Tutorial ICDE'2002, 2002. Distributed Information Systems Laboratory (LSIR).

[4] E. Adar and B. Huberman. Free riding on gnutella. *First Monday*, 5(10), 2000.

[5] N. Agoulmine. *Standards pour la gestion des réseaux et des services*, chapter Gestion de réseau par les politiques : protocole et modèles d'information. Hermes, 2003.

[6] S. Ajmani, D. E. Clarke, C. Moh, and S. Richman. ConChord : Cooperative SDSI certificate storage and name resolution. In P. Druschel, F. Kaashoek, and A. Rowstron, editors, *Proceedings of the 1st International Workshop on Peer-to-Peer Systems - IPTPS'02*, number 2429 in LNCS, pages 141–154. Springer-Verlag, 2002.

[7] S. B. Akers and B. Krishnamurthy. A group-theoretic model for symmetric interconnection networks. *IEEE Transactions on Computers*, 38(4) :555–566, 1989.

[8] D. Anderson. *Peer-to-peer : Harnessing the Power of Disruptive Technologies*, chapter SETI@Home, pages 67–76. O'Reilly & Associates, Inc., 2001.

[9] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4) :335–371, December 2004.

[10] P. Antoniadis and C. Courcoubetis. Market models for P2P content distribution. In G. Moro and M. Koubarakis, editors, *Proceedings of the First International Workshop on Agents and Peer-To-Peer Computing - AP2PC'02*, number 2530 in LNCS, pages 138–143. Springer-Verlag, 2002.

[11] P. Antoniadis, C. Courcoubetis, and R. Mason. Comparing economic incentives in peer-to-peer networks. *Computer networks*, 46 :133–146, 2004.

[12] O. Babaoglu, H. Meling, and A. Montresor. Anthill : A framework for the development of agent-based peer-to-peer systems. In *The 22th International Conference on Distributed Computing Systems (ICDCS '02)*. IEEE Computer Society, 2002.

[13] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk. Cryptographic hash functions : A survey. Technical Report 95-09, Department of Computer Science, University of Wollongong, July 1995.

[14] M. Baldi, S. Gai, and G. P. Picco. Exploiting code mobility in decentralized and flexible network management. In K. Rothermel and R. Popescu-Zeletin, editors, *Proceedings of the First International Workshop on Mobile Agents - MA'97*, number 1219 in LNCS, pages 13–26. Springer-Verlag, 1997.

[15] H. B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communication of the ACM*, 13(7) :422–426, July 1970.

[16] J. Braure. Développement d'une infrastructure de supervision pour la plate-forme JXTA. Master's thesis, Institut Universitaire Professionnalisé - Département de Génie Electrique et Informatique Industrielle - Université Henri Poincaré, Nancy, 2005.

[17] F. Buccafurri and G. Lax. TLS : A tree-based DHT lookup service for highly dynamic networks. In R. Meersman and Z. Tari, editors, *Proceedings of the OTM confederated international conferences On the Move to Meaningful Internet Systems 2004 : CoopIS, DOA, and ODBASE*, number 3290 in LNCS, pages 563–580. Springer-Verlag, 2004.

[18] W. Bumpus, J. W. Sweitzer, P. Thompson, Westerinen ; A. R., and R. C. Williams. *Common Information Model*. Wiley, 2000.

[19] C. Buragohain, D. Agrawal, and S. Suri. A game theoretic framework for incentives in p2p systems. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing - P2P'03*, pages 48–57. IEEE Computer Society, 2003.

[20] F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette, V. Néri, and O. Lodygensky. Computing on large scale distributed systems : Xtremweb architecture, programming models, security, tests and convergence with grid. *FGCS Future Generation Computer Science*, 2004.

[21] D. Caromel, Klauser W., and J. Vayssière. Towards seamless computing and metacomputing in Java. *Concurrency : Practice and Experience*, 10(11-13) :1043–1061, 1998.

[22] J. D. Case, M. Fedor, Schoffstall M. L., and J. Davin. *Simple Network Management Protocol (SNMP)*. Internet Engineering Task Force, may 1990. RFC 1157 (Historic).

[23] M. Castro, P. Drushel, C. Hu, and A. Rowstron. Proximity neighbor selection in tree-based structured peer-to-peer overlays. Technical Report MSR-TR-2003-52, Microsoft Research, 2003.

[24] K. Chan, J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, and A. Smith. *COPS Usage for Policy Provisioning (COPS-PR)*. Internet Engineering Task Force, March 2001. RFC 3084 (Proposed Standard).

[25] T. M. Chen. A model and evaluation of distributed network management approaches. *IEEE journal on selected areas in communications*, 20(4) :850–857, May 2002.

[26] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet : a distributed anonymous information storage and retrieval system. In H. Federrath, editor, *Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability*, number 2009 in LNCS, pages 46–66. Springer-Verlag, 2001.

[27] B. F. Cooper. Ad hoc, self-supervising peer-to-peer search networks. *ACM Transactions on Information Systems*, 23(2) :169–200, April 2005.

[28] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems : Concepts and Design Edn. 3*. Pearson Education, 2001.

[29] R. Cox, A. Muthitacharoen, and R. Morris. Serving DNS using Chord. In P. Druschel, F. Kaashoek, and A. Rowstron, editors, *Proceedings of the 1st International Workshop on*

Peer-to-Peer Systems - IPTPS'02, number 2429 in LNCS, pages 155–165. Springer-Verlag, 2002.

[30] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles - SOSP'01*, pages 202–215. ACM Press, 2001.

[31] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common API for structured P2P overlays. In F. Kaashoek and I. Stoica, editors, *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems - IPTPS'03*, number 2735 in LNCS, pages 33–44. Springer-Verlag, 2003.

[32] M. Datar. Butterflies and peer-to-peer networks. In R. Möhring and R. Raman, editors, *Proceedings of the 10th Annual European Symposium on Algorithms - ESA'02*, number 2461 in LNCS, pages 310–322. Springer-Verlag, 2002.

[33] A. Datta, S. Dirdzijauskas, and K. Aberer. On De Bruijn routing in distributed hash tables : There and back again. Technical Report IC/2004/41, Swiss Federal Institute of Technology Lausanne (EPFL), 2004.

[34] J. Davies, T. Manion, R. Rao, J. Miller, and X. Zhang. *Introduction to Windows Peer-to-Peer Networking.* Microsoft Corporation, July 2004. http ://www.microsoft.com/technet/prodtechnol/winxppro/deploy/p2pintro.mspx.

[35] N. De Bruijn. *Koninklijke Nerderlanse Academie van Wetenschappen Proc.*, volume 49, chapter A combinatorial problem, pages 758–764. Indagationes Math, 1946.

[36] A. Di Ferdinando, P. Mckee, and A. Amoroso. A policy based approach for automated topology management of peer to peer networks and a prototype implementation. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks - POLICY'03*, pages 235–238. IEEE Computer Society Press, 2003.

[37] R. Dingledine, M. J. Freedman, and D. Molnar. The free haven project : Distributed anonymous storage service. In H. Federrath, editor, *Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability*, number 2009 in LNCS, pages 67–95. Springer-Verlag, 2000.

[38] R. Dingledine, M. J. Freedman, and D. Molnar. *Peer-to-peer : Harnessing the Power of Disruptive Technologies*, chapter Free Haven, pages 159–187. O'Reilly & Associates, Inc., 2001.

[39] Distributed Management Task Force. *Common Information Model (CIM) Specification, version 2.2*, June 1999. DSP0004.

[40] Distributed Management Task Force. *Common Information Model (CIM) Core model, version 2.4*, August 2000. DSP0111.

[41] Distributed Management Task Force. *CIM Concepts White Paper, CIM versions 2.4+*, 2003. DSP0144.

[42] Distributed Management Task Force. *CIM Network Model White Paper, CIM version 2.7*, June 2003. DSP0152.

[43] Distributed Management Task Force. *CIM Policy Model White Paper, CIM version 2.7*, June 2003. DSP0139.

[44] Distributed Management Task Force. *CIM System Model White Paper, CIM version 2.7*, June 2003. DSP0150.

[45] Distributed Management Task Force. *CIM User and Security Model White Paper, CIM version 2.7*, June 2003. DSP0139.

[46] Distributed Management Task Force. *Common Information Model (CIM) Metrics Model, version 2.7*, June 2003. DSP0141.

[47] Distributed Management Task Force. *Understanding the Application Management Model, CIM version 2.7*, June 2003. DSP0140.

[48] W. Dou, Y. Jia, H. M. Wang, W. Q. Song, and P. Zou. A p2p approach for global computing. In *Proceedings of the International Parallel and Distributed Processing Symposium - IPDPS'03*, page 248. IEEE Computer Society, 2003.

[49] D. Doval and D. O'Mahony. Overlays networks : A scalable alternative for P2P. *IEEE Internet Computing*, 7(4) :77–82, July-August 2003.

[50] P. Druschel and A. Rowstron. Past : A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 8th IEEE workshop on Hot Topics in Operating Systems - HotOS VIII*, pages 75–80. IEEE Computer Society, 2001.

[51] T. C. Du, E. Y. Li, and A. P. Chang. Mobile agents in distributed network management. *Communincations of the ACM*, 46(7) :127–132, 2003.

[52] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry. *The COPS (Common Open Policy Service) Protocol*. Internet Engineering Task Force, January 2000. RFC 2748 (Proposed Standard).

[53] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi. Efficient broadcast in structured P2P networks. In F. Kaashoek and I. Stoica, editors, *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems - IPTPS '03*, volume 2735 of *LNCS*. Springer-Verlag, 2003.

[54] A. H. Esfahanian and S. L. Hakimi. Fault-tolerant routing in De Bruijn communication networks. *IEEE Transactions on Computers*, 34(9) :777–788, 1985.

[55] G. Fedak, C. Germain, V. Neri, and F. Cappello. Xtremweb : A generic global computing system. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid - CCGRID'01*, pages 582–587. IEEE Computer Society, 2001.

[56] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust incentive techniques for peer-to-peer networks. In *Proceedings of the 5th ACM conference on Electronic Commerce - EC'04*, pages 102–111. ACM Press, 2004.

[57] O. Festor and N. Ben Youssef. *Standards pour la gestion des réseaux et des services*, chapter Le modèle CIM, pages 113–157. Hermes, 2003.

[58] O. Festor and A. Schaff, editors. *Standards pour la gestion des réseaux et des services*. Hermes, 2003.

[59] A. Fiat and J. Saia. Censorship resistant peer-to-peer content addressable networks. In *Proceedings of the 30th annual ACM-SIAM symposium on Discrete algorithms - SODA'02*, pages 94–103. Society for Industrial and Applied Mathematics, 2002.

[60] I. Foster and C. Kesselman. Globus : A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2) :115–128, Summer 1997.

[61] G. Fox, S. Pallickara, and X. Rao. Towards enabling peer-to-peer grids. *Concurrency - Practice and Experience*, 17(7-8) :1109–1131, 2005.

[62] P. Fraigniaud and P. Gauron. The content-addressable network D2B. Technical Report TR-LRI-1349, Laboratoire de Recherche en Informatique - Université Paris-Sud, 2003.

[63] P. Fraigniaud and P. Gauron. An overview of the content-addressable network D2B. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing - PODC'03*, pages 151–151. ACM Press, 2003.

[64] M. J. Freedman and R. Morris. Tarzan : A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security - CCS'02*, 2002.

[65] P. Ganesan and G. S. Manku. Optimal routing in Chord. In *Proceedings of the 15th annual ACM-SIAM symposium on Discrete algorithms - SODA'04*, pages 176–185. Society for Industrial and Applied Mathematics, 2004.

[66] D. Gavalas, D. Greenwood, M. Ghanbari, and M. O'Mahony. Hierarchical network management : a scalable and dynamic mobile agent-based approach. *Computer Networks*, 38(6) :693–711, 2002.

[67] P. Golle, K. Leyton-Brown, I. Mironov, and M. Lillibridge. Incentives for sharing in peer-to-peer networks. In *Proceedings of the 3rd ACM conference on Electronic Commerce - EC'01*, pages 264–267. ACM Press, 2001.

[68] L. Gong. Jxta : A network programming environment. *IEEE Internet Computing*, 5(3) :88–95, May-June 2001.

[69] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of DHT routing geometry on resilience and proximity. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM'03*, pages 381–394. ACM Press, 2003.

[70] I. Gupta, K. Birman, P. Linge, A. Demers, and R. Van Renesse. Kelips : building an efficient and stable DHT through increased memory and background overhead. In F. Kaashoek and I. Stoica, editors, *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems - IPTPS'03*, number 2735 in LNCS, pages 160–169. Springer-Verlag, 2003.

[71] M. Gupta, P. Judge, and M. Ammar. A reputation system for peer-to-peer networks. In *Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video - NOSSDAV'03*, pages 144–152. ACM Press, 2003.

[72] G. Hardin. The tragedy of the commons. *Science*, 162 :1243–1248, 1968.

[73] D. Hausheer, N. C. Liebau, A. Mauthe, R. Steinmetz, and B. Stiller. Token-based accounting and distributed pricing to introduce market mechanisms in a peer-to-peer file sharing scenario. In *3rd IEEE International Conference on Peer-to-Peer Computing - P2P'03*, pages 200–201. IEEE Computer Society, 2003.

