Exploitation des protocoles de découverte de services pour l'auto-configuration du plan de gestion

Chahinez Hamlaoui

Université Nancy II

Exploitation des protocoles de découverte de services pour l'auto-configuration du plan de gestion

MÉMOIRE

soutenu le 21 Juin 2004

pour l'obtention du

DEA Informatique de Lorraine – Ecole Doctorale IAEM Lorraine

par

Chahinez Hamlaoui

Composition du jury

	Noëlle Carbonell	Professeur UHP-Nancy1
	Olivier Festor	Directeur de recherche INRIA
	Didier Galmiche	Professeur UHP-Nancy1
	Dominique Méry	Professeur UHP-Nancy1 et ESIAL
Encadrant :	Laurent Ciarletta	Maître de Conférences École des Mines-Nancy1

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

Mis en page avec la classe thloria.

Remerciements

C'est un grand plaisir pour moi autant qu'un devoir de remercier toutes les personnes, qui de près ou de loin, ont contribué à la réalisation de ce travail.

Je tiens tout d'abord à remercier vivement Laurent Ciarletta mon responsable de stage de DEA, pour son précieux encadrement, ses conseils et son soutien tout au long du stage.

Je remercie également Olivier Festor responsable du projet MADYNES, pour l'opportunité qu'il m'a offerte en m'accueillant au sein de son équipe, et dont j'ai eu à solliciter la compétence et l'expérience.

Mes vifs remerciements vont aussi à tous les membres de l'équipe MADYNES, pour leurs conseils sur le plan technique et intellectuel.

Mes remerciements s'adressent en particulier à B. Mouna, D. Guillaume et V. Vincent (blei82) pour l'aide aussi précieuse qu'efficace qu'ils m'ont régulièrement apportée, et pour la sympathique attention dont ils m'ont entouré.

Résumé

Actuellement, la configuration des infrastructures de gestion (plan de gestion) se fait manuellement. Or avec l'émergence des réseaux ambiants caractérisés par une forte dynamicité, et le grand nombre d'entités à gérer, une telle méthode de configuration va très vite devenir impraticable. De plus, elle est intrinsèquement inadaptée dans des réseaux mobiles tels que les réseaux ad hoc.

Pour la configuration et l'exploitation des services applicatifs, les protocoles de découverte de services ont fait leur apparition ces dernières années. Ils permettent de fournir des services, de les découvrir et de les utiliser de manière transparente. Ces architectures possèdent une grande autonomie et peuvent fonctionner aussi bien sur des imprimantes, des systèmes embarqués que sur de puissantes stations de travail.

Dans ce mémoire, nous étudions l'apport de ces protocoles de découverte de services pour la configuration automatique du plan de gestion. Nous proposons également, un modèle d'auto-configuration du plan de gestion, en se basant sur le protocole DNS-SD [1] (employé avec le mDNS [2]) et le protocole de gestion standard SNMP [3].

Mots clés : Découverte de services, Dynamicité, Gestion.

Abstract

Nowadays, the management infrastructure is configured manually. While manual configuration of management systems is a hard and time-consuming task, it is by nature impracticable in dynamic environment and mobile networks.

Recently, different services discovery protocols have been developped in order to discover, configure and use services (and applications) automatically and in a transparent manner from the user's point of view. These architectures are largely autonomous and can be deployed on systems ranging from printers to embedded operating systems as well as on powerful workstations.

In this document, after introducing the context of service discovery protocols, we present our model of automatic configuration of management systems. This model is based on DNS-SD, used with mDNS, and the SNMP standard protocols.

keywords : Service Discovery, Dynamicity, Management.

[1] Domain Name System - Service Discovery
[2] Multicast DNS
[3] Simple Network Management Protocol

Table des matières

Table des figures

Liste des tableaux

1

Introduction générale

1.1 La gestion

L'administration appelée aussi "gestion" ou "supervision" des réseaux et des services, est une composante cruciale de toute infrastructure de communication. Elle recouvre l'ensemble des fonctions nécessaires pour l'exploitation, l'entretien, et le suivi de l'état du réseau, dans le but d'assurer le fonctionnement optimal en dépit d'éventuels problèmes.

La gestion repose sur cinq domaines principaux :
- La gestion des fautes,
- La gestion de la configuration,
- La gestion de la comptabilité,
- La gestion de la performance,
- La gestion de la sécurité.

Le plan de gestion définit l'ensemble des entités participantes à l'activité de gestion et leur organisation. Ces entités doivent être configurées afin d'effectuer correctement les tâches qu'elles sont supposées fournir.

1.2 Problématique

Les réseaux mobiles et dynamiques connaissent une très forte expansion à l'heure actuelle. Les réseaux *ad-hoc*[4][1] et *Peer to Peer*[5](Pair à Pair) [2] en sont les exemples les plus marquants.

Ces réseaux sont caractérisés par l'absence d'infrastructure fixe, et par la mobilité fréquente de leurs éléments. Ces caractéristiques offrent une certaine flexibilité à leurs utilisateurs, mais augmentent la complexité du système de gestion. La configuration de ce système dans un réseau ad hoc par exemple est quasi-impossible.

Dans les réseaux d'entreprise, qui ont tout de même une infrastructure fixe, le nombre d'entités à administrer augmente, et varie selon la dynamicité du réseau. Ceci confronte l'administrateur à des problèmes majeurs qui touchent d'une manière importante la configuration du plan de gestion. Cette dernière va très vite devenir impraticable.

On peut résumer les problèmes liés à la configuration manuelle, et causés par la structure des réseaux en :

- **La dynamicité du système** : les entités dotées de ressources et exprimant des besoins hétérogènes, peuvent joindre et quitter le réseau à tout moment.

[4]Les réseaux ad hoc sont des réseaux sans fil déployés spontanément, qui permettent de connecter un ensemble de stations indépendamment de toute infrastructure.

[5]Les réseaux Peer to Peer sont des réseaux dynamiques dans lesquels une station peut jouer le rôle à la fois du client et du serveur.

- **L'absence de connaissance préalable sur le réseau** : une entité mobile qui rejoint le réseau pour la première fois, n'a aucune connaissance préalable des données qu'il lui sont nécessaires pour sa configuration, et donc sans l'aide d'un administrateur qui peut lui établir sa configuration manuellement, il lui est impossible d'être supervisé.

- **La croissance des réseaux** : et par conséquence l'augmentation du nombre d'entités à administrer et donc à configurer.

Avec toutes ces contraintes, il est important de songer à une configuration automatique des différentes entités de ce système.

La configuration automatique ou l'auto-configuration est le mécanisme qui permet à une entité (matérielle ou logicielle) d'obtenir les données nécessaires à son fonctionnement sans avoir recours à une intervention manuelle.

L'auto-configuration englobe : le processus de configuration et celui de reconfiguration. La configuration permet à l'entité d'interopérer dans un certain environnement, alors que la reconfiguration le permet lorsque certaines conditions de cet environnement changent.

Il existe cependant des protocoles de découverte de services, qui permettent la découverte dynamique des services, ainsi que de toutes les données nécessaires pour leur utilisation.

L'objectif de notre travail est d'exploiter ces protocoles dans le but de trouver une solution pour auto-configurer le plan de gestion, une solution qui puisse être appliquée dans des réseaux déployés spontanément, aussi bien que dans des réseaux fixes.

1.3 Organisation du rapport

Le chapitre 1 fournit l'état de l'art sur les protocoles de découverte de services, suivi d'une comparaison générale dans laquelle nous allons enrichir les études qui ont déjà été faites dans ce contexte [3, 4, 5, 6, 7, 8] que ce soit par rapport aux protocoles comparés ou bien par rapport aux critères de comparaison, et enfin une comparaison par rapport à l'adéquation des protocoles dans le cadre de l'auto-configuration du système de gestion.

Dans le chapitre 2, nous allons donner un aperçu sur la gestion SNMP, et plus particulièrement sur la configuration des entités participantes dans le système de gestion. Nous allons également présenter dans ce chapitre les travaux existants dans le domaine de l'auto-configuration du plan de gestion et leurs limites.

Le chapitre 3 sera consacré à la contribution résumée dans un modèle de couplage : protocoles de découverte de services/infrastructure de gestion et qui permet l'auto-configuration du plan de gestion.

Nous conclurons par un récapitulatif de ce travail, qui reprend les améliorations apportées, effectue des critiques et suggère des perspectives dans l'objectif d'un travail de thèse. Une annexe qui comporte un exemple du prototype accompagne également ce rapport.

2

Les protocoles de découverte de services

2.1 Introduction

Un protocole de découverte de services est un protocole qui permet de déployer, découvrir et utiliser les services de façon dynamique et spontanée. Par *"service"*, nous désignons de façon générique, toute application possédant une interface clairement définie, capable d'effectuer des traitements ou des actions au profit des utilisateurs appelés *clients*.

Pour bien illustrer le rôle des protocoles de découverte de services, on peut donner l'exemple courant que l'on rencontre dans toute spécification de ces protocoles : supposons que l'on a un client qui rejoint un réseau pour la première fois et qui souhaite imprimer un document. Il doit localiser les imprimantes du réseau, les comparer en fonction de leurs caractéristiques, et en choisir une par la suite. Les protocoles de découverte de services permettent de réaliser tout cela automatiquement et sans intervention d'un administrateur.

Dans la section suivante, nous allons présenter les principaux protocoles de découverte de services, en s'appuyant principalement sur les fonctionnalités suivantes :

2.1.1 Description de services :

La description des services est une composante importante du protocole, elle doit être expressive pour permettre au client de choisir le service correspondant à ses besoins et capacités, et conforme à un standard spécifique, afin de pouvoir traiter les requêtes et assurer l'interopérabilité entre les différents fournisseurs de services[6].

Cette description contient les caractéristiques essentielles du service telles que son type, son adresse, ses attributs et parfois même son interface d'accès sous forme d'API [7] ou d'interface graphique.

2.1.2 Approches de découverte de services :

On distingue deux grandes approches de découverte de services :

1. **L'approche centralisée** :
   Cette approche se base sur une ou plusieurs entités serveurs (jouant le rôle de répertoires), sollicitées par les services pour enregistrer leurs descriptions et par les clients pour la recherche d'un service donné. La découverte de ces entités se fait par l'une des méthodes suivantes (cf. figure 2.1) :
   - **Passive :** Cette découverte est basée généralement sur la combinaison entre le multicast et la périodicité appelée en littérature "announce/listen". Le serveur annonce périodiquement

[6] Dans la suite du document on utilisera indifféremment les termes service et fournisseur de services
[7] API Application Programming Interface

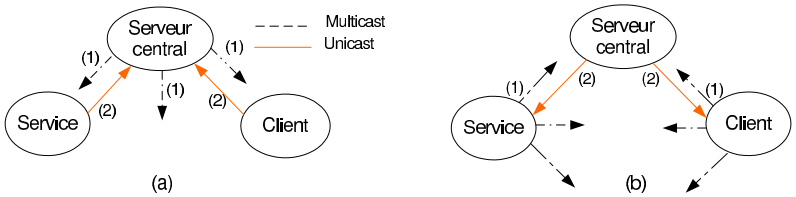

FIG. 2.1 – Approche centralisée : (a) Découverte passive- (b) Découverte active

son identité sur l'adresse multicast qui est une adresse de groupe attribuée au protocole, à laquelle les clients et les services intéressés doivent s'enregistrer (point vers multipoint). Les services et les clients le contactent ensuite en unicast.

- **Active :** Les clients et les services diffusent les requêtes destinées à localiser le serveur, lequel leur répond en unicast.
- **Statique :** Les références du serveur sont connues par configuration manuelle ou en utilisant les options du DHCP [8] [9].