[74] T. Hong. *Peer-to-peer : Harnessing the Power of Disruptive Technologies*, chapter Performance, pages 203–241. O'Reilly & Associates, Inc., 2001.

[75] Sun Microsystems Inc. JXTA metering and monitoring project, May 2003.

[76] Sun Microsystems Inc. The JXTA metering and monitoring project architecture, May 2003.

[77] Sun Microsystems Inc. The JXTA monitor : GUI rendering of metered peer info, May 2003.

[78] International Organisation for Standardization. *Information processing systems – Open Sytems Interconnection – Basic Reference Model – Part 4 : Management framework*, November 1989. ISO/IEC 7498-4 :1989(E).

[79] M. Jasnowski. *JMX Programming*. Wiley, 2002.

[80] M. Jelasity, R. Guerraoui, A. M. Kermarrec, and M. Van Steen. The peer sampling service : Experimental evaluation of unstructured gossip-based implementations. In *Proceedings of the 5th International Conference on Middleware*. ACM/IFIP/USENIX, 2004.

[81] M. A. Jovanovic, F. S. Annexstein, and K. A. Berman. Scalability issues in large peer-to-peer networks : A case study of gnutella. Technical report, University of Cincinnati, 2001.

[82] M. F. Kaashoek and D. R. Karger. Koorde : A simple degree-optimal distributed hash table. In F. Kaashoek and I. Stoica, editors, *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems - IPTPS'03*, number 2735 in LNCS, pages 98–107. Springer-Verlag, 2003.

[83] M. Kahani and H. W. Peter Beadle. Decentralised approaches for network management. *SIGCOMM Computer Communication Review*, 27(3) :36–47, 1997.

[84] G. Kan, Gnutella, and GoneSilent.com. *Peer-to-peer : Harnessing the Power of Disruptive Technologies*, chapter Gnutella, pages 94–122. O'Reilly & Associates, Inc., 2001.

[85] K. Kant. An analytic model for peer to peer file sharing networks. In *Proceedings of the 38th International Communications Conference - ICC'03*. IEEE, 2003.

[86] K. Kant, R. Iyer, and V. Tewari. A performance model for peer to peer file-sharing services. In *Accepted for WWW-11 poster session*, May 2002.

[87] B. Kantor and P. Lapsley. *Network News Transfer Protocol*. Internet Engineering Task Force, February 1986. RFC 977 (Proposed Standard).

[88] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1) :41–50, January 2003.

[89] J. Kleinberg. The small-world phenomenon : an algorithm perspective. In *Proceedings of the 32nd annual ACM symposium on Theory of computing - STOC'00*, pages 163–170. ACM Press, 2000.

[90] A. V. Konstantinou. *Towards autonomic networks*. PhD thesis, Columbia university, 2003.

[91] D. Kosiur. *Understanding Policy-Based Networking*. Wiley, 2001.

[92] T. Koulouris, R. Henjes, K. Tutschku, and H. De Meer. Implementation of adaptive control for P2P overlays. In N. Wakamiya, M. Solarski, and J. P. G. Sterbenz, editors, *Proceedings of the 5th IFIP TC6 International Workshop on Active Networks - IWAN'03*, number 2982 in LNCS, pages 292–306. Springer-Verlag, 2003.

[93] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore : An architecture for global-scale persistent storage. *SIGARCH Computer Architecture News*, 28(5) :190–201, 2000.

[94] A. Kumar, S. Merugu, Jun Xu, and Xingxing Yu. Ulysses : a robust, low-diameter, low-latency peer-to-peer network. In *Proceedings of the 11th IEEE International Conference on Network Protocols - ICNP'03*, pages 258–267. IEEE Computer Society, 2003.

[95] A. Langley. *Peer-to-peer : Harnessing the Power of Disruptive Technologies*, chapter Freenet, pages 123–132. O'Reilly & Associates, Inc., 2001.

[96] J. Lao and C. Yang. Shortest path routing and fault-tolerant routing on De Bruijn networks. *Networks*, 3(35) :207–215, 2000.

[97] L. Larry Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. *SIGCOMM Computer Communication Review*, 33(1) :59–64, 2003.

[98] J. Li, Stribling J., T. M. Gil, R. Morris, and M. F. Kaashoek. Comparing the performance of distributed hash tables under churn. In G. M. Voelker and S. Shenker, editors, *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04)*, volume 3279 of *LNCS*. Springer-Verlag, 2004.

[99] J. Li and Lim D. Y. A robust aggregation tree on distributed hash tables. In V. Sinha, J. Eisenstein, and T. M. Sezgin, editors, *Proceedings of the 2004 Student Oxygen Workshop*, 2004.

[100] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In *Proceedings of the 21st annual symposium on Principles Of Distributed Computing - PODC'02*, pages 233–242. ACM Press, 2002.

[101] J. Linfors, M. Fleury, and The JBoss Group. *JMX : Managing J2EE with Java Management Extensions*. Sams Publishing, 2002.

[102] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-theoretic analysis of structured peer-to-peer systems : routing distances and fault resilience. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM'03*, pages 395–406. ACM Press, 2003.

[103] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy : A scalable and dynamic emulation of the Butterfly. In *Proceedings of the 21st annual ACM symposium on Principles of Distributed Computing - PODC'02*, pages 183–192. ACM Press, 2002.

[104] G. Malkin. *RIP Version 2*. Internet Engineering Task Force, November 1998. RFC 2453 (Standard).

[105] G. S. Manku, M. Bawa, and P. Raghavan. Symphony : Distributed hashing in a small world. In *Proceedings of the 4th USENIX symposium of Internet Technologies and Systems - USITS'03*. USENIX, 2003.

[106] S. Marti and H. Garcia-Molina. Identity crisis : Anonymity vs. reputation in p2p systems. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing - P2P'03*, pages 134–141. IEEE Computer Society, 2003.

[107] J. P. Martin-Flatin, S. Znaty, and J. P. Hubaux. A survey of distributed network and systems management paradigms. Technical Report SSC/1998/024, Ecole Polytechnique Fédérale de Lausanne - EPFL, 1998.

[108] P. Maymounkov and D. Mazières. Kademlia : A peer-to-peer information system based on the XOR metric. In P. Druschel, F. Kaashoek, and A. Rowstron, editors, *Proceedings of the First International Workshop on Peer-to-Peer Systems - IPTPS'02*, number 2429 in LNCS, pages 53–65. Springer-Verlag, 2002.

[109] K. McCloghrie, M. Fine, J. Seligson, K. Chan, S. Hahn, R. Sahita, A. Smith, and F. Reichmeyer. *Structure of Policy Provisioning Information (SPPI)*. Internet Engineering Task Force, August 2001. RFC 3159 (Proposed Standard).

[110] S. Milgram. The small world problem. *Psychology Today*, 2 :60–67, 1967.

[111] J. Miller. *Peer-to-peer : Harnessing the Power of Disruptive Technologies*, chapter Jabber : Conversational Technologies, pages 77–88. O'Reilly & Associates, Inc., 2001.

[112] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical Report HPL-2002-57, HP Laboratories, 2002.

[113] J. Mischke and B. Stiller. Peer-to-peer overlay network management through Agile. In G. Goldszmidt and J. Schönwälder, editors, *Proceedings of the 8th symposium on Integrated Network Management - IM'03*, pages 337–350. Kluwer Academic Publisher, 2003.

[114] R. Mohanty. Instrumentation of the JXTA peer-to-peer framework. Master's thesis, Department of Computer Science and Engineering - Indian Institute of Technology, Kharagpur, 2004.

[115] B. Moore. Policy Core Information Model (PCIM) Extensions. RFC 3460 (Proposed Standard), January 2003.

[116] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen. Policy Core Information Model – Version 1 Specification. RFC 3060 (Proposed Standard), February 2001. Updated by RFC 3460.

[117] M. Naor and U. Wieder. Novel architectures for P2P applications : the continuous-discrete approach. In *Proceedings of the 15th annual ACM symposium on Parallel Algorithms and Architectures - SPAA'03*, pages 50–59. ACM Press, 2003.

[118] J. F. Nash. Equilibrium points in n-person games. *Proceedings of National Academy of Science*, 36 :48–49, 1950.

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

[120] L. Olson. .NET P2P : Writing peer-to-peer networked apps with the microsoft .NET framework. *MSDN Magazine*, 16(2), February 2001.

[121] A. Oram, editor. *Peer-to-peer : Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, Inc., March 2001.

[122] C. G. Plaxton, R. Rajaraman, and Richa A. W. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings the 9th annual ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320. ACM Press, 1997.

[123] D. Ratajczak and J. M. Hellerstein. Deconstructing DHTs. Technical Report IRB-TR-03-042, Intel Reasarch Berkeley, 2003.

[124] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures and Protocols for Computer Communication - SIGCOMM'01*, pages 161–172. ACM Press, 2001.

[125] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Third International Workshop on Networked Group Communication - NGC'01*, pages 31–43, 2001.

[126] M. K. Reiter and A. D. Rubin. Crowds : anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1) :66–92, 1998.

[127] M. Rennhard. *MorphMix – A Peer-to-Peer-based System for Anonymous Internet Access*. PhD thesis, TIK – ETH Zurich – Switzerland, 2004.

[128] M. Rennhard and B. Platter. Introducing morphmix : Peer-to-peer based anonymous internet usage with collusion detection. In *Proceedings of the Workshop on Privacy in the Electronic Society - WPES'02, in association with 9th ACM Conference on Computer and Communications Security - CCS'02*, pages 91–102. ACM Press, 2002.

[129] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman. Reputation systems. *Communication of the ACM*, 43(12) :45–48, 2000.

[130] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond : The Oceanstore prototype. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies - FAST'03*, 2003.

[131] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *Proceedings of the USENIX Annual Technical Conference*, 2004.

[132] S. C. Rhea and J. Kubiatowicz. Probabilistic location and routing. In *Proceedings of the 21st annual joint conference of the IEEE computer and communications societies - INFOCOM 2002*, volume 3, pages 1248–1257. IEEE Computer Society, 2002.

[133] M. Ripeanu. Peer-to-peer architecture case study : Gnutella network. Technical Report TR-2001-26, Department of Computer Science - University of Chicago, 2001.

[134] R. Rivest. The MD5 Message-Digest Algorithm . RFC 1321 (Informational), April 1992.

[135] M. Rose. *The Blocks Extensible Exchange Protocol Core*. Internet Engineering Task Force, March 2001. RFC 3080 (Proposed Standard).

[136] M. Rose, G. Klyne, and D. Crocker. *The Application Exchange (APEX) Access Service*. Internet Engineering Task Force, July 2002. RFC 3341 (Proposed Standard).

[137] A. Rowstron and P. Druschel. Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms - Middleware'01*, number 2218 in LNCS, pages 329–350. Springer-Verlag, 2001.

[138] A. I. T. Rowstron, A. M. Kermarrec, M. Castro, and P. Druschel. SCRIBE : The design of a large-scale event notification infrastructure. In J. Crowcroft and M. Hofmann, editors, *Proceedings of the 3rd international COST264 workshop on Networked Group Communication*, number 2233 in LNCS, pages 30–43. Springer-Verlag, 2001.

[139] J. Saia, A. Fiat, S. D. Gribble, A. R. Karlin, and S. Saroiu. Dynamically fault-tolerant content addressable networks. In P. Druschel, F. Kaashoek, and A. Rowstron, editors, *Revised Papers from the First International Workshop on Peer-to-Peer Systems - IPTPS'01*, number 2429 in LNCS, pages 270–279. Springer-Verlag, 2002.

[140] P. Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP) : Core*. Internet Engineering Task Force, October 2004. RFC 3920 (Proposed Standard).

[141] P. Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP) : Instant Messaging and Presence*. Internet Engineering Task Force, October 2004. RFC 3921 (Proposed Standard).

[142] C. Sangpachatanaruk and T. Znati. Semantic driven hashing (SDH) : An ontology-based search scheme for the semantic aware network (SA Net). In *Proceedings of the 4th International Conference on Peer-to-Peer Computing - P2P'04*, pages 270–271. IEEE Computer Society, 2004.

[143] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In M. G. Kienzle and P. J. Shenoy, editors, *Proceedings of Multimedia Computing and Networking 2002 - MMCN'02*, volume 4673. SPIE, 2002.

[144] S. E. Schechter, R. A. Greenstadt, and M. D. Smith. Trusted computing, peer-to-peer distribution, and the economics of pirated entertainment. In *Proceedings of the Second Workshop on Economics and Information Security*, 2003.

[145] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. Hypercup - hypercubes, ontologies and efficient search on P2P networks. In *Proceedings of the International Workshop on Agents and Peer-to-Peer Computing - AP2PC'02*, 2002.

[146] R. Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architecture and applications. In *Proceedings of the 1st International Conference on Peer-to-Peer Computing - P2P'01*, pages 101–102. IEEE Computer Society, 2001.

[147] C. Shirky. *Peer-to-peer : Harnessing the Power of Disruptive Technologies*, chapter Listening to Napster, pages 21–37. O'Reilly & Associates, Inc., 2001.