2. **L'approche décentralisée** :
   Dans cette approche, il n'y a pas de serveur pour localiser les services. Pour pouvoir gérer la découverte de services dans ce cas, on distingue deux modèles :(cf. figure 2.2)

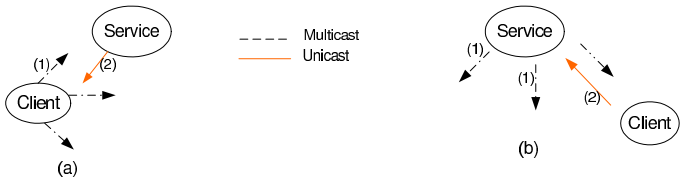

FIG. 2.2 – Découverte décentralisée : (a)le modèle Push - (b)le modèle Pull

- **Le modèle pull** : permet aux clients d'envoyer leurs requêtes à travers le réseau sur l'adresse multicast attribuée au protocole, les services conformes à la requêtes leur répondent en unicast (dans ce modèle, les clients sont proactifs et les services sont réactifs).

- **Le modèle push** : utilisé par les services pour s'annoncer périodiquement en diffusion. Les clients peuvent ensuite les contacter en unicast (dans ce modèle, les services sont proactifs et les clients sont réactifs).

Le choix entre ces deux approches est critique. L'approche centralisée peut être déployée à grande échelle et augmente les performances en termes de consommation de bande passante, mais son fonctionnement dépend de celui du serveur central. L'approche décentralisée ne peut être déployée qu'à petite échelle, mais elle est bien adaptée dans les réseaux sans infrastructure fixe tels que les réseaux ad hoc, où l'on n'a pas toujours d'entité qui peut être présente de façon permanente pour assurer le rôle du serveur central.

2.1.3 Gestion de la dynamicité :

L'une des préoccupations majeures des protocoles de découverte de service est la gestion de la dynamicité. En effet, avec l'émergence des réseaux actuels : peer to peer, ad hoc etc, caractérisés par une forte dynamicité, les clients et les services peuvent joindre ou quitter le réseau fréquemment

[8]Dynamic Host Control Protocol

ou même subire d'éventuels changements (dans leur état ou leur configuration). Pour gérer cela, deux mécanismes sont employés :

1. **Le bail** : qui indique la durée de disponibilité du service. Cette durée est soit définie par le service soit négociée par les deux entités (service/ serveur), elle est renouvelable périodiquement et à son expiration le service est considéré comme non valide .
2. **La notification** : utilisée pour informer les clients souscris de certains changements pouvant les intéresser, parmi lesquels on distingue : l'arrivée et le départ d'un service ou bien le changement des variables d'état d'un service donné. La durée de la souscription des clients doit être renouvelable périodiquement.

2.2 Aperçu sur les protocoles de découverte de services

2.2.1 Jini

Jini [10] est une technologie développée par Sun Microsystems, elle s'appuie sur un environnement Java distribué et utilise le terme de "fédération" pour désigner la coordination entre les différentes entités participantes. En plus de la déclaration et la découverte de service, Jini définit aussi la façon de communiquer avec le service . Son infrastructure se base sur :

1. **Les lookups services (LUS) :** représentent les serveurs centraux de Jini dans lesquels le service enregistre son proxy ; Ce dernier est un objet Java qui permet l'utilisation du service. Le LUS est aussi chargé de répondre aux requêtes des clients, et de les notifier lors d'un éventuel changement dans la fédération Jini, en utilisant dans ce dernier cas le mécanisme *"Distributed Event"*. [11] Ce mécanisme permet à une machine virtuelle Java de d'enregistrer son intérêt concernant l'occurence d'un évènement dans une autre machine virtuelle et de recevoir des notifications quand un tel évènement se produit.

2. **Le processus de découverte :** permet de localiser les lookups services d'une manière : passive (*Multicast Annoucement Protocol*), active (*Multicast Request Protocol*) ou statique, l'interaction se fait ensuite par l'intermédiaire d'un Stub RMI (proxy ou souche) fourni par le LUS, ce proxy implémente l'interface *ServiceRegistrarInterface* qui définit les différentes méthodes que les clients et les services peuvent invoquer pour communiquer avec le LUS.

3. **Le processus d'adhésion :** une fois que le service a découvert le LUS, il utilise la méthode *register* du Stub RMI pour enregistrer sa description sous forme d'interface qui encapsule : son identificateur, son objet proxy et ses différents attributs. Le service indique sa disponibilité à travers un bail renouvelable qui étend la notion du *garbage collector* aux environnements distribués.

4. **La recherche des services :** pour utiliser un service donné, le client Jini invoque la méthode *lookup* de l'interface *ServiceRegistrarInterface* en précisant l'identificateur du service ou bien son type et d'éventuels attributs. Le LUS lui renvoie une copie du proxy du service qui lui permet soit de l'exécuter localement soit de communiquer avec lui à distance. La figure ci-dessous (cf. figure 2.3) décrit les principales étapes dans le protocole Jini dans le cas d'une découverte active :

Jini utilise la brique logicielle RMI [9] [12], qui permet aux entités java distantes de communiquer et de télécharger de façon dynamique le code représentant le service.

Jini est déployée dans divers équipements (serveurs, appareils mobiles, imprimantes...etc.), et continue son développement dans le monde de la domotique.

[9] Java Remote Methode Invocation

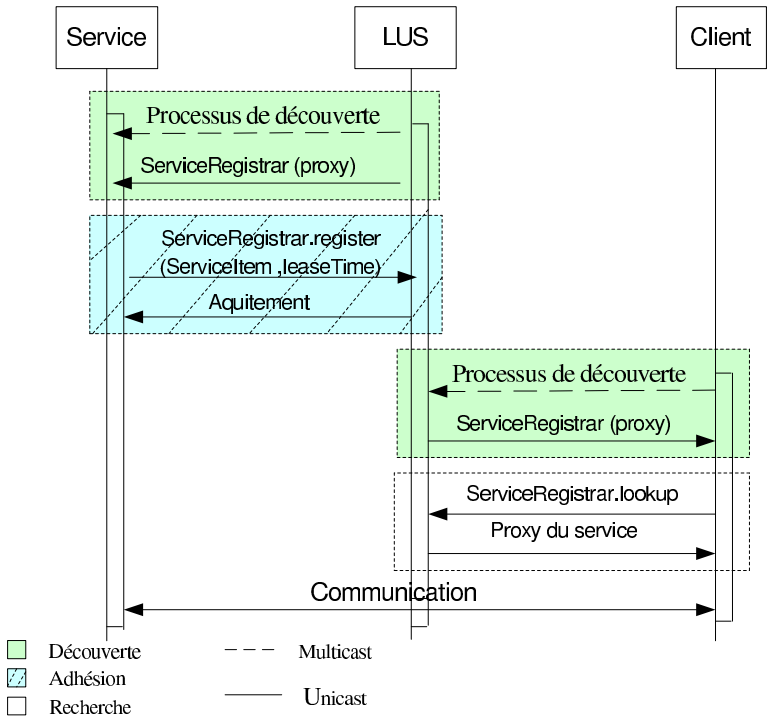

FIG. 2.3 – Les échanges Jini

2.2.2 SLP (Service Location Protocol)

SLP est un standard proposé par l'IETF [10] pour les réseaux IP [13]. Son architecture est similaire à celle de Jini, mais contrairement à cette dernière il adopte les deux approches : centralisée et décentralisée, et sa consommation de bande passante est réduite de 50%[14].

SLP inclut dans son architecture trois types d'agents : l'agent de service SA [11] qui est le processus qui s'exécute au profit du service, l'agent utilisateur UA [12] qui s'exécute au nom du client, et l'agent de répertoire DA [13] qui est le recueil des différentes descriptions de services. Pour la découverte du DA on retrouve les trois méthodes : statique, active et passive. Dans ces deux dernières, si le multicast n'est pas supporté on peut utiliser la diffusion en broadcast.

La communication dans ce protocole s'appuie principalement sur un échange de messages dont les plus importants sont mentionnés dans la figure ci-dessous (cf.figure 2.4) qui schématise une découverte active : le DA annonce son identité en utilisant des messages DaAdvert, les services peuvent ensuite le contacter pour enregistrer leurs descriptions en utilisant des messages SrvReg, et pour lesquels le DA répond par un message d'acquitement SrvAck.

Les requêtes sont structurées dans des messages SrvRqst, qui précisent le type du service et les différents attributs sous forme de prédicats, spécifiés grâce aux opérateurs : & & , !=,= =, =<, > etc . En mode décentralisé, la découverte de services se fait par le modèle Pull.

[10] Internet Engineering Task Force
[11] Service Agent
[12] User Agent
[13] Directory Agent

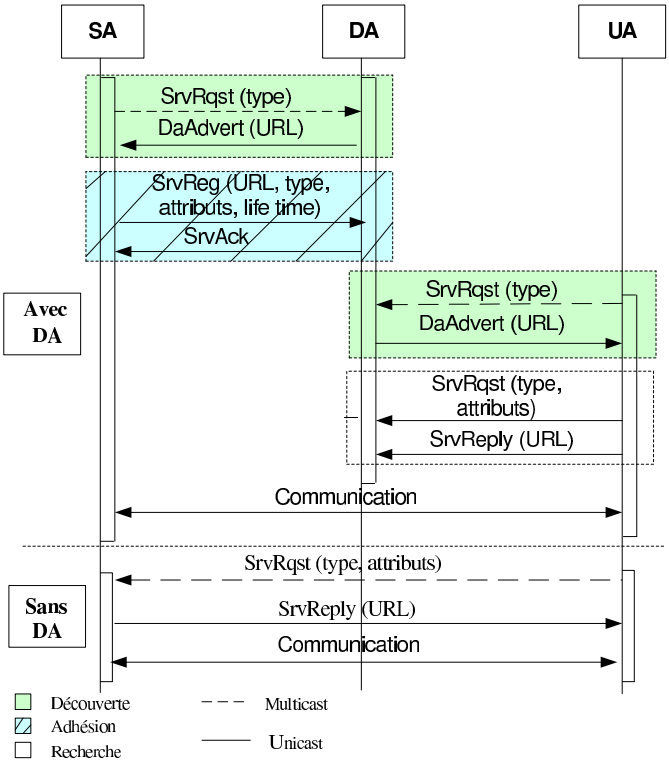

FIG. 2.4 – Les échanges SLP

Comme indiqué dans l'exemple ci-dessous, la description du service dans SLP comporte : *un service URL* qui définit le type du service, le protocole de communication, le numéro de port et le chemin, et *un service template* qui contient les différents attributs du service.

Exemple :
URL = service :printer :lpr :hostname :port/path
Attributs=(printer-name =lj 4050),(printer-model= HP LJ4050 N),
(printer-location= Room 0409),(color-supported=false)

SLP utilise la notion de «scope» pour limiter l'étendue d'envoi de requête à des domaines administratifs ou géographiques particuliers et pour pouvoir établir sa politique de sécurité. Il existe aujourd'hui sous la version 2, et il est implémenté dans plusieurs produits commerciaux tels que Hewlett Packard (imprimantes, PDAs [14] ...etc) et le système d'exploitation Solaris 8.

2.2.3 UPnP (Universal Plug and Play)

UPnP est le protocole de Microsoft, conçu principalement pour être déployé dans les réseaux domotiques et domestiques [15]. Il adopte une approche décentralisée, et utilise les termes : *" device"* et *"point de contrôle"* pour désigner respectivement : l'entité qui offre les services et celle qui les utilise [16].