[148] M. R. Siegl and G. Trausmuth. Hierarchical network management : A concept and its prototype in snmpv2. *Computer Networks and ISDN Systems*, 28(4) :441–452, 1996.

[149] H. J. Siegle. Interconnection networks for SIMD machines. *Computer*, 16(6) :57–65, 1979.

[150] K. N. Sivarajan and R. Ramaswami. Lightwave networks based on De Bruijn graphs. *IEEE/ACM Transactions on Networking*, 2(1) :70–79, 1994.

[151] W. Stalling. *SNMP, SNMPv2 and RMON*. Addison Wesley, 1997.

[152] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. *ACM SIGCOMM Computer Communication Review*, 32(4) :73–86, 2002.

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

[154] Sun Microsystems Inc. *JXTA v2.0 Protocols Specification*, June 2004. http ://spec.jxta.org/nonav/v1.0/docbook/JXTAProtocols.html.

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

[156] B. Traversat, M. Abdelaziz, and E. Pouyoul. *Project JXTA : A loosely-consistent DHT Rendezvous Walker*. Sun Microsystems Inc., march 2003. http ://www.jxta.org/docs/jxta-dht.pdf.

[157] B. Traversat, A. Arora, and M. Abdelaziz. *Project JXTA 2.0 Super-Peer Virtual Network*. Sun Microsystems Inc., may 2003. http ://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf.

[158] D. Tsoumakos and N. Roussopoulos. Analysis and comparison of P2P search methods. Technical Report CS-TR-4539 UMIACS-TR-2003-107, University of Maryland - Departement of Compuster Science, 2003.

[159] P. Uppuluri, N. Jabisetti, Y. Lee, and U. Joshi. P2P Grid : Service oriented framework for resource management. In *To appear in IEEE International Conference on Web Services - ICWS'05*, 2005.

[160] U.S. Departement of commerce. *Secure hash standard*, federal information processing standards publications edition, April 1995. FIPS PUB 180-1.

[161] J. Verbeke, N. Nadgir, G. Ruetsch, and I. Sharapov. Framework for peer-to-peer distributed computing in a heterogeneous, decentralized environment. In M. Parashar, editor, *Proceedings of the Third International Workshop on Grid Computing - GRID'02*, number 2536 in LNCS, pages 1–12. Springer-Verlag, 2002.

[162] S. Voulgaris, D. Gavidia, and M. Van Steen. CYCLON : Inexpensive membershop management for unstructured P2P overlays. *Journal of Network and Systems Management*, 13(2) :197–217, June 2005.

[163] M. Waldman, A. Rubin, and L. Faith Cranor. Publius : A robust, tamper-evident, censorship-resistant, web publishing system. In *Proc. 9th USENIX Security Symposium*, pages 59–72, August 2000.

[164] Y. Wang and J. Vassileva. Trust and reputation mode in peer-to-peer networks. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing - P2P'03*, pages 150–158. IEEE Computer Society, 2003.

[165] B. J. Wilson. *Jxta*. New Riders Publishing, 2002. First Edition.

[166] L. Xiong and L. Liu. Building trust in decentralized peer-to-peer electronic comunities. In *International Conference on Electronic Commerce Research - ICECR-5*, 2002.

[167] J. Xu, A. Kumar, and X. Yu. On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks. *IEEE Journal on Selected Areas in Communications*, 22(1) :151–163, January 2004.

[168] Y. Yemini, G. Goldszmidt, and S. Yemini. Network management by delegation. In I. Krishnan and W. Zimmer, editors, *Proceedings of the 2nd IFIP TC6 symposium on Integrated Network Management - IM'91*, pages 95–107. Elsevier North-Holland, Inc., 1991.

[169] S. Yoshida, K. Kamei, T. Ohguro, and K. Kuwabara. Shine : a peer-to-peer based framework of network community support systems. *Computer Communications*, 26 :1199–1209, August 2003.

[170] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry : An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, 2001.

[171] Y. Zhu, H. Wang, and Y. Hu. Integrating semantics-based access mechanisms with P2P file systems. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing - P2P'03*, pages 118–125. IEEE Computer Society, 2003.

[172] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux : an architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the 11th international workshop on Network and Operating Systems Support for Digital Audio and Video - NOSSDAV'01*, pages 11–20. ACM Press, 2001.

Publications

Conférences internationales avec comité de lecture

[173] G. Doyen, O. Festor, and E. Nataf. A CIM extension for peer-to-peer network and service management. In J. De Souza and P. Dini, editors, *Proceedings of the 11th International Conference on Telecommunication - ICT'04*, number 3124 in LNCS, pages 801–810. Springer-Verlag, 2004.

[174] G. Doyen, E. Nataf, and O. Festor. A performance-oriented management information model for the Chord P2P framework. In J. Vicente and D. Hutchison, editors, *Management of Multimedia Networks and Services - MMNS'04*, number 3271 in LNCS, pages 200–212. Springer-Verlag, 2004.

[175] G. Doyen, E. Nataf, and O. Festor. A hierarchical architecture for a distributed management of P2P networks and services. In J. Schönwälder and J. Serrat, editors, *Proceedings of the 16th IFIP/IEEE International Workshop on Distributed Systems : Operations and Management - DSOM 2005*, number 3775 in LNCS, pages 257–268. Springer-Verlag, 2005.

[176] E. Nataf, O. Festor, and G. Doyen. A SMIng-centric proxy agent for integrated monitoring and provisioning. In G. Goldszmidt and J. Schönwälder, editors, *The 8th IFIP/IEEE International Symposium on Integrated Network Management - IM'03*, pages 491–503. IFIP, Kluwer Academic Publishers, March 2003.

Conférences nationales avec comité de lecture

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

[178] G. Feltin, G. Doyen, and O. Festor. Les protocoles Peer-to-Peer, leur utilisation et leur détection. In *Journées réseaux - JRES'03*, 2003.

Ateliers internationaux avec comité de lecture

[179] G. Doyen, O. Festor, and E. Nataf. Management of Peer-to-Peer services applied to Instant Messaging. In A. Marshall and N. Agoulmine, editors, *Management of Multimedia Networks and Services*, number 2839 in LNCS, pages 449–461. Springer-Verlag, 2003. End-to-End Monitoring Workshop - E2EMON'03.

[180] G. Doyen, E. Nataf, and O. Festor. Performance management of Distributed Hash Tables. In C. D. Kloos, D. Larrabeiti, A. Marin, C. Campo, R. Romeral, I. Estevez-Ayres, and

M. Uruena, editors, *Proceedings of the 11th IFIP WG 6.4, WG 6.6 and WG 6.9 workshop on Networked Applications - EUNICE'05*, pages 97–102, 2005.

Glossaire

Le domaine de l'informatique et des réseaux utilise un grand nombre d'acronymes. Nous donnons la description de ceux qui sont mentionnés dans le manuscrit.

APEX	*APplication EXchange* : Protocole de niveau application qui assure des fonctions de haut niveau comme la découverte de services ou la gestion de présence.	21
API	*Application Programming Interface* : Une interface de programmation.	45
AVP	*Active Virtual Peer* : Une technologie fondée sur les réseaux actifs pour la gestion de topologie des réseaux P2P non structurés.	58
BEEP	*Block Extensible Exchange Protocol* : Protocole de niveau session qui prend en compte des mécanismes standard comme l'authentification et permet d'établir des sessions P2P entre deux entités.	21
CAN	*Content Addressable Network* : Un réseau au contenu addressable qui repose souvent sur une DHT.	31
CFS	*Collaborative File System* : Une application P2P de stockage de fichiers.	34
CIM	*Common Information Model* : Proposition du DMTF qui comporte un formalisme et un modèle de l'information pour les systèmes informatiques.	24
COPS	*Common Open Policy Service* : Un protocole dédié à la gestion de réseaux par politiques	49
COPS-PR	*COPS for PRovisioning* : Une extension du protocole COPS qui repose sur le modèle de provision.	49
DDNS	*Distributed Domain Name System* : une implantation P2P du DNS.	34
DHCP	*Dynamic Host Configuration Protocol* : Un protocole qui permet de configurer des machines et notamment leur adresse IP.	60
DHT	*Distributed Hash Table* : Une infrastructure P2P de routage et de localisation qui repose sur une table de hachage distribuée.	29
DMBN	*Dynamic Multi-Butterfly Network* : Une proposition de DHT fondée sur les réseaux *Butterfly*.	42
DNS	*Domain Name System* : Un service de résolution des noms de machines qui les traduit en adresses IP.	16
DOLR	*Decentralized location and routing* : Infrastructure P2P de localisation et accès à des objets distants.	45

IETF	*Internet Engineering Task Force* : Organisme composé de chercheurs et d'industriels qui travaillent sur l'évolution de l'Internet à travers la proposition de standards.	21
IP	*Internet Protocol* : Protocole standard de routage de niveau réseau. En outre, il définit un format d'addresse logique des éléments.	21
IPv6	*Internet Protocol version 6* : La sixième version du protocole IP.	121
ISO	*International Organization for Standardization* : Organisme mondial qui identifie les normes internationales nécessaires pour les entreprises, les gouvernements et la société, les élabore en partenariat avec les secteurs qui les appliqueront, les adopte au moyen de procédures transparentes fondées sur la contribution des pays, et les met à disposition pour une application dans le monde entier.	49
JXTA	*JuXTApose* : Plate-forme générique pour le développement d'application P2P proposée par Sun Microsystems Inc.	24
LORIA	*Laboratoire Lorrain de Recherche en Informatique et Automatique* : Le laboratoire qui m'a accueilli durant ma thèse. C'est une unité mixte de recherche (UMR 7503) commune à cinq établissements dont la mission principale concerne la recherche fondamentale et appliquée au niveau international dans le domaine des STIC.	3
MADYNES	*Management of Dynamic Networks and Services* : L'équipe de recherche au sein de laquelle j'ai effectué ma thèse. Sa thématique concerne la gestion des réseaux et services et leur sécurité.	3
MIMD	*Multiple Instructions Multiple Data* : Modèle de programmation qui effectue plusieurs instructions sur plusieurs données en parallèle.	24
MMAPPS	*Marker MAnagement of Peer-to-Peer Services* : Un projet de recherche qui vise à concevoir des modèles de marché pour les communautés P2P d'échange de contenu.	55
MMP	*Metering and Monitoring Project* : Projet Jxta pour l'implantation Java qui consiste en une instrumentation de la plate-forme, un service de monitorage et une console d'administration distante.	27
NNTP	*Network News Transfert Protocol* : Un protocole d'échange de nouvelles sur Internet.	19
OPL	*Object Policy Language* : Une extension d'OSL pour la vérification de la cohérence des règles de changement d'état.	60
OSL	*Object Spreadsheet Language* : Un langage de représentation de règles propagation pour la gestion des systèmes autonomes.	60
P2P	*Pair à Pair ou Peer-to-Peer* : Un modèle distribué où les entités appelées pairs jouent le double rôle de client et serveur.	11
PC	*Personal Computer* : Un ordinateur personnel.	24

PDA	*Personal Digital Assistant* : Un ordinateur tenant dans la main pouvant être connecté à un ordinateur classique pour y déposer ou récupérer des données.	24
PDP	*Policy Decision Point* : Un éléments chargé de la mise en application par des PEPs de politiques mises en œuvre sur un réseau.	49
PEP	*Policy Enforcement Point* : Un point d'application de règles de politiques dictées par un PDP	49
PIB	*Policy Information Base* : Une base de données de politiques	51
QoS	*Quality of Service* : La qualité de service.	51
RNRT	*Réseau National de Recherche en Télécommunications* : Réseau composé de partenaires issus de la recherche publique et de l'industrie et qui promouvoit les innovations françaises en termes de télécommunications à travers la proposition de projets.	121
RPC	*Remote Procedure Call* : Un protocole d'appel de fonctions distantes.	16
RSVP	*Resource reSerVation Protocol* : Un protocole de signalisation pour la réservation de ressources développé pour les services qui requièrent de la QoS.	51
SDSI	*Simple Distributed Security Infrastructure* : Certificats de sécurité.	34
SETI	*Search for Extra Terrestrial Intelligence* : Programme de recherche américain qui consiste à écouter l'espace pour capter un signal de communication extraterrestre.	23
SLM	*Service Level Management* : La gestion de services.	53
SMI	*Structure of Management Information* : Un langage de modélisation des informations de gestion utilisé dans l'approche SNMP.	49
SNMP	*Simple Network Management Protocol* : Un protocole dédié à la gestion de réseau. Il est standardisé par l'IETF.	49
SPI	*Service Provider Interface* : Terme défini dans le projet MMP qui désigne un composant offrant un service.	58
SPPI	*Structure of Policy Provisioning Information* : Un langage de modélisation de gestion utilisé dans la cadre de la gestion par politiques.	49
SRDI	*Shared Resources Directory Index* : Service de découverte et de localisation de ressources de Jxta qui relie les pairs de rendez-vous entre eux.	27
STIC	*Sciences et Technologies de l'Information et de la Communication* : La thématique de recherche dans laquelle s'inscrivent les travaux effectués au LORIA.	3
TCP	*Transmission Control Protocol* : Un protocole de niveau transport fiable.	21
TTL	*Time To Live* : Le temps de vie d'un paquet émis sur un réseau.	43
VLAN	*Virtual Local Area Network* : Un réseau local virtuel de niveau liaison.	59
VPN	*Virtual Private Network* : Un réseau privé virtuel. Il permet à un usager de travailler sur un réseau différent de celui auquel il appartient tout en conservant des services identiques.	48

W3C	*World Wide Web Consortium* : Consortium qui vise à proposer des applications, des standards et des recommandations pour les applications *web*.	26
WBEM	*Web Based Enterprise Management* : Une approche de gestion de réseaux et services proposée par le DMTF.	49
XML	*eXtensible Markup Language* : Un méta langage standardisé par le W3C qui permet de générer des langages à base de balises.	26

Annexes

Annexe A

Compléments sur le modèle de l'information de gestion

Dans cette annexe, nous présentons deux sous-modèles qui font partie de notre modèle générique pour la représentation de l'information de gestion des réseaux et services pair à pair. Le premier est relatif aux ressources et le second au routage.