UPnP repose sur toute une pile de protocoles standards, qui lui permettent d'accomplir les fonctionnalités suivantes :

[14] Personal Digital Assistants

- **L'auto-configuration IP (Auto IP)** : Permet à une machine d'obtenir une adresse IP en l'absence du serveur DHCP, en générant tout simplement une adresse aléatoire non routable sur Internet et en envoyant par la suite une requête ARP [15] pour voir si une autre machine n'a pas déjà choisi cette adresse.

- **La description du device et des services** : Cette description est structurée dans un document XML contenant le type du device, le nom du constructeur, la liste des *devices* embarqués ainsi que la liste des différents services et leurs descriptions.

- **L'annonce et la découverte de services** : Elles se font en multicast à travers le protocole SSDP [16] qui utilise deux variantes du protocole HTTP [17] basées sur UDP [18] (HTTPMU : pour les requêtes et les annonces, et HTTPU : pour les réponses). L'annonce du device contient sa durée de disponibilité et l'URL où les points de contrôles peuvent récupérer sa description.

- **Le contrôle et l'invocation d'actions** : Le point de contrôle peut exécuter des actions sur le device en utilisant l'URL de contrôle contenue dans sa description et le mécanisme RPC [19] à travers le protocole SOAP [20] [17].

- **La gestion des événements** : La description d'un service inclut une liste de variables qui définissent l'état de celui-ci ; le point de contrôle peut souscrire afin de recevoir la mise à jour de ces variables en cas de changement. Cette souscription est effectuée à l'URL du serveur d'événements du service qui se trouve dans le document de description en utilisant l'architecture de notification GENA [21].

- **La présentation** : Dans le document de description du device, le constructeur peut rajouter l'URL d'une page HTML qui représente l'interface du device.

UPnP a été déployé dans plusieurs produits, entre autres : Windows XP, UPnP Internet Gateway et l'imprimante UPnP.

2.2.4 Salutation

Salutation est un protocole, proposé par un consortium d'entreprises, pour la découverte de services dans les petits réseaux indépendamment du protocole de transport sous-jacent. Son architecture se base sur des entités appelées SLMs [22], déployées par les clients et les services pour la découverte et l'enregistrement de services, ainsi que pour maintenir des sessions entre un client et un service donné [18].

Les entités n'ayant pas de SLM peuvent utiliser des SLMs distants à travers un mécanisme RPC. Chaque SLM est localisé par un identificateur universel unique et communique avec les autres par le protocole SunRPC en utilisant le Pull (cf.figure 2.5).

Pour assurer l'interopérabilité entre les entités réseaux ayant des environnements d'exécution différents, Salutation définit deux interfaces de programmation :

- SLM-API : Utilisée par les clients et les services pour la communication avec leur SLM.

[15] Address Resolution Protocol
[16] Simple Service Discovery Protocol
[17] Hyper Text Transfer Protocol
[18] User Datagram Protocol
[19] Remote Procedure Call
[20] Simple Object Access Protocol
[21] Generic Event Notification Architecture
[22] Salutation Managers

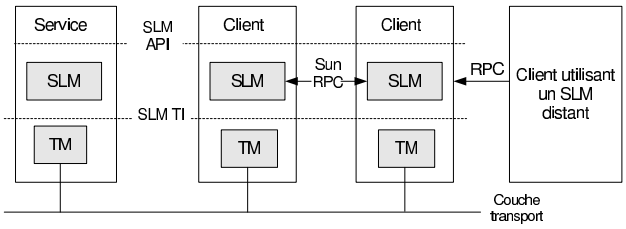

FIG. 2.5 – Salutation

– SLM-TI : Utilisée par le SLM pour communiquer avec les modules TM [23] dépendants de la couche de transport.

Les services dans Salutation sont décrits à travers des enregistrements *Functional Unit Description Record* composés de : l'identificateur du service (ex :[Scan], [print]...etc) et d'une liste *d'attributs Record* contenant pour chaque attribut : son identificateur (printColor), l'identificateur de la fonction de comparaison (strEqualTo, intEqualOrGreaterThan...etc) et finalement sa valeur.

L'invocation de services dans Salutation appelée *service session management*, se fait soit indépendamment du SLM c'est-à-dire par un protocole défini dans la description du service (native mode) soit en étant prise en charge par le SLM qui définit un format spécial des données transportées (Salutation mode)[19].

2.2.5 SSDS (Secure Service Discovery Service)

SSDS est un protocole de découverte élaboré dans le cadre du projet Ninja de l'université de Berkeley. Son architecture basée sur la téchnologie *Java* vise le déploiement et la découverte de services à grande échelle d'une manière sécurisée. En effet, en plus des clients, services et serveurs, SSDS inclut aussi une autorité de certification assurant l'intégrité, l'authentification et la confidentialité des communications, ainsi qu'un gestionnaire de capacité responsable des droits d'accès des clients [20].

Les serveurs sont structurés hiérarchiquement en plusieurs domaines, et chaque serveur annonce sur l'adresse multicast globale choisie pour ce protocole : son identité, les différentes adresses des domaines qu'il couvre ainsi que d'autres informations concernant l'autorité de certification, le gestionnaire de capacité et les fréquences des annonces des services (pour gérer les annonces multicast par rapport à la bande passante disponible). Ces données sont signées et estampillées pour garantir l'intégrité et éviter le rejeu.

Les annonces des services destinées principalement aux serveur sont envoyées périodiquement en multicast en utilisant *«Secure One Way Broadcast Protocol»* une approche hybride de cryptage qui assure la confidentialité et l'authentification, et sont structurées sous forme d'un document XML contenant l'adresse Java RMI, le temps d'expiration du service et la liste des clients autorisés à utiliser ce service. Cette liste est aussi transmise au gestionnaire de capacité qui génère les certificats d'accès des différents clients aux services.

Les requêtes structurées aussi en XML sont envoyées aux serveurs avec les droits d'accès de chaque client et les différents attributs demandés.

L'accès aux serveurs et aux services est fait selon *Authenticated-RMI* qui est une autre application du groupe Ninja, et qui permet d'invoquer des méthodes sur des objets distants d'une façon

[23] Transport Manager

authentifiée et cryptée.

SSDS est un protocole basé sur Java. Un prototype de son implémentation a été testé sur Linux mais il est possible de le déployer dans différentes plates-formes.

2.2.6 DNS-SD (DNS Service Discovery)

DNS-SD est un protocole issu des travaux du groupe zero conf [21] de l'IETF qui définit une architecture de découverte de services basée sur l'infrastructure de DNS[24] [22].

Les services dans DNS-SD sont décrits en utilisant le format standard des *records*[25] PTR (pour Pointer), SRV (pour service) et TXT (pour texte) du DNS. Le PTR utilisé dans le DNS pour la translation des adresses IP en nom de machines permet dans DNS-SD d'énumérer les instances d'un type de service donné [26], le SRV permet de localiser la machine qui offre le service, et le record TXT stocke les paires attribut/valeur donnant des informations additionnelles sur le service.

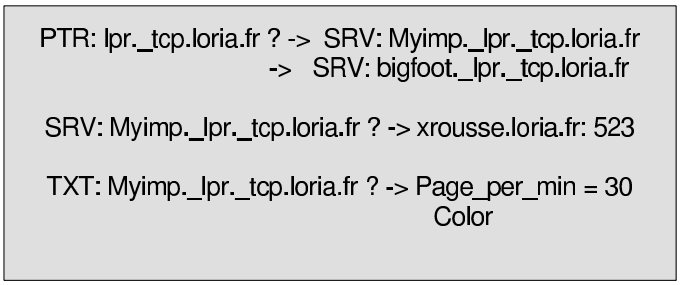

FIG. 2.6 – Découverte des services DNS-SD

La figure ci-dessous (cf.figure 2.6) illustre la découverte d'un service d'impression. Le client envoie une requête PTR dans laquelle il précise le type de service (lpr), le protocole de transport (tcp) et le domaine (loria.fr), et reçoit en réponse la liste de toutes les instances d'imprimantes disponibles (Myimp et bigfoot). Une requête SRV pour une instance donnée (Myimp) lui permet ensuite de connaitre l'URL de la machine qui offre le service et son numéro de port (xrousse et le port 523). Enfin pour choisir entre les services de même type, il lui suffit de demander les attributs contenus dans le record TXT.

Pour s'enregistrer au niveau du serveur DNS qu'on suppose connaître d'une façon statique (Option DHCP), les services utilisent le protocole Dynamic Updates [23], et le bail RRTTL [27] pour indiquer leurs temps de mise en cache.

Dans les petits réseaux, DNS-SD peut se passer du serveur DNS en utilisant le protocole mDNS[28] [24] pour diffuser les annonces et les requêtes en multicast.

Le multicast DNS repose sur le principe de caches distribués - hébergés sur chaque machine - et s'appuie sur la coopération des hôtes pour répondre aux différentes requêtes. On retrouve en plus des requêtes standards de DNS, deux autres plus sophistiquées :

[24]Domain Name System : DNS est l'infrastructure de nommage qui permet de faire correspondre les adresses IP et les nom de machines (URL)
[25]Enregistrements
[26]Chaque instance est identifiée par un nom attribué par le fournisseur du service
[27]Resource Record TTL
[28]Multicast DNS

- *One short queries, Accumulating multiple responses* : Consiste en une retransmission de la requête jusqu'à recevoir un nombre suffisant de réponses.
- *Continuous querying* : Représente une sorte de navigation qui permet au client d'être toujours au courant des différents services disponibles, auxquels il s'intéresse.

Les réponses peuvent être envoyées en multicast (pour mettre à jour les différents caches) ou en unicast "à la demande", si le client le souhaite.

Pour gérer le nombre de messages multicast transitant à travers le réseau, mDNS utilise plusieurs mécanismes, parmi lesquels : *Known Answer Suppression* qui indique aux entités de ne pas répondre aux requêtes répétées, et *Duplicate Answer Suppression* pour ne pas transmettre une même réponse par deux entités différentes. Il est aussi possible d'avoir dans une même réponse les valeurs des records SRV et TXT ce qui permet de limiter le nombre de requêtes.

L'implémentation de ce protocole a été réalisée initialement sur des Macs OS x, et on peut l'intégrer dans toutes les plates-formes en utilisant les APIs spécifiques (par exemple les APIs Java) ce qui garantit une hétérogénité complète.

2.2.7 UDDI (Universal Description, Discovery and Integration)

UDDI est une réalisation conjointe des industriels IBM, Microsoft et Ariba qui définit un standard permettant la description, l'enregistrement et l'utilisation de services Web à travers l'Internet [25].

UDDI utilise des annuaires distribués UBR [29] qui hébergent des informations sur les intervenants du e-business (sociétés, administrations etc.) ainsi que les descriptions des différents services Web proposés.

La publication dans un annuaire UDDI doit être structurée en quatre types de données dans le but de fournir une meilleure compréhension et une localisation plus rapide des services [26], cette structure se compose de :

- **BusinessEntity** : chaque service appartient à une organisation identifiée par : un nom, une description, des numéros de téléphone et des adresses de contacts...etc, cette entité regroupe toutes ces données.
- **BusinessService** : inclut les informations non techniques relatives au service, telles que son nom et une brève description.
- **BindingTemplate** : définit les points d'entrée au service tels que : URL, mailto, fax, téléphone etc.
- **tModel** : comprend des liens vers les informations techniques du service, tels que la référence à la partie abstraite du fichier WSDL [30] qui décrit les différentes méthodes disponibles. [27]

Ces informations structurées dans un document XML seront par la suite classées au niveau des annuaires en trois catégories :

- **Les White Pages** : Correspondent à BusinessEntity.
- **Les Yellow Pages** : Contiennent un répertoire des services et des produits, et un index géographique. Il subdivisent les entreprises selon une taxonomie standard. (correspondent à BusinessService).
- **Les Green Pages** : Contiennent les règles du e-commerce, les descriptions des services ...etc. (correspondent à BindingTemplate et tModel)

[29] UDDI Business Registry
[30] Web Services Definition Language

Pour assurer l'interopérabilité dans ces systèmes complètement hétérogènes, UDDI se base sur le langage XML et le protocole SOAP pour toute communication entre les différentes entités participantes.