A.1 Modèle de ressources

Dans le cadre d'un service, les pairs fournissent des ressources qui sont consommées par d'autres. La quantité de ressources disponible ou la manière dont les ressources sont utilisées sont des informations importantes pour la gestion, car elles conditionnent fortement la qualité de service dans lequel elles s'inscrivent. Prenons l'exemple de la *Tragédie des communs* [72]. Celle-ci traite, dans le cadre d'une communauté, de la difficulté à gérer des biens communs. Etant libres et gratuits, ceux-ci sont souvent trop sollicités et du coup deviennent inaccessibles à la majorité des participants. Ce phénomène, s'applique parfaitement au modèle P2P. Dans le cas d'un service de partage de fichiers, une ressource très populaire requise par un trop grand nombre d'usagers, devient inaccessible car le pair qui l'héberge est saturé de requêtes. Il est donc nécessaire, pour pouvoir garantir le bon fonctionnement des services P2P, qu'une infrastructure de gestion obtienne une vue des ressources disponibles et de la manière dont elles sont utilisées.

Nous avons retenu quatre types de ressources qu'un pair peut partager avec une communauté. Les fichiers sont les ressources partagées dans le cadre des services de partage ou stockage de fichiers. Ensuite, un pair peut partager une zone d'espace de stockage dans laquelle seront stockés des fichiers provenant d'autres pairs. On trouve ce type de fonctionnement dans Freenet [26] qui copie chaque fichier requis tout au long du chemin entre les pairs serveur et client du fichier. Le troisième type de ressource que nous avons identifié est la puissance de calcul, représentée au travers de cycles CPU. Ce type de ressource est utilisé dans les applications de calcul distribué comme Seti@Home [8] qui utilisent les processeurs de machines oisives connectées à l'Internet. Enfin, la bande passante disponible pour communiquer via un réseau est le dernier type de ressource que nous considérons dans notre modèle. Aujourd'hui, cette ressource, considérée comme cruciale, est la plus rare de toutes celles citées précédemment [69]. En effet, les puissances de calcul des machines actuelles, de même que les espaces de stockage sont très satisfaisants, mais la capacité des infrastructures de communication reste faible face à la quantité de données que l'on aimerait échanger, par exemple dans le cadre de services de diffusion de contenu. Etre capable de gérer la bande passante nous semble donc un aspect important de la gestion des ressources

P2P.

Diagramme de classes CIM du modèle de ressources

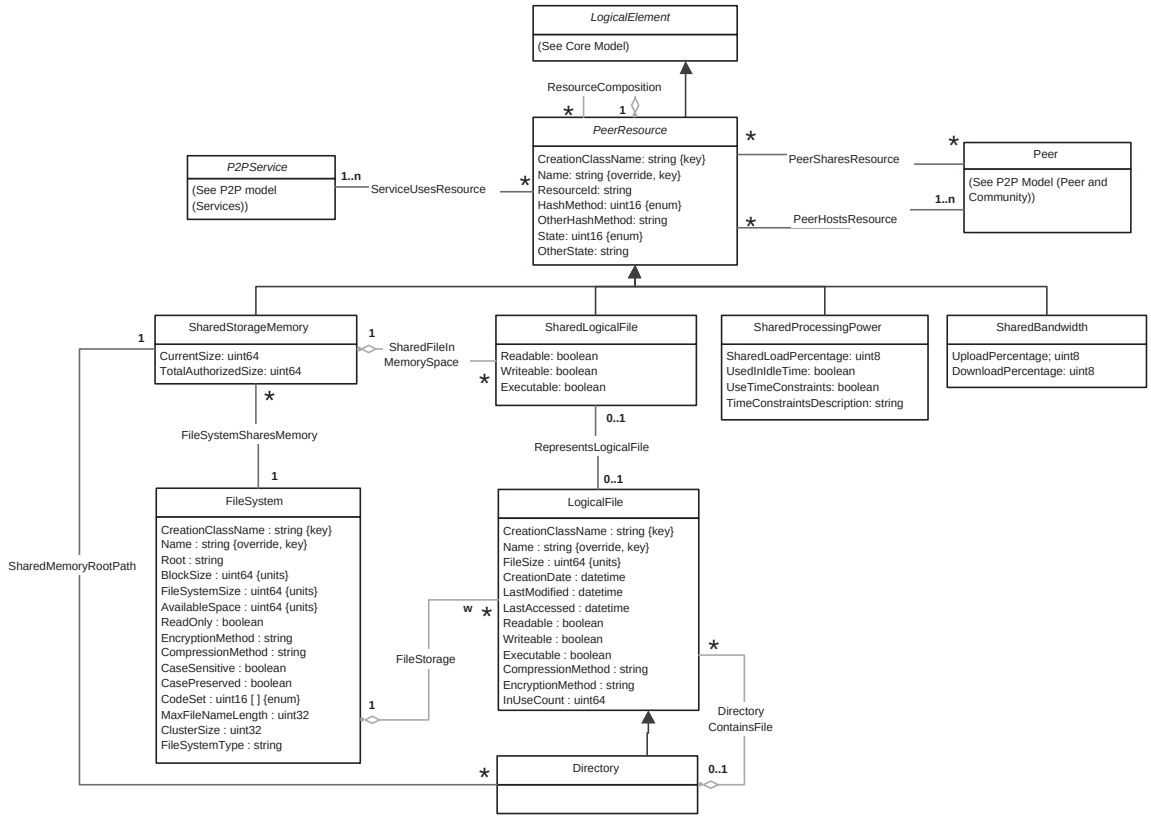


FIG. A.1 – Diagramme de classes CIM du sous-modèle P2P de ressources

Le modèle que nous proposons pour représenter les ressources d'une communauté P2P est illustré sur la figure A.1. Il s'articule autour de la classe abstraite P2P_PeerResource qui représente une ressource générique disponible dans une communauté. Elle se caractérise principalement par le nom de la ressource (Name) et son identifiant (ResourceId) ; la fonction de hachage utilisée pour générer l'identifiant étant renseignée par le biais des attributs HashMethod et OtherHashMethod. Nous avons choisi de la définir comme une sous-classe de la classe CIM_LogicalElement et non de la classe CIM_EnabledLogicalElement du modèle *Core* car, si une ressource P2P peut se trouver dans différents états, ceux décrits dans le cadre de la classe CIM_EnabledLogicalElement ne correspondent pas à la sémantique de notre contexte. C'est pourquoi, nous avons ajouté les attributs State et OtherState qui représentent l'état d'une ressource (disponible, indisponible, erronée, ...). La classe P2P_PeerResource est liée à la classe P2P_Peer à travers deux associations : la première repose sur la classe P2P_PeerSharesResource et met en relation une ressource et le pair qui la partage et à qui elle appartient. La cardinalité de cette association * - * indique qu'un pair peut partager zéro ou plusieurs ressources, et qu'une ressource peut être possédée par zéro ou plusieurs pairs. Ensuite, la classe P2P_PeerHostedResource représente la manière dont une ressource est hébergée. Sa cardinalité indique qu'une ressource est hébergée sur au moins un pair, mais qu'un pair peut héberger zéro ou plusieurs ressources. Enfin, la fourniture et la

consommation de ressources entrent toujours dans le cadre d'un service donné. Pour représenter cette sémantique, nous avons intégré la classe d'association P2P_ServiceUsesResource qui lie la classe P2P_PeerResource à la classe P2P_P2PService et indique qu'une ressource est utilisée dans le cadre d'au moins un service. Pour finir, nous avons autorisé la représentation de la composition de ressources par le biais de la classe d'association P2P_ResourceComposition qui hérite de la classe Component du modèle *Core*. Bien que les attributs et la cardinalité de ces classes soient identiques, nous avons préféré définir une nouvelle classe pour distinguer le lien générique de composition de notre cas d'étude sur les ressources P2P.

Etant donné la classe abstraite P2P_PeerResource qui représente génériquement une ressource P2P, nous avons défini quatre sous-classes qui représentent les ressources concrètes que nous avons recensées dans la section A.1 :

La classe P2P_SharedProcessingPower représente le partage de cycles CPU. Cette information peut être utile dans le cas d'une application de calcul distribué, mais aussi dans des applications diverses, lorsque la machine hébergeant le pair possède des ressources limitées, tel un téléphone portable ou un assistant personnel. Les attributs de cette classe informent de la manière dont la puissance de calcul est partagée avec : le pourcentage d'utilisation d'un service donné (SharedLoadPercentage), les contraintes temporelles liées à l'usage du processeur (UseOfTimeConstraints et TimeConstraintsDescription) et la possible utilisation du processeur durant les périodes d'inactivité de la machine (UseInIdleTime).

La classe P2P_SharedBandwidth informe de la manière dont un pair partage ses ressources de communication réseau. Les attributs UploadPercentage et DownloadPercentage informent du pourcentage d'utilisation des canaux montants et descendants. Cette classe est particulièrement utile dans le cas d'une application d'échange de contenu nécessitant le transfert de données conséquentes qui pourrait monopoliser une part trop importante de la bande passante.

La classe P2P_SharedStorageMemory représente un espace de stockage partagé par un pair, où d'autres pairs vont pouvoir déposer des données. Ses attributs renseignent sur la taille courante de l'espace partagé (CurrentSize) et sur le seuil limite fixé par une application ou l'usager (TotalAuthorizedSize). Nous avons rattaché cette classe au système de fichier auquel elle appartient par le biais de la classe d'association P2P_FileSystemSharesMemory qui la lie à la classe CIM_FileSystem du sous-modèle pour les Systèmes [44] du modèle *Common*. Concernant les cardinalités, on considère qu'un espace de stockage partagé appartient à un et un seul système de fichier, alors qu'un système de fichiers peut héberger zéro ou plusieurs espaces partagés. Enfin à travers l'association P2P_SharedMemoryRootPath, nous indiquons le ou les dossiers qui correspondent à la racine de l'espace partagé.

La classe P2P_SharedLogicalFile représente finalement des fichiers partagés. Nous avons pourvu cette classe d'attributs concernant les droits d'accès au niveau d'une communauté, du fichier qu'elle représente (Readable, Writeable et Executable). Un fichier partagé est lié à un espace de stockage par le biais de l'association P2P_SharedFileInMemorySpace qui stipule qu'un fichier appartient à un et un seul espace partagé, alors que ce dernier peut contenir zéro ou plusieurs fichiers. Enfin, à travers la classe d'association P2P_RepresentsLogicalFile, nous avons laissé la possibilité de mettre éventuellement[1] en relation un fichier ressource P2P avec sa représentation CIM dans la classe CIM_LogicalFile, telle qu'elle est définie dans le sous-modèle de systèmes.

[1] par le biais d'une cardinalité 0..1 - 0..1

A.2 Modélisation du routage

Comme nous l'avons présenté dans le chapitre 3, maintenir une topologie virtuelle et garantir le bon fonctionnement d'un service de routage est un point sensible pour un modèle à caractère dynamique tel que le modèle P2P. C'est pourquoi, il est important qu'une infrastructure de gestion puisse obtenir une vue des services de routage et calcul de routes ainsi que des tables de routage des pairs participant à une communauté. Le sous-modèle de réseaux [42] du modèle *Common* propose deux modèles qui concernent ces aspects, et nous les avons étendus pour le modèle P2P.

Diagramme de classes CIM des services P2P de routage et calcul de routes

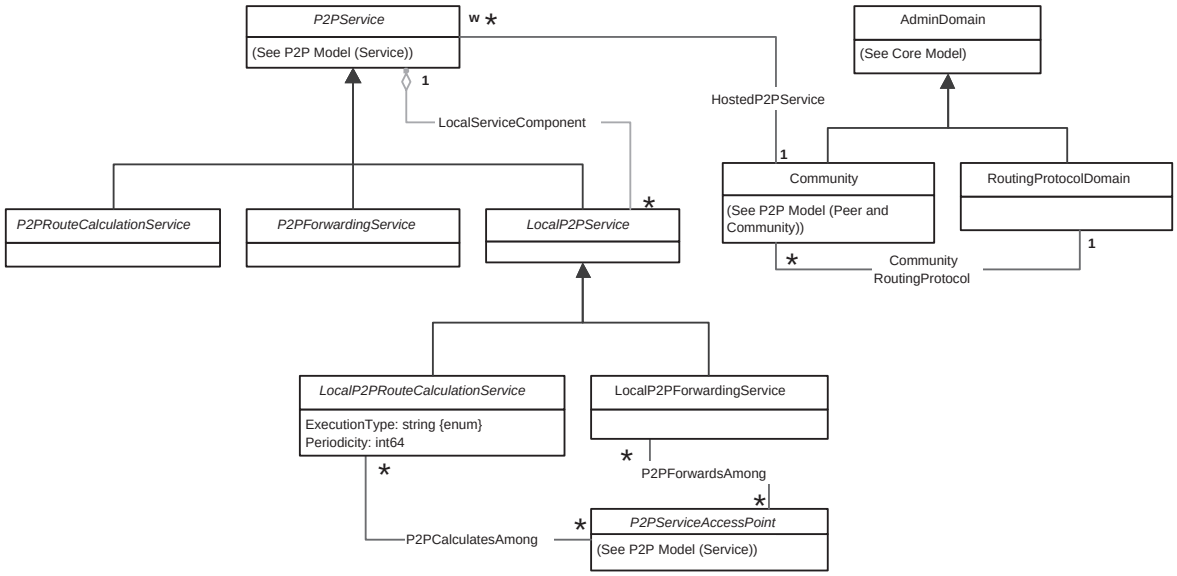


FIG. A.2 – Diagramme de classes CIM des services P2P de routage et calcul de routes

Notre proposition de modélisation CIM pour les services de routage et de calcul de routes P2P est représentée sur la figure A.2. En accord avec la modélisation de services P2P présentée dans la section 5.3.2, nous proposons une vue locale et globale de ces services. Nous n'avons pas ajouté de propriétés à ces classes pour respecter la généricité de notre modèle, sauf pour la classe P2P_localP2PRouteCalculationService qui représente une vue locale du service de mise à jour des tables de routage car, de manière générale, son exécution peut être déclenchée par la détection d'un changement, comme l'insertion d'un nouveau pair dans la communauté, ou à période fixe. Les attributs ExecutionType et Periodicity renseignent sur ces modes opératoires. Ensuite, en accord avec le modèle proposé par CIM dans le sous-modèle de réseaux, les classes P2P_LocalP2PRouteCalculationService et P2P_LocalP2PForwardingService sont liées, respectivement par le biais des classes associatives P2P_P2PForwardsAmong et P2P_P2PCalculatesAmong à des points d'accès qui représentent l'interface par laquelle ces services communiquent.

Pour finir, nous avons ajouté la sous-classe P2P_P2PRoutingProtocolDomain qui hérite de la classe CIM_RoutingProtocolDomain. Cette classe, par le biais de la classe d'association P2P_CommunityRoutingProtocol, informe du protocole de routage en vigueur dans une communauté, sachant qu'une communauté se définit par un protocole de routage unique.

Diagramme de classes CIM des tables de routage

Le diagramme de classes CIM représenté sur la figure A.3 est quasiment celui que CIM propose dans le sous-modèle de réseaux [42] et il est suffisamment générique pour représenter la table de routage qu'un pair maintient. La classe CIM_NextHopRoute représente une entrée dans une table de routage. Elle se caractérise par un identifiant unique (InstanceId), l'adresse de destination de la route (DestinationAddress), une métrique de distance (AdminDistance et RouteMetric), le fait que la route soit statique (IsStatic) et son type (TypeOfRoute).

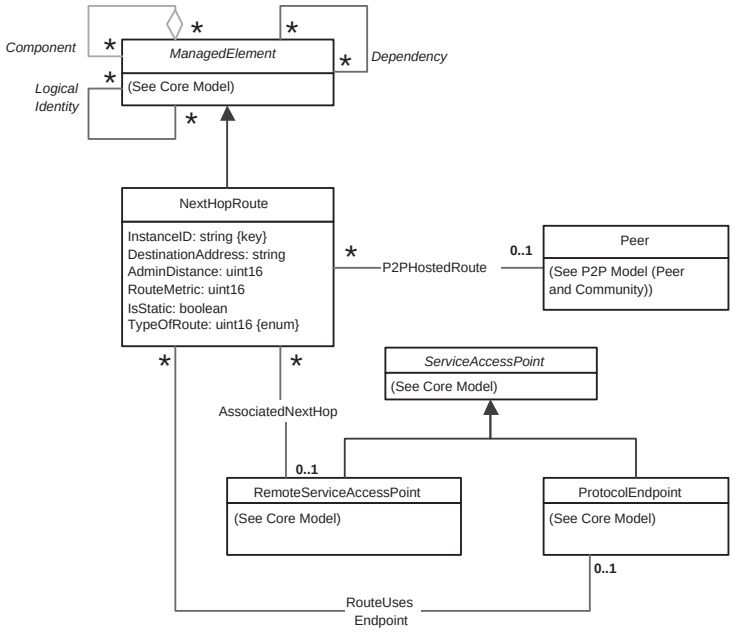


FIG. A.3 – Diagramme de classes CIM pour les tables de routage P2P

L'interface par lequel les messages sont transférés est représentée par le biais de la classe CIM_ProtocolEndpoint qui est liée à la classe CIM_NextHopRoute à travers la classe d'association CIM_RouteUsesEndpoint. Le prochain saut à joindre pour atteindre la destination est modélisé par la classe d'association CIM_AssociatedNextHop qui relie la classe CIM_NextHopRoute à la classe CIM_RemoteServiceAccessPoint.

Etant donné ce modèle, nous avons mis en relation une table de routage, représenté par chacune de ses entrées, avec le pair qui l'héberge. Pour cela, nous avons ajouté la classe d'association P2P_P2PHostedRoute qui relie la classe CIM_NextHopRoute à la classe P2P_Peer, sachant qu'une entrée de table appartient éventuellement à un pair donné[2], et qu'un pair peut héberger zéro ou plusieurs entrées d'une table de routage.

[2]pour pouvoir utiliser cette classe hors du contexte P2P

Annexe B

Spécification du modèle de l'information générique dans le formalisme MOF

B.1 Fichier principal

```
// ==================================================================
// The elements of the Peer and Community model of the P2P CIM
// extension.
// ==================================================================

// ==================================================================
// Elements of the Core and Common models
// ==================================================================
#pragma include ("Core27.mof")
#pragma include ("System27.mof")
#pragma include ("Network27_Pipes.mof")

// ==================================================================
// The P2P Extension
// ==================================================================
#pragma include ("pac.mof")
#pragma include ("svc.mof")
#pragma include ("res.mof")

// ==================================================================
// The JXTA Extension
// ==================================================================
#pragma include ("jxta.mof")
```

B.2 Fichier relatif au modèle de l'organisation

```
// ==================================================================
// The elements of the organization model of the P2P CIM extension.
// ==================================================================

// ==================================================================
```

```
// Peer
// ==================================================================
[Version ("1.0.0"), Description (
    "This class represents a peer in a P2P application.") ]
class P2P_Peer : CIM_EnabledLogicalElement {

   [Key, MaxLen (256), Description (
"CreationClassName indicates the name of the class or the "
"subclass used in the creation of an instance. When used "
"with the other key properties of this class, this property "
"allows all instances of this class and its subclasses to "
"be uniquely identified.") ]
    string CreationClassName;

    [Key, MaxLen (256), Description (
"The identifier of the peer in the context of the executed "
"application.") ]
    string PeerId;

    [Override ("Name"), MaxLen (256), Description (
"The Name property identifies the peer which is managed."
"Additionnal information is located in the object's"
"Description property.") ]
    string Name;

    [Description (
"The ArrivalTime property indicates the date when the "
"managed peer started the P2P application.") ]
    datetime ArrivalTime;
};

// ==================================================================
// TopologicalLink
// ==================================================================
[Association, Version ("1.0.0"), Description (
    "P2P_TopologicalLink is a  association used to establish "
    "a topological relationships between Peers.") ]
class P2P_TopologicalLink : CIM_Dependency {

    [Override ("Antecedent"), Description (
"Antecedent represents the peer which own the topological "
"link") ]
    P2P_Peer REF Antecedent;

    [Override ("Dependent"), Description (
"Dependent represents the peer which is referenced in a "
"topological link") ]
    P2P_Peer REF Dependent;

    [Description ("IsSymetric is true if the relation is the "
"topological relation is symetric.") ]
    boolean IsSymetric;

    [Description (
```

```
        "The Description property provides a textual description of "
        "the object.") ]
    string Description;
};

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

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

    [Override ("Name"), MaxLen (256), Description (
        "The Name property identifies the community which is "
"managed. Additionnal information is localed in the "
"object's Description property.") ]
    string Name;
};

// ==================================================================
// ParticipatingPeers
// ==================================================================
[Association, Version ("1.0.0"), Description (
    "A ParticipatingPeers association represents the gathering of "
    "peers into communities.") ]
class P2P_ParticipatingPeers : CIM_Dependency {

    [Override ("Antecedent"), Min (1), Description (
"Antecedent represents the communities a peer belongs to. "
"link") ]
    P2P_Community REF Antecedent;

    [Override ("Dependent"), Min (1), Description (
"Dependent represents the set of peer belonging to a "
"particular community.") ]
    P2P_Peer REF Dependent;

};

// ==================================================================
// HostedPeer
// ==================================================================
[Association, Version ("1.0.0"), Description (
    "A HostedPeer association represents the link between a peer"
    "and the computer system which hosts it.") ]
class P2P_HostedPeer : CIM_Dependency {
    [Override ("Antecedent"), Description (
        "Antecedent represents the computer system which hosts "
        "peers.") ]
```

```
    CIM_ComputerSystem REF Antecedent;

    [Override ("Dependent"), Description (
        "Dependent represents the peer hosted on a particular "
        "computer system.") ]
    P2P_Peer REF Dependent;
};
```

B.3 Fichier relatif au modèle de services

```
// ==================================================================
// The elements of the Service model of the P2P CIM extension.
// ==================================================================

// ==================================================================
// P2PService
// ==================================================================
[Abstract, Version ("1.0"), Description (
    "The P2P_P2PService class represents a P2P service as the "
    "aggregation of local instances executed by peers. In other "
    "words, it represents a service in a global way.") ]
class P2P_P2PService : CIM_Service {
};

// ==================================================================
// LocalP2PService
// ==================================================================
[Abstract, Version ("1.0"), Description (
    "The P2P_LocalP2PService class represents a P2P service as a "
    "local instance executed by a particular peer. In other "
    "words, it represents a service in a local way.") ]
class P2P_LocalP2PService : P2P_P2PService {
};

// ==================================================================
// P2PServiceAccessPoint
// ==================================================================
[Abstract, Version ("1.0"), Description (
    "P2P_P2PServiceAccessPoint has a semantic which is similar to "
    "CIM_ServiceAccessPoint but it enables a clear separation "
    "between CIM services as defined in the core model and "
    "P2P services.") ]
class P2P_P2PServiceAccessPoint : CIM_ServiceAccessPoint {
};

// ==================================================================
// PeerServiceAccessPoint
// ==================================================================
[Version ("1.0"), Description (
    "P2P_PeerServiceAccessPoint defines an access point to a P2P "
    "service by the use of a peer identifier.") ]
class P2P_PeerServiceAccessPoint : P2P_P2PServiceAccessPoint {
```

```
    [MaxLen (256), Description (
        "The identifier of a peer, as defined in the P2P_Peer class "
"which enables the access of a P2P service.") ]
    string PeerId;
};

// ===================================================================
// P2PServiceAccessBySAP
// ===================================================================
[Association, Version ("1.0"), Description (
    "P2P_ P2PServiceAccessBySAP has a semantic which is similar to "
    "CIM_ServiceAccessBySAP but it enables a clear separation "
    "between the latter class as defined in the core model and the "
    "one defined in this model.") ]
class P2P_P2PServiceAccessBySAP : CIM_ServiceAccessBySAP {

    [Override ("Antecedent") Min(1), Description (
        "The P2P Service.") ]
    P2P_P2PService REF Antecedent;

    [Override ("Dependent"), Min(1), Description (
        "An Access Point for a P2P Service. Access points are "
"dependent in this relationship since they have no function "
        "without a corresponding Service.") ]
    P2P_P2PServiceAccessPoint REF Dependent;
};

// ===================================================================
// LocalServiceComponent
// ===================================================================
[Association, Aggregation, Version ("1.0"), Description (
    "P2P_LocalServiceComponent links a local service instance "
    "executed on a particular peer to a global service which "
    "applies to a community.") ]
class P2P_LocalServiceComponent : CIM_Component {
    [Override ("GroupComponent"), Aggregate, Min(1), Max(1),
        Description ("The global Service.") ]
    P2P_P2PService REF GroupComponent;

    [Override ("PartComponent"), Description (
        "The local component Service.") ]
   P2P_LocalP2PService REF PartComponent;
};

// ===================================================================
// P2PServiceParticipatingPeers
// ===================================================================
[Association, Version ("1.0"), Description (
    "The P2P_P2PServiceParticipatingPeers links a service to the "
    "peers which participate to it. The link is redundant with the "
    "P2P_LocalServiceComponent and HostedLocalP2PService "
    "association classes.") ]
class P2P_P2PServiceParticipatingPeers {
```

```
    [Key, Description ("The P2P Service.") ]
    P2P_P2PService REF Service;

    [Key, Description ("The peers which participate to the "
        "service.") ]
    P2P_Peer REF Peer;
};

// ==================================================================
// HostedP2PService
// ==================================================================
[Association, Version ("1.0"), Description (
    "P2P_HostedP2PService links a service to the community it is "
    "related to.") ]
class P2P_HostedP2PService : CIM_HostedService {

    [Override ("Antecedent"), Max (1), Min (1),
        Description ("The hosting community.") ]
    P2P_Community REF Antecedent;

    [Override ("Dependent"), Weak,
        Description ("The P2P Service hosted in the community. "
        "The service is weak to the community because several "
        "identical services can be instanciated in different "
        "communities. The weak association enables to distinguish "
        "them.") ]
    P2P_P2PService REF Dependent;
};

// ==================================================================
// HostedLocalP2PService
// ==================================================================
[Association, Version ("1.0"), Description (
    "P2P_HostedLocalP2PService links a local service to the peer it "
    "is hosted on.") ]
class P2P_HostedLocalP2PService : CIM_HostedService {

    [Override ("Antecedent"),
        Description ("The hosting peer.") ]
    P2P_Peer REF Antecedent;

    [Override ("Dependent"), Weak,
        Description ("The local P2P Service hosted on a peer. "
        "The local service is weak to the community because "
        "several identical services can be instanciated on "
        "different peers. The weak association enables to "
        "distinguish them.") ]
    P2P_LocalP2PService REF Dependent;
};

// ==================================================================
// HostedP2PAccessPoint
// ==================================================================
```

```
[Association, Version ("1.0"), Description (
    "P2P_HostedP2PAccessPoint links the different access points a "
    "community owns in order to access its services.") ]
class P2P_HostedP2PAccessPoint : CIM_HostedAccessPoint {

    [Override ("Antecedent"), Max (1), Min (1),
        Description ("The hosting community.") ]
    P2P_Community REF Antecedent;

    [Override ("Dependent"), Weak,
        Description ("The P2P accesspoint hosted in the
        "community.") ]
    P2P_P2PAccessPoint REF Dependent;
};
```