La recherche et la publication des services se font par programmation en utilisant les APIs SOAP (API Inquiry, API Publishing), ou par simple accès aux différents UBRs de Microsoft, IBM ou autres.

2.2.8 WS-Discovery (Web Service Discovery)

Cette nouvelle architecture qui s'adosse comme son nom l'indique à la technologie des Web Services a été élaborée par Microsoft et Intel, en collaboration avec BEA System, IBM, et Canon. Cette approche vise à étendre l'architecture des services Web, et de la porter à l'univers des services matériels de tout type. [28]

WS-Discovery est un protocole décentralisé conçu pour les réseaux pair à pair et ad hoc et basé sur la découverte en multicast des services dans un réseau local. Son principe est très proche de UPnP vu qu'on retrouve les deux modèles :

- Push : reflété par les messages *Hello* et *Bye* envoyés par les services pour la gestion de leur disponibilité.
- Pull : reflétant les requêtes des clients.

Toutes les données générées par ce protocole (description des services, requêtes, annonces) sont structurées en XML et sont envoyées en utilisant le protocole SOAP au dessus de UDP.

Pour limiter le trafic multicast généré par les requêtes des clients, et optimiser le temps de latence lors de la découverte dans les réseaux ad hoc, on peut rajouter une entité intermédiaire jouant le rôle de proxy (DP [31]), dont la découverte se fait de manière implicite. En effet, en réponse à une requête en multicast cherchant un service donné le DP répond. Cela permet ainsi au client de basculer en mode «DP specific protocol» c'est à dire de communiquer directement avec le DP au lieu d'utiliser le multicast.

Au premier regard, ce protocole semble assez proche du standard UDDI. Cependant, à la différence d'UDDI qui englobe de nombreuses informations tierces, WS-Discovery se limite à la simple description du service, et de plus il n'est pas conçu pour être déployé à l'échelle mondiale comme UDDI. En outre, les deux protocoles peuvent être utilisés en complément l'un de l'autre. En effet dans un réseau local, un serveur UDDI utilisant WS-Discovery peut se signaler sur un réseau IP.

2.3 Analyse comparative des approches

2.3.1 Description des services

Comme nous venons de le voir, la description de service doit informer le client de tout ce qui est nécessaire pour pouvoir l'utiliser. UPnP, UDDI, WS-Discovery et SSDS ont choisi une technologie existante XML qui est très expressive et extensible. UPnP utilise des schémas de description standardisés par UPnP Forum alors que UDDI utilise la technologie WSDL.

La description dans SLP est faite selon un service template et un service URL standardisé par l'IANA. DNS-SD utilise les records SRV et TXT implémentés par DNS alors que Salutation utilise des formats propres où tous les services et attributs sont définis dans sa spécification. Ceci entraîne une certaine restriction pour les développeurs de services. Jini décrit ses services à travers une interface Java.

[31] Discovery Proxy

2.3.2 Recherche de serveurs et déclaration de services

Jini, SLP et SSDS utilisent les mêmes approches permettant aussi bien à un client qu'à un fournisseur de service de localiser un serveur. La déclaration de services se fait ensuite par enregistrement au niveau du serveur.

Dans SLP la découverte statique est assurée par l'option 78 du DHCP, alors que pour la découverte active, il utilise un multicast convergent de SrvRqst qui consiste en une retransmission de la requête destinée à chercher les serveurs, quand aucune réponse n'est reçue avec une incrémentation exponentielle de l'intervalle de temps entre de deux transmissions consécutives. SSDS utilise la découverte passive ou active. Dans ce dernier cas et afin de réduire la latence, les clients utilisent la recherche par anneaux croissants, qui est similaire au multicast convergent de SLP avec une incrémentation du TTL [32] de la requête. Cependant, et bien que les services s'enregistrent au niveau des serveurs, ils continuent d'émettre des annonces périodiques pour des raisons de robustesse.

Dans UDDI et DNS-SD les serveurs centraux sont supposés être connus d'une manière statique. Dans DNS-SD, la référence du serveur DNS est connue par les options du DHCP, alors que dans UDDI on retrouve des URLs [33] spécifiques aux opérateurs SAP, IBM et Microsoft.

UPnP et WS-Discovery, basés sur une approche décentralisée, utilisent le modèle Push pour la publication des services. WS-Discovery permet cependant l'utilisation de proxy (DP), dont la découverte se fait d'une façon implicite, vu qu'il répond aux requêtes multicast des clients, leur indiquant qu'ils peuvent basculer sur un mode centralisé.

Dans Salutation qui utilise aussi une approche décentralisée, les services sont soit enregistrés sur l'entité qui les offre, ou bien sur une entité distante (si le fournisseur de service ne possède pas de SLM) à l'aide d'un SLM distant dont la découverte se fait :(i)par diffusion de requête, (ii) par configuration statique, (iii) ou bien par envoi de requêtes en unicast vers un serveur central contenant le répertoire des SLMs du réseau.

2.3.3 Les requêtes

1. **L'envoi de requêtes** :
   Dans UDDI, Jini, SLP, DNS-SD et SSDS, le client envoie sa requête au serveur qu'il a découvert. Dans le cas où il n'aurait pas trouvé de serveur, le traitement diffère d'un protocole à un autre :

   Un client Jini peut utiliser la technique dite *peer lookup*, dans laquelle le client se substitue au serveur et demande aux fournisseurs de services de venir s'enregistrer auprès de lui. Il sélectionne ensuite les services dont il a besoin et refuse les autres. Dans DNS-SD et SLP le Pulling est employé, alors que dans SSDS, les annonces des services peuvent être écoutées par les clients qui seront donc informés des services disponibles dans leur domaine.

   Dans Salutation les requêtes sont envoyées vers un SLM spécifié en unicast, contrairement à UPnP et WS-Discovery, où on utilise le Pull.
2. **L'expressivité des requêtes** :
   En formulant une requête, il se peut qu'il existe plus d'un service qui réponde aux critères de recherche du client, le nombre de choix que le client aura donc à faire est proportionnel à l'expressivité de sa requête.

   Jini, SLP, Salutation et SSDS permettent de mentionner les attributs dans les requêtes, Jini se base sur l'égalité des attributs, SLP et Salutation utilisent des opérateurs prédéfinis qui

[32] Time To Live
[33] http ://uddi.sap.com, http ://uddi.ibm.com et http ://uddi.microsoft.com

aident à formuler des requêtes très expressives, SSDS quant à lui s'appuie sur un matching entre la description du service et la requête en utilisant Xset [29] ; L'évaluation des requêtes dans ces protocoles se fait donc au niveau du serveur central ou bien au niveau du service dans le cas décentralisé, cependant et vu que dans Jini et SSDS les requêtes ne sont pas aussi expressives que dans SLP et Salutation, un deuxième filtrage selon les attributs peut se faire ensuite du côté client.

Dans DNS-SD, UDDI, WS-Discovery et UPnP il est impossible de formuler des requêtes selon les attributs, leur évaluation se fait donc du côté client afin qu'il puisse choisir le service adéquat parmi les alternatives.

2.3.4 Gestion de la dynamicité

1. **Le bail** :
   La notion de bail (leasing) est très importante dans les environnements dynamiques, puisqu'elle permet de gérer la disponibilité des services. Cette fonctionnalité est implémentée dans tous les protocoles que nous avons étudiés à l'exception d' UDDI, WS-Discovery et Salutation. Cependant la disponibilité du service est gérée par des messages "Bye" dans WS-Discovery, ce qui n'est pas aussi fiable que le leasing, alors que dans Salutation, le client peut demander à son SLM de contrôler la disponibilité d'un service donné à travers des messages envoyés périodiquement ce qui peut engendrer un trafic important.

2. **La notification** : La notification est aussi importante que le bail. Le protocole UPnP notifie tous les clients en utilisant des messages SSDP lorsqu'un service rejoint ou quitte le réseau. De plus, les clients peuvent s'inscrire au niveau du service et recevoir des notifications lorsque celui-ci subit des changements. De même, les clients de Salutation peuvent s'inscrire pour être notifiés des changements des variables dynamiques d'un service donné.

   Dans Jini, une configuration optionnelle est mise en œuvre pour tenir compte des évènements qui peuvent se produire, en s'inscrivant soit au niveau du LUS soit au niveau du service même.

   UDDI a proposé récemment une versions bêta de son implémentation, dans laquelle elle spécifie une API qui permet de souscrire à des services et d'être alerté en cas de modifications apportées à l'annuaire. La notification dans WS-Discovery est faite d'une façon implicite à travers des messages "Hello" envoyés en multicast à tout le monde, donc les clients n'ont pas besoin de souscrire.

   Dans DNS-SD et SSDS, l'utilisation du modèle push implique que les clients peuvent être seulement notifiés de l'arrivé ou du départ des services, alors que dans SLP aucun mécanisme n'a été spécifié.

2.3.5 La sécurité

La sécurité n'a été totalement couverte que dans SSDS. En effet ce dernier gère le contrôle d'accès des clients aux services et utilise aussi bien l'authentification que le cryptage pour sécuriser toutes les communications.

Dans SLP, l'intégrité des descriptions des services et de l'annonce du DA est garantie au travers de signatures digitales. Dans Jini et DNS-SD la sécurité est restreinte à celle offerte respectivement par Java et par l'infrastructure du DNS (DNSsec) [30], alors que dans Salutation il est possible de s'authentifier en utilisant l'identificateur de l'utilisateur et son mot de passe.

Dans la dernière version de UDDI on fait référence aux signatures numériques préservant l'intégrité des descriptions des services. La sécurité dans le protocole WS-Discovery n'a pas été traitée

(c'est un protocole récent) mais ses auteurs recommandent l'utilisation du modèle décrit dans WSS-SOAP Security pour établir un mécanisme de sécurité [31].

2.3.6 Routage de requêtes et déploiement à grande échelle

Une requête qui ne trouve pas de réponse dans le sous-réseau où elle a été envoyée est généralement routée vers d'autres serveurs afin de trouver d'éventuelles réponses. Le mécanisme de propagation d'informations entre les serveurs a été clairement défini dans SSDS en déployant toute une hiérarchie de serveurs où chacun d'eux est au courant des services disponibles dans son sous-arbre de hiérarchisation, à l'aide d'un mécanisme de hachage.

Dans Jini on a mentionné l'enregistrement des proxys des LUS dans d'autres LUS, mais l'organisation de ces derniers n'est pas spécifiée. Dans DNS-SD, et comme on se base sur l'infrastructure DNS, il est possible d'utiliser la même infrastructure hiérarchique des serveurs que le DNS. Dans SLP, on fait référence à WASRV pour étendre les portées de ce protocole. Cette approche définit dans chaque domaine administratif un agent AA [34] qui annonce certains services (définis par l'administrateur) disponibles dans son domaine et un autre agent BA [35] qui écoute d'une façon sélective aussi les annonces des services des autres domaines et les annonce dans son domaine [32]. Dans UDDI, on utilise le "Nuage UDDI"c'est à dire qu'il suffit de s'inscrire dans l'un des UBRs d'un opérateur et cette inscription sera ensuite répercutée dans les autres UBRs.