B.4 Fichier relatif au modèle de communication

```
// ===================================================================
// The elements of the Communication model of the P2P CIM extension.
// ===================================================================

// ===================================================================
// P2PPipe
// ===================================================================
[Version ("1.0"), Description (
    "P2P_P2PPipe has a semantic which is similar to "
    "CIM_NetworkPipe but it enables a clear separation "
    "between the latter class as defined in the core model and "
    "the one defined in this model.") ]
class P2P_P2PPipe : CIM_NetworkPipe {

};

// ===================================================================
// PeerUsesPipe
// ===================================================================
[Association, Version ("1.0"), Description (
    "The P2P_PeerUsesPipe links a peer to the different pipes it "
    "uses. A P2P pipe follows the semantic defined in the Network "
    "Common model and thus is binded to two peers.") ]
class P2P_PeerUsesPipe {

    [Key, Min(2), Max(2), Description (
        "The Peer which uses pipes.") ]
    P2P_Peer REF Peer;

    [Key, Description ("The pipes used by a peer.") ]
    P2P_P2PPipe REF Pipe;
};

// ===================================================================
// HostedP2PPipe
// ===================================================================
```

```
[Association, Version ("1.0"), Description (
    "The P2P_Hosted links pipes to the community in which it is "
    "used.") ]
class P2P_HostedP2PPipe {

    [Key, Weak, Description (
        "A pipe hosted in a community.") ]
    P2P_P2PPipe REF Peer;

    [Key, Min(1), Max(1), Description (
        "The community which hosts pipes.") ]
    P2P_Community REF Pipe;
};
```

B.5 Fichier relatif au modèle de ressources

```
// ==================================================================
// The elements of the resource model of the P2P CIM extension.
// ==================================================================

// ==================================================================
// PeerResource
// ==================================================================
[Abstract, Version ("1.0.0"), Description (
    "This class represents a resource in a P2P application.") ]
class P2P_PeerResource : CIM_LogicalElement {

   [Key, MaxLen (256), Description (
"CreationClassName indicates the name of the class or the "
"subclass used in the creation of an instance. When used "
"with the other key properties of this class, this property "
"allows all instances of this class and its subclasses to "
"be uniquely identified.") ]
    string CreationClassName;

    [Override ("Name"), MaxLen (256), Description (
"The Name property identifies the resource which is "
" managed. Additionnal information is located in the "
"object's Description property.") ]
    string Name;

    [Key, MaxLen (256), Description (
"The identifier of the resource in the context of the "
"executed application.") ]
    string ResourceId;

    [Description (
        "HashMethod is an integer enumeration that indicates the "
        "method used to generate the ResourceId of the resource." ),
        ValueMap {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9",
            "10..65535" },
        Values {"Unknown", "Other", "MD4", "MD5", "SHA-0",
            "SHA-1", "SHA-224", "SHA-256", "SHA-384", "SHA-512",
```

```
            "Vendor Reserved" },
        ModelCorrespondence {
            "P2P_PeerResource.OtherHashMethod"} ]
    uint16 HashMethod;

     [Description (
        "A string describing the hash method when the "
        "HashMethod property is set to 1 (\"Other\"). This property "
        "MUST be set to NULL when HashMethod is any value other "
        "than 1."),
        ModelCorrespondence {"P2P_PeerResource.HashMethod"} ]
    string OtherHashMethod;

   [Description (
        "State is an integer enumeration that indicates the "
        "state of the resource." ),
        ValueMap {"0", "1", "2", "3", "4", "5..65535" },
        Values {"Unknown", "Other", "Available", "Unavailable",
            "Corrupted", "Vendor Reserved" },
        ModelCorrespondence {
            "P2P_PeerResource.OtherState"} ]
    uint16 State;

    [Description (
        "A string describing the resource's state when the "
        "State property is set to 1 (\"Other\"). This property "
        "MUST be set to NULL when State is any value other "
        "than 1."),
        ModelCorrespondence {"P2P_PeerResource.State"} ]
    string OtherState;
};

// ==================================================================
// ServiceUsesResource
// ==================================================================
[Association, Version ("1.0"), Description (
    "The P2P_ServiceUsesResource links resources to services which "
    "uses them.") ]
class P2P_ServiceUsesResource {

    [Key, Min(1), Description ("The service which uses a "
        "resource.") ]
    P2P_P2PService REF Service;

    [Key, Description ("The resource used in a service.") ]
    P2P_PeerResource REF Resource;
};

// ==================================================================
// PeerSharesResource
// ==================================================================
[Association, Version ("1.0"), Description (
    "The P2P_PeerSharesResource links resources to peers which "
    "shares and thus own them. In a P2P application, a single "
```

```
    "resource can be duplicated and be shared by several peers. "
    "This is why the cardinality of this association is *-*.") ]
class P2P_PeerSharesResource {

    [Key, Description ("The peer which shares a "
        "resource.") ]
    P2P_Peer REF Peer;

    [Key, Description ("The resource shared by peers.") ]
    P2P_PeerResource REF Resource;
};

// ==================================================================
// PeerSharesResource
// ==================================================================
[Association, Version ("1.0"), Description (
    "The P2P_PeerSharesResource links resources to peers which "
    "shares and thus own them. In a P2P application, a single "
    "resource can be duplicated and be shared by several peers. "
    "This is why the cardinality of this association is *-*.") ]
class P2P_PeerSharesResource {

    [Key, Description ("The peer which shares a resource.") ]
    P2P_Peer REF Peer;

    [Key, Description ("The resource shared by peers.") ]
    P2P_PeerResource REF Resource;
};

// ==================================================================
// PeerHostsResource
// ==================================================================
[Association, Version ("1.0"), Description (
    "The P2P_PeerHostsResource links resources to peers which "
    "hosts them.") ]
class P2P_PeerHostsResource {

    [Key, Description ("The peer which hosts a resource.") ]
    P2P_Peer REF Peer;

    [Key, Description ("The resource hosted by peers.") ]
    P2P_PeerResource REF Resource;
};

// ==================================================================
// ResourceComposition
// ==================================================================
[Association, Aggregation, Version ("1.0"), Description (
    "P2P_ResourceComposition allow the composition of resource "
    "into a aggregated one. Typically, it can used to represent "
    "files and chunks which are distributed in a community.") ]
class P2P_ResourceComposition : CIM_Component {
    [Override ("GroupComponent"), Aggregate, Min(1), Max(1),
        Description ("The aggregated resource.") ]
```

```
    P2P_PeerResource REF GroupComponent;

    [Override ("PartComponent"), Description (
        "The element resource of the aggregation.") ]
   P2P_PeerResource REF PartComponent;
};

// ==================================================================
// SharedStorageMemory
// ==================================================================
[Version ("1.0.0"), Description (
    "This class represents a memory space shared in a P2P "
    "application.") ]
class P2P_SharedStorageMemory : P2P_PeerResource {

    [Description (
        "CurrentSize represents the total size used by resource "
        "located in the shared space. Of course, this value has not "
        "to exceed the TotalAuthorizedSize one." )]
    uint64 CurrentSize;

    [Description (
        "TotalAuthorizedSize represents the maximum space that "
        "resources can use in this shared space. This value can "
        "be set by the user or by any service-dependent policy." )]
    uint64 TotalAuthorizedSize;
};

// ==================================================================
// SharedLogicalFile
// ==================================================================
[Version ("1.0.0"), Description (
    "This class represents a file shared in a P2P application. "
    "Access rights are given in the context of the P2P application "
    "and can be different from these of the local file system.") ]
class P2P_SharedLogicalFile : P2P_PeerResource {

    [Description (
        "Boolean indicating that the File can be read.") ]
    boolean Readable;

    [Description (
        "Boolean indicating that the File can be written.") ]
    boolean Writeable;

    [Description (
        "Indicates the file is executable.") ]
    boolean Executable;
};

// ==================================================================
// SharedFileInMemorySpace
// ==================================================================
[Association, Aggregation, Version ("1.0"), Description (
```

```
    "P2P_ResourceComposition allow the composition of resource "
    "into a aggregated one. Typically, it can used to represent "
    "files and chunks which are distributed in a community.") ]
class P2P_SharedFileInMemorySpace : CIM_Component {
    [Override ("GroupComponent"), Aggregate, Min(1), Max(1),
        Description ("The aggregated resource.") ]
    P2P_SharedStorageMemory REF GroupComponent;

    [Override ("PartComponent"), Description (
        "The element resource of the aggregation.") ]
   P2P_SharedLogicalFile REF PartComponent;
};

// ===================================================================
// SharedProcessingPower
// ===================================================================
[Version ("1.0.0"), Description (
    "This class represents a file shared in a P2P application. "
    "Access rights are given in the context of the P2P application "
    "and can be different from these of the local file system.") ]
class P2P_SharedProcessingPower : P2P_PeerResource {

    [Description (
        "indicates the maximum load the related service can use. "
        "This value is expressed as a percentage."
    uint8 SharedLoadPercentage;

    [Description (
        "Indicates that the service requiring processing power must "
        "be executed when the host system is idle.") ]
    boolean UsedInIdleTime;

    [Description (
        "Indicates that the service requiring processing power must "
        "be exectuted during the time constraints described in the "
        "TimeConstraintsDescription property.") ]
    boolean UseTimeConstraints;

    [Description (
        "A textual description of the time intervals in which the "
        "shared processing power can ba used in a P2P application. "
        "Currently, there is no syntax which describes these time "
        "constraints. But, if UseTimeConstraints is false, this "
        "attribute must be set to null.") ]
    string TimeConstraintsDescription;
};

// ===================================================================
// SharedBandwidth
// ===================================================================
[Version ("1.0.0"), Description (
    "This class represents the bandwidth shared by a peer in the "
    "context of a P2P application.") ]
class P2P_SharedBandwidth : P2P_PeerResource {
```

```
    [Description (
        "Maximum upload bandwidth allowed for share.") ]
    uint8 UploadPercentage;

    [Description (
        "Maximum download bandwidth load allowed for share.") ]
    uint8 DownloadPercentage;
};

// ==================================================================
// SharedMemoryRootPath
// ==================================================================
[Association, Version ("1.0"), Description (
    "SharedMemoryRootPath represents one or several root path on "
    "the hosting file system of the shared memory space.") ]
class P2P_SharedMemoryRootPath {

    [Key, Min(1), Max(1), Description (
        "A shared memory space.") ]
    P2P_SharedStorageMemory REF SharedSpace;

    [Key, Description (
        "The root path.") ]
    CIM_Directory REF Directory;
};

// ==================================================================
// FileSystemSharesMemory
// ==================================================================
[Association, Version ("1.0"), Description (
    "FileSystemSharesMemory represents the file system which hosts "
    "a shared memory space.") ]
class P2P_FileSystemSharesMemory {

    [Key, Description (
        "A shared memory space.") ]
    P2P_SharedStorageMemory REF SharedSpace;

    [Key, Min(1), Max(1), Description (
        "The hosting file system.") ]
    CIM_FileSystem REF FileSystem;
};

// ==================================================================
// RepresentsLogicalFile
// ==================================================================
[Association, Version ("1.0"), Description (
    "RepresentsLogicalFile links a file shared in a P2P application "
    "to the one hosted on a file system.") ]
class P2P_RepresentsLogicalFile {

    [Key, Min(0), Max(1), Description (
        "A file shared in a P2P application.") ]
```

```
    P2P_SharedLogicalFile REF SharedSpace;

    [Key, Min(0), Max(1), Description (
        "A file hosted on a file system.") ]
    CIM_LogicalFile REF FileSystem;
};
```