Les tableaux (6.1, 6.2) dans l'annexe résument les caractéristiques importantes de chaque protocole.

2.4 Comparaison dans le contexte de l'auto-configuration du plan de gestion

Bien qu'ils aient un but commun et offrent des fonctionnalités similaires, ces protocoles de découverte de services se basent sur des caractéristiques différentes qui peuvent résoudre des problèmes différents dans chacun de leur contexte cible.

Dans la section suivante, nous allons situer les besoins qui semblent importants pour l'auto-configuration du plan de gestion, et préciser l'adéquation de ces protocoles dans le cadre de l'auto-configuration.

2.4.1 Déploiement dans les réseaux avec ou sans infrastructure fixe

On remarque que certains protocoles quoique centralisés peuvent être implémentés dans les réseaux sans infrastructure fixe, du moment qu'ils utilisent des mécanismes dynamiques pour localiser le serveur central.

Cependant, dans cette approche, le problème se pose lorsque cette entité centrale quitte le réseau. En effet, on se voit dans ce cas confronté à de nouvelles contraintes qu'on peut résumer dans cette série de questions : comment élire un nouveau serveur central ?, sur quels critères faut-il se baser pour élire ce serveur ?,et enfin qui va élire ce serveur ?

Les protocoles qui adoptent une approche décentralisée sont meilleurs pour les réseaux sans infrastructure fixe, mais ils ne sont adaptés qu'aux petits réseaux. En effet, les annonces et requêtes envoyées en multicast génèrent une importante charge du réseau.

[34] Agent Advertising
[35] Brokering Agent

Le protocole de découverte de service idéal pour l'auto-configuration du plan de gestion, doit être déployable dans les architectures avec ou sans infrastructure fixe et aussi dans les petits et grands réseaux. Par conséquent il doit implémenter les deux approches : centralisée et décentralisée (ce choix se justifie par le fait que chaque approche est conçue pour un type de réseau spécifique).

Les protocoles qui utilisent une découverte manuelle du serveur tel que UDDI, et ceux qui n'adoptent que l'approche décentralisée (Salutation, UPnP et WSDiscovery) ne sont pas appropriés pour répondre à nos besoins.

2.4.2 Mobilité fréquente et gestion de la dynamicité

Avec la mobilité des différentes entités du réseau, qui peuvent apparaitre et disparaitre d'une façon fréquente, il est impératif que le protocole utilisé emploie le mécanisme du bail. En effet, cette notion est importante puisque l'auto-configuration englobe : la configuration et la reconfiguration. Cette dernière se fait dans le cas où le service n'est plus valide, ce qui est bien sûr détecté à travers le bail (leasing).

Le calibrage du leasing nécessite de trouver le bon compromis entre la disponibilité du service et le nombre des messages de mise à jour. Pour cela son calibrage doit être bien défini par rapport à la dynamicité du réseau.

2.4.3 Ressources des périphériques

Les réseaux sont généralement hétérogènes et contiennent de plus en plus d'équipements mobiles tels que les PDAs, les laptops et les téléphones portables...etc. De tels équipements sont caractérisés par des capacités de stockage et d'énergie assez faible, on ne peut donc pas s'attendre à ce qu'ils implémentent des protocoles lourds ou des architectures complexes.

Jini et SSDS qui s'appuient sur un environnement Java, imposent que les équipements embarquent une JVM [36] et utilisent le protocole Java RMI, et donc des mécanismes de sérialisation et de désérialisation. De même UPnP exige l'implémentation de protocoles standards conçus pour des équipements non mobiles et nécessite l'utilisation de parser XML[37] pour manier les documents XML, ce qui n'est pas très adéquat pour des unités mobiles.

2.4.4 Ressources du réseau

Les ressources du réseau reflétées par la bande passante, doivent être prise en considération, surtout si l'approche est décentralisée, car dans ce cas les communications se font en partie avec des technologies réseaux sans fil caractérisées par une bande passante limitée.

Jini et SSDS, basés sur la technologie du code mobile et bien qu'ils adoptent une approche centralisée, consomment beaucoup de bande passante [14]. Les protocoles décentralisés qui n'implémentent pas de mécanismes pour la gestion du trafic tels que UPnP [38] [33] et Salutation consomment d'avantage de bande passante.

---

[36] Java Virtual Machine

[37] analyseur syntaxique

[38] Au démarrage du système, le client UPnP de Windows Me essaye de localiser tous les services du réseau, générant ainsi n*n paquets pour n services et un client

2.4.5 Possibilité de déclarer le service avec les attributs nécessaires

Le protocole de découverte de service doit nous permettre de déclarer notre propre service avec les attributs nécessaires pour son utilisation. Ceci n'est pas possible avec Salutation qui impose une restriction sur les types et les attributs des services.

Les protocoles qui semblent donc les plus adaptés pour l'auto-configuration du plan de gestion sont : SLP et DNS-SD (mDNS pour le cas décentralisé). Nous avons opté pour DNS-SD pour les raisons suivantes :

- DNS-SD emploi des mécanismes performants pour la gestion du trafic dans le cas décentralisé.
- DNS-SD, contrairement à SLP, emploie les deux modèles Pull et Push. Il serait intéressant d'exploiter le "Push" qui permet de générer moins de trafic sur le réseau, et diminue la charge CPU de l'entité qui l'administre [34].
- DNS-SD exploite une architecture supportée par tous les équipements du réseau.
- DNS-SD employé avec le mDNS présente un protocole simple conçu pour les réseaux déployés spontanément [35].

2.5 Conclusion

Dans ce chapitre, nous avons présenté une synthèse des principales fonctionnalités offertes par les protocoles de découverte de services.

Des travaux de recherche dans ce domaine se poursuivent, et visent de nouveaux paradigmes, concernant spécialement la découverte de service dans les réseaux ad hoc. Parmi ces protocoles on retrouve : SDP[39][36] et SPDP [40][37].

Ces protocoles spécifiques aux réseaux ambiants ne peuvent être déployés dans des réseaux fixes, vu qu'ils se basent sur des mécanismes de routage propres aux réseaux ad hoc.

Cependant, une entité mobile peut être connectée à un réseau mobile, comme elle peut l'être à travers un réseau fixe, pour cela nous avons choisi un protocole générique qui peut être appliqué dans les deux cas.

[39] Service Discovery Protocol for Bluetooth Networks
[40] Secure Pervasive Discovery Protocol

3

La gestion des réseaux

3.1 Introduction

Les systèmes de gestion NMS [41] sont basés sur cinq standards : SNMP[42] (Standard de l'Internet), CMIP[43] (Standard de l'OSI[44]), TMN [45], IEEE (Standard des LANs et MANs) et la gestion par Web [38]. D'autres travaux continuent de voir le jour, entre autres : JMX et NetConf [39][40].

Cependant SNMP qui est le protocole de référence en matière de gestion, reste le plus déployé à cause de la simplicité de son architecture, et de son implémentation.

Pour illustrer le rôle des protocoles de gestion, nous allons commencer par présenter les concepts fondamentaux de SNMP, ensuite, nous allons montrer comment se fait la configuration du système de gestion. Mais tout d'abord, commençons par présenter le modèle organisationnel, qui décrit les enjeux de la gestion, et qui est adapté à tous les protocoles, y compris SNMP.

3.1.1 Le modèle organisationnel

Le modèle organisationnel décrit les composants du système de gestion, leurs rôles et leurs relations entre eux [38]. On retrouve donc :

- Les périphériques du réseau (hub, switch, routeur etc) exécutant un agent de gestion. Cet agent est une entité logicielle qui connaît les paramètres du périphérique sur lequel il s'exécute, et dont le rôle est de superviser localement ce dernier. L'agent détient une base d'informations de gestion.

- Le manager (gestionnaire) : est le processus hébergé sur une ou plusieurs stations de gestion. Ce manager administre le réseau à distance, à travers des commandes lancées par l'opérateur ou au moyen de scripts. Le manager détient une vue globale sur l'ensemble des équipements à gérer et dont les paramètres sont stockés dans une base de données de gestion MDB [46].

  Ces paramètres sont des données statiques associées à chaque équipement. Il peut y avoir par exemple : son adresse IP, son type, le type de la base d'informations implémenté etc [47]. Le manager doit aussi stocker le fichier de spécification de la base d'informations de gestion, qui lui permet d'interroger l'agent.

Le modèle Manager/Agent est un modèle semblable au modèle Client/Serveur, où le manager représente le client et l'agent représente le serveur. Ce modèle est basé sur une coopération bidirectionnelle représentée par une relation m :n, dans le sens où le manager peut gérer plusieurs agents,

[41] Network Management System
[42] Simple Network Management Protocol
[43] Common Management Information Protocol
[44] Open System Interconnection
[45] Telecommunication Management Network
[46] Management Data Base
[47] Le but de cette base de données est d'éviter de demander ces paramètres à chaque fois

et un agent peut être géré par plusieurs managers (on parle alors de la gestion distribuée) [41].

L'assignation des rôles n'est pas toujours statique. En effet, un manager peut acquérir le rôle d'un agent dans le cas où il est administré par un autre manager, ceci est appelé en gestion : MOM (Manager Of Manager).

3.2 Le protocole SNMP

SNMP fait partie de la suite des protocoles TCP/IP, il utilise le protocole de transport UDP[48] et existe en trois versions. La dernière (version 3) a été proposée suite à la hausse continuelle des attaques, qui a fait apparaître le besoin de sécuriser ce protocole, tout en restant compatible avec les anciennes versions (v1 et v2) . Les échanges SNMP se font soit par :

- Scrutation (*polling*) des équipements : Le manager envoie sa requête, puis collecte les informations à travers les réponses des agents, et par la suite détermine l'opération à entreprendre.
- Notification suite à un évènement particulier : L'agent peut communiquer de façon asynchrone et sans être sollicité, en envoyant ses notifications (appelées aussi alertes ou traps) au manager.

Chaque agent maintient une base d'informations de gestion (MIB), et c'est le protocole de gestion qui permet de suivre et modifier leurs valeurs.

3.3 La MIB : Management Information Base

La MIB est une base de données virtuelle dans le sens où SNMP ne définit pas ses caractéristiques physiques, mais plutot son organisation logique, conforme à la structure SMI[49] [42]. La MIB SNMP est une collection de données (appelés objets gérés) inhérentes à chaque agent, et qui peuvent être observées et contrôlées à distance par le manager.

Chaque objet géré est décrit par un attribut pouvant prendre certaines valeurs, et par un paramètre qui précise les droits d'accès à cet objet (lecture seule, lecture/écriture, accessible/non accessible etc). Le formalisme utilisé pour spécifier les objets gérés est la syntaxe ASN.1 [50][43].

La MIB a une organisation hiérarchique, dans la quelle chaque objet géré est adressé par un OID [51] qui correspond au chemin depuis la racine jusqu'à cet objet.

3.4 Le protocole d'échange :

Le protocole SNMP définit trois sortes d'intéractions entre le manager et les agents (cf.figure 3.1) :

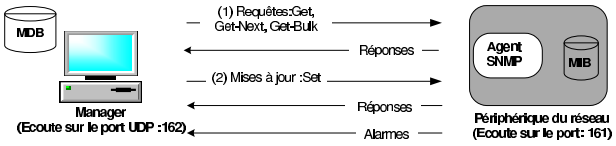

FIG. 3.1 – Les échanges SNMP

[48] User Datagram Protocol
[49] Structure Management of Information
[50] Abstract Syntax Notation
[51] Object IDentifier

1. Les requêtes, produites par le manager, afin de consulter les valeurs des objets gérés. On retrouve les primitives : GET, GET-Next et GET-Bulk.

2. Les mises à jour, permettant à la station de gestion d'affecter de nouvelles valeurs au objets gérés, à travers la primitive : SET.

3. Les alarmes, initiées par les agents suite à l'occurence d'un évènement spécifique tel qu'un problème de communication sur une interface, ou bien suite à sa réinitialisation. Les notification se font à travers la primitive : TRAP.

3.5 Configuration du plan de gestion :

Comme dans tout autre système, les entités participantes dans le plan de gestion doivent être configurées, afin de pouvoir interopérer et accomplir leurs tâches.

Cette configuration concerne donc le manager et les agents. Elle consiste pour l'agent, à établir les paramètres nécessaires à son administration, c'est à dire les paramètres qui lui permettent de savoir par qui il est supervisé, et à qui il doit envoyer ses alarmes. Pour le manager, elle se résume à l'identification de tous les paramètres relatifs à chaque agent pour pouvoir l'administrer. Ces paramètres sont ensuite rajoutés par le manager à la MDB pour des utilisations ultérieures.

Aujourd'hui, la configuration du plan de gestion se fait manuellement que ce soit pour le manager ou bien pour l'agent.

3.5.1 Configuration des agents :

Chaque agent nécessite l'intervention de l'administrateur réseau afin d'établir les paramètres de configuration adéquats.

Ces paramètres dépendent bien sûr du protocole de gestion utilisé. On retrouve par exemple dans le cas de SNMPv1 et SNMPv2 : l'adresse IP ou l'URL du manager, l'adresse IP de la destination des alarmes, et puis les noms de communautés qui permettent le contrôle d'accès (on retrouve deux noms de communautés : l'une publique pour l'accès en lecture et l'autre privée pour l'accès en lecture/écriture).

Bien qu'il soit possible dans SNMP d'avoir un système central intermédiaire qui permette de configurer l'ensemble des agents du réseau [44] [45], il est toujours considéré comme un système qui nécessite une configuration manuelle [46], puisqu'il faut d'abord configurer le système intermédiaire pour fournir les paramètres adéquats.

3.5.2 Configuration du manager :

La configuration du manager englobe le processus de découverte du réseau, et l'identification des paramètres nécessaires au processus de gestion [38] :

3.5.2.1 Le processus de découverte du réseau :

La découverte du réseau permet au manager de récupérer les adresses IP des différents équipements. Cette découverte se fait soit par :

a- Balayage des tables ARP : Le manager lit périodiquement les entrées des tables ARP[52] de tous les routeurs afin de récupérer les adresses IP des différents équipements. Ceci n'est pas vraiment correct puisqu'il peut y avoir des entités qui font partie du réseau et dont les adresses IP

[52] Adress Resolution Protocol

ne figurent pas dans les tables ARP (c'est le cas par exemple d'une entité qui a expirée i.e qui n'a pas communiqué depuis longtemps ou bien qui n'a jamais communiqué à travers le réseau).

b- Ping en broadcast : Le manager envoie périodiquement un Ping en broadcast, et collecte ensuite les réponses des différents équipements.

Contrairement à la découverte par tables ARP, le ping en broadcast permet au manager de récupérer les adresses IP de toutes les machines du réseau, mais il cause cependant la saturation momentanée des liens due aux réponses envoyées simultanées. De plus le manager récupère les adresses de tous les équipements, même ceux qui n'implémentent pas un agent de supervision (ce qui est similaire à la découverte par balayage des tables ARP).

3.5.2.2 L'identification des paramètres nécessaires au processus de gestion :

Pour chaque agent que le manager a découvert, il va collecter ensuite tous les paramètres qui lui sont nécessaires pour commencer à l'administrer, entre autres : le type de MIB implémenté, la version du protocole supporté, et puis le modèle de sécurité et de contrôle d'accès si le protocole de gestion est sécurisé.

Pour cela le manager procède par *polling*, c'est à dire qu'il envoie plusieurs requêtes et collecte ensuite les réponses des agents. Par exemple, dans le cas du protocole SNMP, une fois que le manager a récupéré l'adresse IP de l'agent, il envoie une requête "GET" sur le numéro de port 162 pour savoir si l'équipement implémente le protocole SNMP, ensuite il envoie une autre requête "GET-Next" pour récupérer le type de MIB implémenté, et puis, pour qu'il puisse continuer à interroger l'agent, il doit installer le même fichier de spécification de la MIB (la méthode de récupération de ce fichier est indépendante du protocole de gestion)[47].

Cependant, certains de ces paramètres ne sont pas pris en considération par le protocole de gestion. En effet,il n'exite pas de requête spécifique pour demander à l'agent par exemple le modèle de sécurité déployé. De plus, en interrogeant les agents de cette manière, le manager suppose le déploiement d'un protocole de gestion spécifique, ce qui n'est donc pas opérationnel dans le cas d'un système hétérogène qui supporte plusieurs protocoles de gestion.

3.6 Le monde de l'auto-configuration

Parmi les travaux qui ont déjà été menés dans le domaine de l'auto-configuration, on retrouve : DHCP, Auto IP, auto-configuration IPv6, et puis certains travaux sur l'auto-configuration des routeurs. Ces mécanismes permettent à une machine d'obtenir dynamiquement et automatiquement sa configuration réseau (adresse IP, masque de sous réseau, adresse du serveur DNS etc.) c'est à dire les paramètres qui lui sont nécessaires pour communiquer .

En ce qui concerne les travaux qui ont ciblés l'auto-configuration du système de gestion, on retrouve :

3.6.1 Le projet NESTOR (Network Self managemenT and ORganization) :

Ce projet définit une architecture pour la configuration automatique de tous les équipements du réseau (et pas seulement les entités du plan de gestion)[44], une telle approche est basée sur une entité centrale RDS [53] qui doit être configurée de telle façon à définir les paramètres de configuration à appliquer à chaque entités. De plus, elle reflète la configuration de chaque équipement à tout instant.

[53]Resource Directory Server

Cette approche n'est pas adaptée pour les réseaux déployés spontanément car elle se base sur une entité centrale et elle nécessite une préconnaissance sur le réseau. De plus elle est implémentée en Jini et présente donc toute les limites de ce protocole que l'on a déjà identifiées .

3.6.2 L'option du DHCP pour la configuration des agents SNMP :

Cette approche [45] consiste à peupler le serveur DHCP, avec les paramètres nécessaires à la configuration des agents SNMP. Elle est donc restreinte au protocole SNMP. Elle ne permet pas au manager de connaître les entités à administrer donc il doit procéder d'une façon manuelle. Elle ne peut également pas être appliquée dans un réseau déployé spontanément puisqu'elle se base sur une entité centrale qui est le serveur DHCP. Ce serveur doit être configuré pour fournir ces paramètres et doit être toujours présent dans le réseau, ce qui n'est pas évident dans un environnement dynamique.

Donc, en résumé, les différents travaux proposés nécessitent tous une certaine préconfiguration, et se font par l'intermédiaire d'une entité centrale. Alors que dans un environnement dynamique, il n'est bien sûr pas possible d'avoir une entité présente d'une façon permanente, pour jouer le rôle de recueil des différentes configurations.

Pour les travaux qui ont cependant exploité les protocoles de découverte de services dans le plan de gestion : on retrouve une approche qui utilise le protocole de découverte de service *Jini*[48], pour l'administration des services. Dans cette approche les agents s'enregistrent au niveau du LUS et le manager fait de même, ensuite le manager va souscrire pour recevoir les notifications pouvant se produire sur les agents, ces notifications vont remplacer les traps SNMP.

Bien que cette approche permette aux services de savoir vers qui envoyer les traps en cas de problème, elle ne permet pas l'hétérogénéité des protocoles de gestion, vu que son but principal est l'administration par Jini.

Cette approche ne vise que l'administration des services et non pas tous les équipements du réseau, et se base sur une architecture centralisée complexe qui ne peut être utilisée dans un réseau déployé spontanément. Enfin elle est restreinte à Java et présente toutes les limites du protocole Jini.

On retrouve aussi une version préliminaire[54] qui propose d'utiliser le protocole SLP pour la découverte des serveurs WBEM (agent WBEM) par le client WBEM (manager).

Cette approche définit une relation directionnelle qui permet au manager de connaître les agents WBEM mais pas le contraire.

On mentionne aussi que les services (agents WBEM) annoncent leur description, ce qui n'est pas possible dans une approche décentralisée avec SLP qui n'emploie que le mode Pull et pas de Push. Ceci ne permet donc pas une bonne gestion de la charge CPU du manager.

3.7 Conclusion

La configuration du système de gestion qui se fait aujourd'hui manuellement présente plusieurs limites : elle nécessite la présence d'un administrateur réseau, elle entraîne beaucoup d'échanges rien que pour connaître les paramètres nécessaires pour commencer la supervision, elle nécessite une préconnaissance du protocole de gestion implémenté par l'agent etc. En plus de toutes les limites qu'on vient d'identifier, d'autres contraintes liées aux types des réseaux (dynamicité, absence d'infrastructure fixe...) font que le besoin de l'auto-configuration se fait vraiment ressentir.

[54]DMTF WBEM Discovery Using the Service Location Protocol 1.0.0 January 2004

4

Modèle de couplage : Infrastructure de gestion/Découverte de services

4.1 Introduction

Le modèle de couplage nous permet d'auto-configurer les entités du plan de gestion en utilisant le protocole de découverte de service *DNS-SD* et le protocole de gestion *SNMP*.

Dans ce modèle (cf. figure 4.1) nous avons d'un côté les besoins de l'auto-configuration, et de l'autre les mécanismes offerts par les protocoles de découverte de services et qui permettent de répondre à ces besoins.