B.6 Fichier relatif au modèle de routage

```
// ==================================================================
// The elements of the Routing model of the P2P CIM extension.
// The classes defined in this scheme do not contain any particular
// property. Their goal is just to enable a clear separation between
// CIM classes and P2P classes.
// ==================================================================

// ==================================================================
// P2PRouteCalculationService
// ==================================================================
[Version ("1.0"), Description (
    "The P2P_P2PRouteCalculationService class represents a global "
    "P2P route calculation service.") ]
class P2P_P2PRouteCalculationService : P2P_P2PService {
};

// ==================================================================
// P2PForwardingService
// ==================================================================
[Version ("1.0"), Description (
    "The P2P_P2PForwardingService class represents a global P2P "
    "forwarding service.") ]
class P2P_P2PForwardingService : P2P_P2PService {
};

// ==================================================================
// P2PLocalRouteCalculationService
// ==================================================================
[Version ("1.0"), Description (
    "The P2P_P2PLocalRouteCalculationService class represents a "
    "local P2P route calculation service.") ]
class P2P_P2PLocalRouteCalculationService : P2P_P2PLocalService {
};

// ==================================================================
// P2PLocalForwardingService
// ==================================================================
[Version ("1.0"), Description (
    "The P2P_P2PLocalForwardingService class represents a local P2P "
    "forwarding service.") ]
class P2P_P2PForwardingService : P2P_P2PLocalService {
};
```

```
// ==================================================================
// P2PCalculatesAmong
// ==================================================================
[Association, Version ("1.0"), Description ("see the "
    "CalculatesAmong class from the CIM network common model.") ]
class P2P_P2PCalculatesAmong {

    [Min(1), Description (
        "The P2P route calculation service.") ]
    P2P_P2PLocalRouteCalculationService REF RouteCalculationService;

    [Description (
        "The P2P service access point.") ]
    P2P_P2PServiceAccessPoint REF AccessPoint;
};

// ==================================================================
// P2PForwardsAmong
// ==================================================================
[Association, Version ("1.0"), Description ("see the "
    "ForwardsAmong class from the CIM network common model.") ]
class P2P_P2PForwardsAmong {

    [Min(1), Description (
        "The P2P forwarding service.") ]
    P2P_P2PLocalForwardingService REF ForwardingService;

    [Description (
        "The P2P service access point.") ]
    P2P_P2PServiceAccessPoint REF AccessPoint;
};

// ==================================================================
// CommunityRoutingProtocol
// ==================================================================
[Association, Version ("1.0"), Description (
    "P2P_CommunityRoutingProtocol links a community to a particular "
    "routing protocol domain and thus defines the routing protocol "
    "used in the community") ]
class P2P_CommunityRoutingProtocol {

    [Min(1), Description (
        "The domain of the routing protocol.") ]
    CIM_RoutingProtocolDomain REF Domain;

    [Description (
        "The community which uses the protocol.") ]
    P2P_Community REF Community;
};

// ==================================================================
// P2PHostedRoute
// ==================================================================
[Association, Version ("1.0"), Description (
```

```
    "P2P_P2PHostedRoute links a route table entry to the peer which "
    "hosts it.") ]
class P2P_P2PHostedRoute {

    [Description (
        "The P2P routing table entry.") ]
    CIM_NextHopRoute REF Domain;

    [Min(0), Max(1), Description (
        "The peer which hosts the routing table entry.") ]
    P2P_Peer REF Peer;
};
```