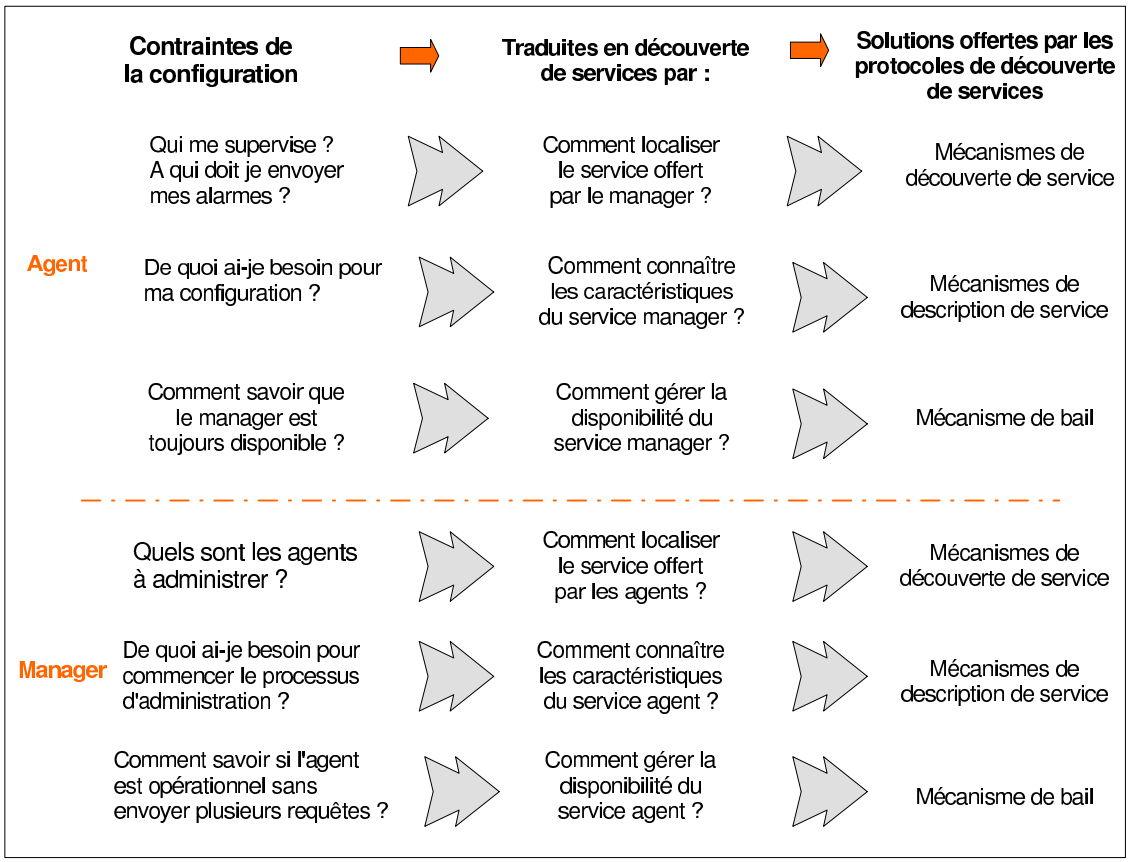

FIG. 4.1 – Modèle de couplage : Infrastructure de gestion/Découverte de services

Dans ce modèle on définit *"l'auto-configuration"* comme un service particulier, et on détaillera par la suite les points suivants :

- La distribution des rôles clients/services entre le manager et l'agent.
- La description du service d'auto-configuration.
- Le processus d'auto-configuration.
- La gestion de la dynamicité.

Nous allons donc appliquer ce modèle dans une approche décentralisée (basée sur le mDNS) qui est l'approche la plus adaptée aux réseaux ambiants et dynamiques déployés spontanément.

4.2 Distribution des rôles

Dans ce modèle une même entité peut représenter à la fois le client et le service. En effet, pour l'auto-configuration des agents, le manager représente le service alors que les agents représentent les clients, et pour l'auto-configuration du manager, les agents représentent les services et le manager représente le client. (voir tableau 4.1).

	Service	**Client**
Auto-configuration du manager	Agent	Manager
Auto-configuration de l'agent	Manager	Agent

TAB. 4.1 – Distribution des rôles

4.3 Description du service "auto-configuration"

La description du service "auto-configuration" doit être expressive pour permettre à l'agent de choisir le manager adéquat. Par manager adéquat on sous entend le manager dont les attributs sont les mêmes que ceux recherchés par l'agent, c'est à dire le manager qui supporte le même protocole de gestion que l'agent, la même version, le même modèle de sécurité etc. Ceci est très important dans le cas d'un système hétérogène (cette contrainte d'hétérogènité caractérise souvent les réseaux ad hoc). Cette description doit aussi permettre au manager de choisir les agents à administrer.

La description des services englobe les enregistrements PTR, SRV et TXT.

4.3.1 Description du service "Manager"

Dans l'enregistrement PTR on retrouve le type du service et le protocole de transport, l'enregistrement SRV comporte l'URL du manager et le numéro de port dédié à la gestion [55] , et enfin l'enregistrement TXT regroupe le reste des informations, et définit les attributs du manager résumés en :

- L'adresse IP de la destination des alarmes.
- Les noms de communautés pour le contrôle d'accès.
- Des précisions sur le mécanisme de sécurité déployé par le manager (dans le cas bien sûr d'un protocole sécurisé) tel que le modèle de sécurité déployé, les algorithmes de cryptage et de hachage utilisés ...etc.

[55] les ports 162 et 161 sont dédiés au protocole SNMP mais on n'est pas obligé de les utiliser, aussi le numéro de port peut être utilisé pour un protocole autre que SNMP et qui n'a pas de port attribué par l'IANNA

On peut aussi rajouter une liste de contacts, sous forme la forme d'une adresse mail ou bien d'un numéro de téléphone.[56]

4.3.2 Description du service "Agent"

Pour la description du service "Agent", l'enregistrement PTR fournit le type du service et le protocole de transport, l'enregistrement SRV comporte son URL et son numéro de port, alors que l'enregistrement TXT regroupe les informations indispensables pour sa supervision, et donc :

- La version du protocole de gestion.
- La MIB implémentée sous forme d'OID.
- Le modèle de sécurité et de contrôle d'accès dans le cas du protocole sécurisé SNMPv3 (SNMPv3 est bâti sur une architecture modulaire et peut donc comporter plusieurs sous modules de sécurité et de contrôle d'accès)
- L'URL du fichier de spécification de la MIB. Pour ce paramètre, on propose que chaque agent stocke une page HTML qui définit la spécification de la MIB qu'il implémente (MIB générique telle que la MIB-2 ou la MIB RMON, ou bien MIB propriétaire telle que la MIB Cisco) [41]. Le manager utilise ensuite cette URL pour télécharger ce fichier.

  La spécification de chaque OID de la MIB doit être clairement définie par une description textuelle car avec le nombre de MIBs qui ne cesse d'augmenter et par conséquent le nombre d'objets gérés[57], le manager ne peut connaître à quoi correspond chaque OID. Cette problématique a déja été exposée dans [47] où on propose de la résoudre par un stockage des fichiers de spécification des OIDs dans des répertoires distribués.

4.4 Processus d'auto-configuration du manager

L'auto-configuration du manager se fait à travers le service *"Agent"* et permet :
- L'automatisation du processus de découverte du réseau.
- L'identification des paramètres de configuration, nécessaires au manager pour administrer l'agent.

1- L'automatisation du processus de découverte du réseau :
Les réseaux d'aujourd'hui sont caractérisés par une très forte dynamicité. Les entités agents peuvent joindre et quitter le réseau d'une façon fréquente, provocant ainsi une grande charge dans les phases de découverte du réseau. La découverte poussée d'habitude à l'initiative du manager, n'est donc pas appropriée dans ce cas.

Afin de limiter le trafic généré par cette dernière et de diminuer la charge CPU du gestionnaire, on va déléguer cette partie du travail aux agents, c'est à dire que la découverte du réseau va être prise en charge par les agents qui vont s'annoncer périodiquement en utilisant le modèle *Push*.

2- Identifications des paramètres de configuration :
L'annonce de l'agent contient toute la description de son service d'auto-configuration, c'est à dire qu'elle encapsule les enregistrements PTR, SRV et TXT.

Donc au lieu que l'agent annonce juste le type de son service, et attende par la suite que le manager le contacte pour connaître ses différentes capacités et sa localisation, l'agent va publier toute sa description en une seule fois.

[56] D'autres données peuvent être rajouté à cet enregistrement et dépondent de la distribution de l'outil utilisé pour la gestion
[57] 150 MIB maitenues par l'IETF et 13000 objets gérés, sans compter ceux des MIBs propriétaire

Le manager va ensuite extraire toutes ces données qu'il rajoute dans sa MDB. Cette base de données est maintenue à jour automatiquement à travers les mises à jours des RRTTLs de chaque agent (voir section 4.2.5).

4.5 Processus d'auto-configuration des agents

Le service *"manager"* permet la configuration de tous les paramètres essentiels à l'agent pour qu'il puisse être administré. Le processus d'auto-configuration se déroule comme suit :

1. L'agent envoie une requête PTR sur l'adresse attribuée au mDNS (Pull) pour la découverte du manager adéquat. Dans sa requête il mentionne qu'il veut recevoir en réponse les informations supplémentaires du record SRV et TXT ce qui est possible dans le DNS-SD (ceci permet de limiter le trafic sur le réseau).
2. Suite à ce *Pull*, le manager adéquat va s'annoncer en multicast (Push). Ceci permet aux autres agents qui viennent de rejoindre le réseau de s'auto-configurer sans relancer de nouvelles requêtes de découverte du manager.
3. L'agent récupère ensuite les données nécessaires à sa configuration qu'il rajoutera dans son fichier de configuration (le reste des informations tel que la liste des contacts est stocké dans un autre fichier dédié à cette tâche).

4.5.1 Gestion de la dynamicité

Le mécanisme de bail est très utile pour détecter l'indisponibilité d'une entité donnée (quelle soit manager ou agent).

Le protocole SNMP définit une trap *"cold start"* qui indique la mise en marche de l'agent. Cependant, si la machine sur laquelle tourne cet agent tombe en panne ou quitte le réseau, le manager ne peut le savoir que s'il lui envoie une requête et qu'il ne reçoit pas de réponse. Dans ce cas il saura que cette machine n'est pas opérationnelle mais il ne pourra pas en connaître la cause, c'est à dire savoir si elle est éteinte, en panne ou si elle a quitté le réseau.

L'utilisation du mécanisme de bail, reflété dans le DNS-SD par le temps de mise en cache RRTTL, permet de gérer la dynamicité du système. En effet le manager et cela sans qu'il envoie de requête, sera informé si un agent rejoint ou quitte le réseau, et il pourra également détecter si cet agent est en panne par le non renouvellement de son RRTTL.

Le bail est mis à jour périodiquement à chaque chaque 80 % [58] du RRTTL initial. Dans le cas où une entité décide de quitter le réseau avant l'expiration de son bail, elle envoie un message de désenregistrement aux autres entitées.

Cette notion de bail permet au manager de gérer son MDB et par conséquent d'avoir une vue cohérente du système, et permet systématiquement à l'agent de se reconfigurer.

4.5.2 Processus de reconfiguration

Dans le cas où le manager n'est plus disponible, c'est à dire qu'il s'est désenregistré ou qu'il n'est plus opérationnel (il n'a pas mis à jour son RRTTL), ou bien dans le cas où l'agent est une entité mobile qui rejoint un autre réseau ; l'agent doit se reconfigurer en relançant le même processus de configuration, et donc :

[58] Valeur fixée dans le mDNS mais qui peut être définie par rapport à la stabilité du réseau

– Si (RRTTL Manager = 0) ou (désenregistrement du manager) ou (l'agent à rejoint un autre réseau "Détection par changement de l'interface réseau de l'agent") refaire le processus d'auto-configuration.

Si aucun manager ne répond, l'agent utilise la requête *Continuous querying* qui lui permet de détecter la disponibilité d'un nouvel manager.

4.6 Amélioration du Push par le Push dirigé

Le modèle *Push* permet au manager de savoir si un nouvel agent est rajouté au réseau, il lui permet aussi de gérer la disponibilité de cet agent à travers les mises à jour du RRTTL.

Cependant, et comme ces annonces n'intéressent que le manager, il est donc inutile de les envoyer en multicast à l'adresse 224.0.0.251. On utilisera plutôt ce que nous appelons un "Push dirigé".

Dans le "Push dirigé", nous allons modifier les paramètres de l'algorithme de découverte pour que l'annonce ne soit transmise qu'au manager, l'agent va donc exécuter les étapes suivantes :

1. *Récupérer l'URL du manager.*
2. *Tant que (RRTTL Manager != 0) ou (! = désenregistrement du manager) faire :*
   Si la durée de disponibilité de l'agent a atteint 80% du RRTTL Agent : envoyer la mise à jour de l'agent sur l'adresse du manager.

L'agent récupère donc l'URL du manager en utilisant le mécanisme du Pull (voir section 4.2.4), puis il envoie son annonce et ses mises à jour directement à ce manager, qu'il a découvert au lieu d'utiliser l'adresse 224.0.0.251.