Annexe C

Spécification du modèle de l'information pour JXTA dans le formalisme MOF

C.1 Fichier principal

```
// ==================================================================
// Extensions for the Peer and Community sub-model
// ==================================================================

// ==================================================================
// JxtaPeerGroup
// ==================================================================
[Version ("1.0.0"), Description (
     "Models the JXTA concept of peergroup.") ]
class JXTA_JxtaPeerGroup : P2P_Community {

};

// ==================================================================
// JxtaPeer
// ==================================================================
[Version ("1.0.0"), Description (
    "Models the JXTA concept of peer.") ]
class JXTA_JxtaPeer : P2P_Peer {

    [Description (
        "The IsManageable is set to true if the peer host an "
        "agent.") ]
    boolean IsManageable;

};

// ==================================================================
// JxtaParticipatingPeer
// ==================================================================
[Version ("1.0.0"), Description (
    "Models the participation of a peer within a peergroup.") ]
```

```
class JXTA_JxtaParticipatingPeer : P2P_ParticipatingPeers {

    [Description (
        "The role of the peer within the peergroup : Edge, "
        "Rendezvous and/or Relay.") ]
    string State;

    [Description (
        "The IsRelay attribute is set to true if the considered "
        "peer is acting as an relay peer within the peergroup.") ]
    boolean IsRelay;

    Description (
        "Is set to true if the considered peer is acting as an "
        "rendezvous peer within the peergroup.") ]
    boolean IsRendezvous;
};

// ==================================================================
// RendezvousConnection
// ==================================================================
[Version ("1.0.0"), Description (
    "Models the connection of a peer to a rendezvous peer.") ]
class JXTA_RendezvousConnection : P2P_TopologicalLink {

    [Override ("Antecedent"), Description (
        "The rendezvous peer which the peer used.") ]
    JXTA_JxtaPeer REF Antecedent;

    [Override ("Dependent"), Description (
        "The peer which uses the rendezvous peer.") ]
    JXTA_JxtaPeer REF Dependent;

    [Description (
        "Contains the PeerID of the rendezvous peer which the peer "
        "is connected.") ]
    string PeerID;

    [Description (
        "Contains the current connection state of the peer to the "
        "rendezvous peer : Connecting, Connected or Disconnected.") ]
    string State;

    datetime TransitionTime;

    [Description (
        "Contains the time period during the peer is granted to "
        "connect to the rendezvous.") ]
    uint64 Lease;

    uint64 BeginConnectionTime;

    Description (
        "Is set to true if the peer is connected on the rendezvous "
```

```
        "peer.") ]
    boolean Connected;

    Description (
        "Is set to true if the peer is connecting on the rendezvous "
        "peer.") ]
    boolean Connecting;

    uint64 TimeConnected;

    Description (
        "Is set to true if the peer has been disconnected from the "
        "rendezvous peer.") ]
    boolean Disconnected;

    uint64 DisconnectTime;

    uint32 NumConnectionsBegun;

    uint32 NumConnectionsEstablished;

    [Description (
        "Contains how many times the connection has been refused from "
        "the rendezvous peer.") ]
    uint32 NumConnectionsRefused;

    [Description (
        "Contains the amount of time elapsed between the request of "
        "connection and the acquitment from the rendezvous peer.") ]
    uint64 TotalTimesToConnect;

    [Description (
        "Contains the amount of time elapsed since the peer had "
        "renewed its lease with the rendezvous.") ]
    uint64 LastLeaseRenewalTime;

    [Description (
        "Contains the number of time a peer had renewed its lease "
        "with the rendezvous.") ]
    uint64 NumLeaseRenewals;

    uint32 NumDisconnects;

    uint64 TotalTimeConnected;

    datetime TimeConnectionEstablished;
};

// ===================================================================
// RelayConnection
// ===================================================================
[Version ("1.0.0"), Description (
    "Models the connection of a peer to a relay peer.") ]
class JXTA_RelayConnection : P2P_TopologicalLink {
```

```
    [Override ("Antecedent"), Description (
        "The relay peer which the peer used.") ]
    JXTA_JxtaPeer REF Antecedent;

    [Override ("Dependent"), Description (
        "The peer which uses the relay peer.") ]
    JXTA_JxtaPeer REF Dependent;
};

// ==================================================================
// RendezvousPeerView
// ==================================================================
[Version ("1.0.0"), Description (
    "this peer which is connected to a rendezvous peer ") ]
class JXTA_RendezvousPeerView : P2P_TopologicalLink {

    [Override ("Antecedent"), Description (
        "A rendezvous peer used by the rendezvous peer in its "
        "Rendezvous Peer View.") ]
    JXTA_JxtaPeer REF Antecedent;

    [Override ("Dependent"), Description (
        "The rendezvous peer which use an other rendezvous peer in "
        "its Rendezvous Peer View.") ]
    JXTA_JxtaPeer REF Dependent;
};

// ==================================================================
// Extensions for the P2P Services sub-model
// ==================================================================

// ==================================================================
// JxtaPeerGroupService
// ==================================================================
[Version ("1.0.0"), Description (
    "Models a service which is provided by the entire peergroup.") ]
class JXTA_JxtaPeerGroupService : P2P_P2PService {

    [Key, Description (
        "Contains the unique identification number of the peergroup "
        "service.") ]
    string ModuleSpecID;

    [Description (
        "Contains the name of the Java package which implements the "
        "peergroup service.") ]
    string Code;

    [Override("PrimaryOwnerContact"), Description (
        "Contains the way to contact the provider of the peergroup "
        "service.") ]
    string PrimaryOwnerContact;
```

```
    [Override("PrimaryOwnerName"), Description (
        "Contains the name of the provider of the peergroup "
        "service.") ]
     string PrimaryOwnerName;
};

// ==================================================================
// JxtaLocalPeerGroupService
// ==================================================================
[Version ("1.0.0"), Description (
    "Models the part of the peergroup service that is host on a "
    "peer.") ]
class JXTA_JxtaLocalPeerGroupService : P2P_LocalP2PService {

    [Key, Description (
        "Contains the identification number of the peergroup "
        "service which the local part belongs to.") ]
    string ModuleSpecID;

    [Weak, Description (
        "Contains the PeerID of the peer which hosts the part of "
        "the peergroup service.") ]]
    string PeerId;

    [Description (
        "Contains the name of the Java package which implements "
        "the peergroup service.") ]
    string Code;

    [Override("PrimaryOwnerContact"), Description (
        "Contains the way to contact the provider of the peergroup "
        "service.") ]
    string PrimaryOwnerContact;

    [Override("PrimaryOwnerName"), Description (
        "Contains the name of the provider of the peergroup "
        "service.") ]
     string PrimaryOwnerName;
};

// ==================================================================
// JxtaPeerService
// ==================================================================
[Version ("1.0.0"), Description (
    "Models a service which is provided by a peer in particular.") ]
class JXTA_JxtaPeerService : P2P_LocalP2PService {

    [Key, Weak, Description (
        "Contains the PeerID of the peer which provides this "
        "service.")]
    string PeerId;
};

// ==================================================================
```

```
// JxtaLocalRendezvousService
// ===========================================================================
[Version ("1.0.0"), Description (
    "Model the local part of the core Rendezvous service. There are "
    "metrics from MMP too.") ]
class JXTA_JxtaLocalRendezvousService : JXTA_JxtaLocalPeerGroupService {

    [Description (
        "Contains the current state of the peer which host the part "
        "of Rendezvous service : Edge or Rendezvous.") ]
    string State;

    [Description (
        "Indicates when the last transition happens.") ]
    datetime TransitionTime;

    [Description (
        "Is set to true if the peer is acting as a edge, false "
        "otherwise.") ]
    boolean Edge;

    [Description (
        "Is set to true if the peer is acting as a rendezvous, "
        "false otherwise.") ]
    boolean Rendezvous;

    [Description (
        "Time which the peer begins to act as an edge.") ]
    datetime EdgeStartTime;

    [Description (
        "Total amount of time the peer has been acted as an edge.") ]
    uint64 TotalEdgeTime;

    [Description (
        "Number of times the peer swap from rendezvous to edge.") ]
    uint32 NumEdgeTransitions;

    [Description (
        "Time which the peer begins to act as a rendezvous.") ]
    datetime RendezvousStartTime;

    [Description (
        "Total amount of time the peer has been acted as a
        "rendezvous.") ]
    uint64 TotalRendezvousTime;

    [Description (
        "Number of times the peer swap from edge to rendezvous.") ]
    uint64 NumRendezvousTransitions;

    uint32 NumReceivedProcessedLocally;
    uint32 NumReceivedRepropagatedInGroup;
    uint32 NumReceivedInvalid;
```

```
    uint32 NumReceivedDead;
    uint32 NumReceivedLoopback;
    uint32 NumReceivedDuplicate;
    uint32 TotalReceivedUndelivered;
    uint32 TotalReceived;
    uint32 NumPropagated;
    uint32 NumFailedPropagating;
    uint32 NumRepropagated;
    uint32 NumFailedRepropagating;
    uint32 NumPropagatedToPeers;
    uint32 NumFailedPropagatingToPeers;
    uint32 NumPeersPropagatedTo;
    uint32 NumPropagatedInGroup;
    uint32 NumPropagatedToNeighbors;
    uint32 NumWalks;
    uint32 NumFailedWalks;
    uint32 NumWalkedToPeers;
    uint32 NumFailedWalkToPeers;
    uint32 NumPeersWalkedTo;

    [Description (
        "Amount of time the peer is acting like edge without "
        "swap.") ]
    uint64 TimeAsEdge;

    [Description (
        "Amount of time the peer is acting like rendezvous without "
        "swap.") ]
    uint64 TimeAsRendezvous;
};

// ===================================================================
// JxtaLocalPipeService
// ===================================================================
[Version ("1.0.0"), Description (
    "Models the local part of the core Pipe service. There are "
    "metrics too.") ]
class JXTA_JxtaLocalPipeService : JXTA_JxtaLocalPeerGroupService {

    [Description (
        "Contains the starting date of the service on the peer.") ]
    datetime StartTime;

    [Description (
        "Contains the amount of time the service is running on the "
        "peer.") ]
    uint64 UpTime;

    [Description (
        "Contains the amount of InputPipes succesfully created.") ]
    uint32 NumInputPipesCreated;

    [Description (
        "Contains the amount of OutputPipes succesfully created.") ]
```

```
    uint32 NumOutputPipesCreated;

    [Description (
        "Contains the amount of InputPipes succesfully closed.") ]
    uint32 NumInputPipesClosed;

    [Description (
        "Contains the amount of OutputPipes succesfully closed.") ]
    uint32 NumOutputPipesClosed;

    [Description (
        "Contains the amount of InputPipes failed to create.") ]
    uint32 NumInputPipesFailedToCreate;

    [Description (
        "Contains the amount of OutputPipes failed to create.") ]
    uint32 NumOutputPipesFailedToCreate;
};

/ ==================================================================
// JxtaLocalEndpointService
// ==================================================================
[Version ("1.0.0"), Description (
    "Models the local part of the core Endpoint service. There are "
    "metrics too from MMP.") ]
class JXTA_JxtaLocalEndpointService : JXTA_JxtaLocalPeerGroupService {

    datetime EndpointStartTime;
    uint64 EndpointUpTime;
    uint32 DemuxMessageProcessed;
    uint32 IncomingMessageSentToEndpointListener;
    uint32 InvalidIncomingMessage;
    uint32 NoListenerForIncomingMessage;
    uint32 ErrorProcessingIncomingMessage;
    uint32 NoDestinationAddressForDemuxMessage;
    uint32 NoSourceAddressForDemuxMessage;
    uint32 DiscardedLoopbackDemuxMessage;
    uint32 IncomingMessageFilteredOut;
};

// ==================================================================
// JxtaLocalResolverService
// ==================================================================
[Version ("1.0.0"), Description (
    "Models the local part of the core Resolver service. There are "
    "metrics too from MMP.") ]
class JXTA_JxtaLocalResolverService : JXTA_JxtaLocalPeerGroupService {

    uint32 NumInvalidSrdiMessages;
    uint32 NumSrdiMessagesToUnknownHandler;
    uint32 NumInvalidResponses;
    uint32 NumResponsesToUnknownHandler;
    uint32 NumInvalidQueries;
    uint32 NumQueriesToUnknownHandler;
```

```
};

// ==================================================================
// JxtaLocalMembershipService
// ==================================================================
[Version ("1.0.0"), Description (
    "Models the local part of the core Memebership service.") ]
class JXTA_JxtaLocalMembershipService : JXTA_JxtaLocalPeerGroupService {
};

// ==================================================================
// JxtaLocalDiscoveryService
// ==================================================================
[Version ("1.0.0"), Description (
    "Models the local part of the core Discovery service.") ]
class JXTA_JxtaLocalDiscoveryService : JXTA_JxtaLocalPeerGroupService {
};

// ==================================================================
// JxtaLocalPeerInfoService
// ==================================================================
[Version ("1.0.0"), Description (
    "Models the local part of the core PeerInfo service.") ]
class JXTA_JxtaLocalPeerInfoService : JXTA_JxtaLocalPeerGroupService {
};

// ==================================================================
// JxtaLocalAlwaysAccessService
// ==================================================================
[Version ("1.0.0"), Description (
    "Models the local part of the core AlwaysAccess service.") ]
class JXTA_JxtaLocalAlwaysAccessService : JXTA_JxtaLocalPeerGroupService {
};

// ==================================================================
// JxtaHandler
// ==================================================================
[Abstract, Version ("1.0.0"), Description (
    "Models the JXTA concept of Handler") ]
class JXTA_JxtaHandler : P2P_P2PServiceAccessPoint {

    [Key, Description (
        "Contains the name of the handler which allows to identify it.") ]
    string HandlerName;
};

// ==================================================================
// JxtaQueryHandler
// ==================================================================
[Version ("1.0.0"), Description (
    "Implementation of an JXTA_JxtaHandler. There are metrics from "
    "MMP too.") ]
class JXTA_JxtaQueryHandler : JXTA_JxtaHandler {
```

```
    [Description (
            "Indicates if the handler is registered or unregistered. "
        "If not it do not carry about incoming or outgoing messages.") ]
    string Registered;

    uint32 NumResponses;
    uint64 ResponseProcessingTime;
    uint64 AverageResponseProcessingTime;
    uint64 ResponseTime;
    uint64 AverageResponseTime;
    uint32 NumResponseErrors;
    uint32 NumQueries;
    uint32 NumQueriesRepropagated;
    uint64 QueryProcessingTime;
    uint64 AverageQueryProcessingTime;
    uint32 NumQueryErrors;
    uint32 NumQueriesSentInGroup;
    uint32 NumQueriesSentViaWalker;
    uint32 NumQueriesSentViaUnicast;
    uint32 NumErrorsSendingQueries;
    uint32 NumErrorsPropagatingQueries;
    uint32 NumResponsesSentInGroup;
    uint32 NumResponsesSentViaWalker;
    uint32 NumResponsesSentViaUnicast;
    uint32 NumErrorsSendingResponses;
    uint32 NumErrorsPropagatingResponses;
    uint32 NumQueriesSent;
    uint32 NumQuerySendErrors;
    uint32 NumResponsesSent;
    uint32 NumResponseSendErrors;
    string QueryDestinationMetrics[];
};

// ==================================================================
// JxtaSrdiHandler
// ==================================================================
[Version ("1.0.0"), Description (
    "Implementation of JXTA_JxtaHandler but this one is used with "
    "SRDI. There are metrics from MMP too.") ]
class JXTA_JxtaSrdiHandler : JXTA_JxtaHandler {

    [Description (
        "Indicates if the handler is registered. If not it do not "
        "carry about incoming or outgoing messages.") ]
    boolean Registered;

    uint32 NumProcessed;
    uint32 NumErrorsWhileProcessing;
    uint64 TotalProcessTime;
    uint32 NumMessagesSentViaWalker;
    uint32 NumMessagesSentViaUnicast;
    uint32 NumErrorsSendingMessages;
    uint32 NumErrorsPropagatingMessages;
    string SrdiDestinationMetrics[];
```

```
};

// ==================================================================
// JxtaHandlerForService
// ==================================================================
[Version ("1.0.0"), Description (
    "Bind an handler with the service it working for.") ]
class JXTA_JxtaHandlerForService : P2P_P2PServiceAccessBySAP {

    [Override ("Antecedent"), Description (
        "The required handler which works for the service.") ]
    JXTA_JxtaHandler REF Antecedent;

    [Override ("Dependent"), Description (
        "The Service that is dependent on the handler.") ]
    P2P_P2PService REF Dependent;
};

// ==================================================================
// Extensions for the Communication sub-model
// ==================================================================

// ==================================================================
// JxtaPipeEndpoint
// ==================================================================
[Abstract, Version ("1.0.0"), Description (
    "Models a JXTA PipeEndpoint.") ]
class JXTA_JxtaPipeEndpoint : CIM_ProtocolEndpoint {

    [Key, Weak, Description (
        "The PeerId of the peer which hosts this endpoint.") ]
    string PeerId;

    [Key, Description (
        "The Id of the pipe the PipeEndpoint belongs to.") ]
    string Id;

    [Description (
        "Contains the date which the InputPipeEndpoint has been "
        "bound to the pipe.") ]
    datetime BindingTime;

    [Description (
        "Contains the name of the pipe this InputPipeEndpoint has "
        "been bound with.") ]
    string Name;

    [Description (
        "Contains the type of the pipe : Unicast, Propagated or "
        "SecureUnicast.") ]
    string Type;

    [Description (
        "Indicates if the InputPipeEndpoint has been closed.") ]
```

```
    boolean IsClosed;
};

// ==================================================================
// JxtaInputPipeEndpoint
// ==================================================================
[Version ("1.0.0"), Description (
    "A JXTA InputPipe.") ]
class JXTA_JxtaInputPipeEndpoint : JXTA_JxtaEndpoint {

    [Description (
        "Contains the amount of message received in the pipe with "
        "this InputPipeEnpoint.") ]
    uint64 NumMessageReceived;
};

// ==================================================================
// JxtaOutputPipeEndpoint
// ==================================================================
[Version ("1.0.0"), Description (
    "A JXTA OutputPipe.") ]
class JXTA_JxtaOutputPipeEndpoint : JXTA_JxtaEndpoint {

    [Description (
        "Contains the amount of message sent in the pipe using this "
        "OutputPipeEnpoint.") ]
    uint64 NumMessageSend;
};

// ==================================================================
// JxtaPipe
// ==================================================================
[Version ("1.0.0"), Description (
    "Models a JXTA concept of pipe. A JXTA_JxtaPipe is composed by "
    "differents CIM_NetworkPipe. Attribute AggregationBehavior is "
    "set to Unknown.") ]
class JXTA_JxtaPipe : CIM_NetworkPipe {

    [Description (
        "Contains the name of the JXTA pipe.") ]
    String Name;

    [Description (
        "Contains the type of the pipe : Unicast, Propagated or "
        "SecureUnicast."),
        Values {"JxtaUnicast", "JxtaPropagated", "JxtaSecureUnicast"}]
    String Type;
};

// ==================================================================
// PeerUsesPipe
// ==================================================================
[Version ("1.0.0"), Description (
    "Associate the peer which uses a pipe with the latter.") ]
```

```
class JXTA_PeerUsesPipe : CIM_Dependency {

    [Override ("Antecedent"), Description (
        "The pipe used by the peer.") ]
    CIM_NetworkPipe REF Antecedent;

    [Override ("Dependent"), Description (
        "The peer which uses the pipe.") ]
    JXTA_JxtaPeer REF Dependent;
};

// ===================================================================
// end of file
// ===================================================================
```

Résumé

Rendu populaire par Napster à la fin des années 90, le modèle pair à pair (P2P) est aujourd'hui utilisé dans des environnements contraints comme les entreprises, les administrations ou les universités. Ce modèle permet de repousser les limites induites par le modèle client-serveur. En effet, la décentralisation de ses composants permet d'assurer le passage à l'échelle, la tolérance aux fautes, l'équilibre de la charge et du trafic, et la répartition des coûts des applications qui le mettent en œuvre. Néanmoins, pour pouvoir garantir un niveau de service, ce modèle requiert l'intégration d'une infrastructure de supervision adaptée.

Etant donné ce cadre, nous avons travaillé sur la conception et la mise en œuvre de mécanismes de supervision, fondés sur des approches standard, pour les réseaux et services P2P. Concernant la modélisation de l'information de gestion, nous avons conçu une extension de CIM pour le modèle P2P. Afin de valider cette proposition, nous l'avons implantée sur Jxta. Nous avons développé une infrastructure de supervision composée d'agents de gestion, d'un gestionnaire centralisé et d'une application graphique de gestion. Nous avons ensuite spécialisé notre modèle de l'information pour les tables de hachage distribuées. Nous avons abstrait le fonctionnement de ces infrastructures, proposé un ensemble de métriques qui caractérisent leur performance, et déduit un modèle de l'information qui les intègre.

La seconde partie de notre travail a porté sur le modèle de l'organisation du plan de supervision. Nous avons proposé un modèle hiérarchique, construit et maintenu de manière distribuée, qui permet aux pairs d'organiser le plan de gestion selon une arborescence de gestionnaires et d'agents. Cette proposition a été mise en œuvre sur une implantation de Pastry. Des tests préliminaires d'évaluation ont montré son bon comportement.

Mots-clés: Pair à pair, gestion de réseaux, tables de hachage distribuées, modèle de l'information, CIM, architecture de gestion, Jxta, Pastry

Abstract

The peer-to-peer model has been made famous by the Napster at the end of the 90's and is now used in constraint environments such as enterprises or universities. This model extends the limits of the client/server one. Indeed, the decentralization of its components enables the scalability, the fault tolerance, the load and traffic balance and the costs distribution of all the applications built over it. Nevertheless, in order to ensure a service level, this model requires the integration of an adapted management framework.

Given this context, we worked on the conception and deployment of management mecanisms, based on standard approaches, for P2P networks and services. Concerning the management information model, we designed a CIM extension for the P2P model. In order to validate this proposal, we deployed it over Jxta. It consists in a complete centralised management framework composed of agents, a central manager and a graphical management application. Then, we refined our information model for distributed hash tables. We abstracted the operational behavior of these infrastructures, proposed a set of metrics which feature their performance, and deduced an information model which integrate them.

The second part of our work deals with the organization of the management plan. We proposed a hierarchical model, built and maintained in a distributed way, which enables the organisation of the management plan in a tree composed of manager and agents. Some first evaluation tests have shown its good behavior.

Keywords: Peer-to-peer, network management, distributed hash tables, information model, CIM, management architecture, Jxta, Pastry