Le *"Push dirigé"* réduit le trafic d'une façon très significative, il diminue la charge CPU du gestionnaire en déléguant cette phase de découverte aux agents, et de plus il améliore le temps CPU consommé par les différentes entités du système, si on le comparaît au *"Push"*, car ses annonces ne seront traîtées que par le manager qui l'administre.

En effet, le "Push dirigé" donne le meilleur compromis, comme on peut le constater dans l'exemple suivant (cf. figure 4.2) où on suppose avoir un manager et n agents. Le tableau 4.2 résume la comparaison entre la découverte classique par ping en broadcast, la découverte qu'on propose et qui se base sur le modèle *Push* et enfin l'amélioration par *Push dirigé*.

Exemple

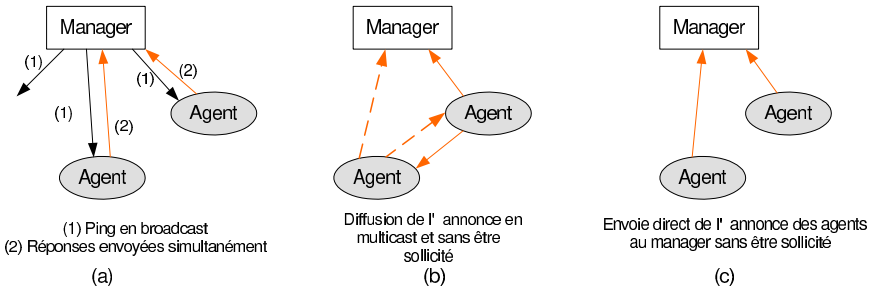

FIG. 4.2 – (a) Découverte classique par ping en broadcast - (b) Découverte par Push - (c) Découverte par Push dirigé

	Découverte du réseau	**Identification des paramètres de gestion**
Approche classique par ping	1 paquet généré par le ping + n paquets de réponses envoyés simultanément	Plusieurs requêtes pour identifier le reste des informations
Approche du Push	n paquets du push qui ne sont pas envoyés simultanément mais qui sont traîtés par toutes les entités du système	une seule annonce comporte toutes les informations nécessaires
Approche du Push dirigé	n paquets du push dirigé qui ne sont pas envoyés simultanément et qui ne sont traîtés que par le manager	une seule annonce comporte toutes les informations nécessaires

TAB. 4.2 – Comparaison entre la découverte par ping en broadcast, par Push et par Push dirigé

4.7 Synthèse

On peut résumer l'auto-configuration du plan de gestion, dans le cas décentralisé en trois phases principales qu'on a schématisé dans la figure ci-après (cf. figure 4.3) :

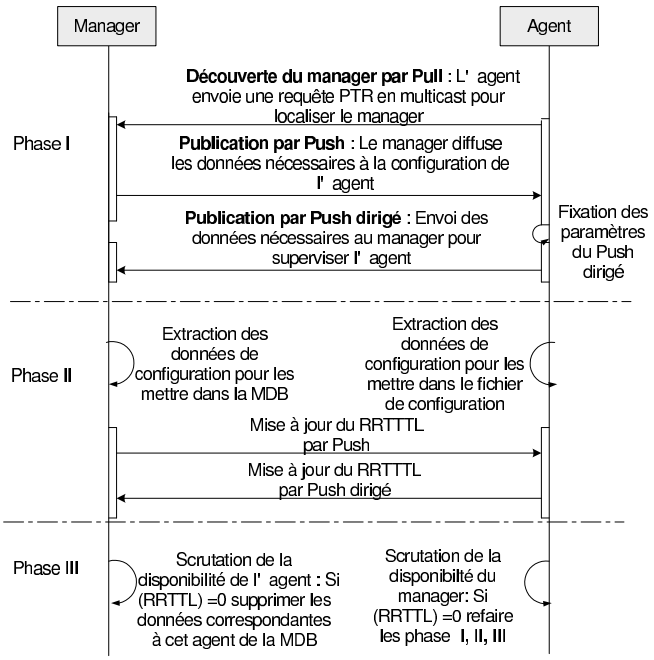

FIG. 4.3 – Auto-configuration du plan de gestion selon une approche décentralisée

1. **Phase de découverte et de publication** : Chaque agent qui rejoint le réseau va utiliser le mécanisme du Pull pour localiser le manager. Il utilise ensuite le mécanisme du Push dirigé, pour l'informer qu'une nouvelle entité est rajoutée au réseau, et pour lui envoyer les données nécessaires pour pouvoir le superviser. De même, le manager s'annonce périodiquement en utilisant le Push.

2. **Phase d'extraction de données** : L'agent récupère tout ce qui est nécessaire à sa configuration, afin de fixer les paramètres de sa configuration et ceux du Push dirigé.

Le manager, quant à lui, rajoute dans sa MBD, toutes les données nécessaires à l'administration de l'agent.

3. **Phase de scrutation** : L'agent continue de scruter la disponibilité du manager, et vice-versa. Dans le cas où le manager n'est plus disponible, l'agent relance une procédure de reconfiguration, et dans le cas où l'agent n'est plus disponible, il sera automatiquement supprimé de la MDB.

4.8 Exemple d'implémentation

Nous avons déployé un prototype de ce modèle de couplage, pour lequel nous avons utilisé :
- le JmDNS-0.2 [49] qui est une implémentation en Java du protocole DNS-SD employé avec le mDNS [59].
- La suite logicielle Net-SNMP-5.1 [50] qui définit un ensemble d'outils de gestion sous linux. Celle-ci regroupe un agent SNMP, divers outils d'interrogation ainsi qu'une bibliothèque de fonctions C.

4.8.1 Description des services

JmDNS utilise la classe *ServiceInfo* pour la description des services. Pour le type des services manager et agent nous avons défini respectivement le type *Auto-Conf-Manager-SNMP* et *Auto-Conf-Agent-SNMP* où nous avons fait apparaître le protocole de gestion (SNMP), pour permettre aux différentes entités de faire un premier filtrage sur ce critère (rappel : il est n'est pas possible de mensionner les attributs du service désiré dans les requêtes DNS-SD).

La version du protocole n'est pas mentionnée dans le type du service puisqu'une même entité SNMP peut supporter les trois versions. Cette dernière est par contre rajoutée à l'enregistrement TXT en plus des autres attributs. Précisant aussi que dans cette description on permet plusieurs valeurs pour le même attribut (ce qui est possible avec le DNS-SD).

La description du service dépend de la version supportée. Dans l'exemple ci-dessous on retrouve une description du service manager supportant la version 1 et 2 (cf. figure 4.4) et 3 (cf. figure 4.5) , et la description du service agent supportant les versions 1 et 2 (cf. figure 4.6). La description des services *"agent"* et *"manager"* que nous avons proposée doit être standardisée pour permettre l'automatisation l'insertion de ces données dans le fichier de configuration de l'agent et la MDB du manager.

```
TXT: Version = v1, v2  // L' attribut version peut prendre plusieur valeurs
     Mod_Secu = Communauté   // Le modèle de sécurité dans le cas de SNMPv1 et v2 est par
                               défaut  communauté c' est à dire basé sur le contôle d' accès
                               par nom de communauté
     rw_Community = Priv   // Nom de communauté privé qui permet l' accès aux objets gérés en
                              lecture et écriture
     ro_Community = Pub    // Nom de communauté privé qui permet l' accès aux objets gérés
                              en lecture seulement
     trap_Community =  mes traps    // Nom de communauté dédié à l' authentification des alarmes
     Sys_Contact = admine@loria.fr  // La liste de contacte sous forme d' adresse mail  du
                                       gestionnaire
     Sys_Location = bureau B250     // Le numéro de bureau du gestionnaire
```

FIG. 4.4 – Description du service d'un manager SNMP supportant les versions 1 et 2

[59] Multicast DNS Service Discovery

```
TXT: Version = v3
     Mod_Secu = USM  // Ce manager supporte le modèle de sécurité USM. USM est un modèle
                        qui permet d' assurer la sécurité SNMP à travers des mécanismes de
                        hachage et de cryptage. Cet attribut peut prendre plusieurs valeurs
     Algo_Cryptage = DES  // Cet attributs dépend du modèle de sécurité déployé.
     Algo_Hachage = SHA-1  // Cet attributs dépend du modèle de sécurité déployé.
     Sys_Contact = admine@loria.fr  // L' adresse mail du gestionnaire
     Sys_Location = bureau B250  // Le numéro de bureau du gestionnaire
```

FIG. 4.5 – Description du service d'un manager SNMP supportant la version 3

```
TXT: Version = v1, v2  // L' attribut version peut prendre plusieur valeurs
     Mod_Secu = Communauté // Le modèle de sécurité dans le cas de SNMPv1 et v2 est par
                              défaut communauté c' est à dire basé sur le contôle d' accès
                              par nom de communauté
     Mib_Type = .1.3.6.1.2.1  // Cet OID définit que l' agent implémente la MIB-2
     Path_MibSpec = /usr/local/ share/ snmp/ mib.html // définit le chemin d' accès au fichier de
                                                         spécification de la MIB
```

FIG. 4.6 – Description du service d'un agent SNMP supportant les versions 1 et 2

4.8.2 Auto-configuration de l'agent

L'agent SNMP se configure au moyen du fichier /usr/local/share/snmp/snmpd.conf. Donc au démarrage de la machine (boot), l'agent SNMP lance un processus de découverte du manager en envoyant une requête PTR de la forme : *PTR : Auto-Conf-Manager-SNMP._tcp.local ?*.

A la réception des annonces des managers, l'agent évalue ces descriptions et choisit le manager adéquat. Il récupère ensuite les différents paramètres de configuration qu'il met dans le fichier snmpd.conf, et initialise le paramètre *"adresse"* pour pouvoir effectuer le Push dirigé. C'est à dire qu'il change la valeur de la variable : MDNS_GROUPE qui prend par défaut la valeur 224.0.0.251 par l'adresse du manager qu'il récupère de l'enregistrement SRV de ce dernier.

Une fois sa configuration établie, l'agent va envoyer la description de son service au manager.

4.8.3 Auto-configuration du manager

A la réception des données nécessaires à l'administration de l'agent, la méthode *add (DNSEntry entries[])*[60] de la classe *DNSCache* qui est la classe responsable de la maintenance de la cohérence du cache JmDNS, va prendre en charge l'ajout d'une nouvelle entrée à la MDB du manager. Mais avant elle doit en premier temps vérifier si c'est une nouvelle annonce ou bien juste une mise à jour.

4.8.4 Gestion de la dynamicité

La classe *DNSRecord* vérifie la validité de chaque entrée du cache JmDNS. Dans le cas où une entrée n'est pas mis à jour ou bien dans le cas où l'entité qui offre le service correspondant à cette entrée décide de se désenregistrer et envoie donc la notification : *unregisterService(ServiceInfo)*, le traitement diffère :

- Pour la manager la méthode *remove de la classeDNSCache* va supprimer cette entrée du cache local et de la base de données de gestion.

[60] méthode responsable de l'ajout de nouvelles entrées au cache JmDNS

– Pour l'agent elle va supprimer aussi cette entrée de son cache local, supprimer son fichier de configuration et par la suite relancer le processus d'auto-configuration.

4.9 Conclusion

Dans ce chapitre nous avons proposé un modèle de couplage entre le protocole SNMP et le protocole DNS-SD employé avec le mDNS. Ce modèle basé sur une approche décentralisée peut être déployé aussi dans un réseau étendu.

En effet, la contrainte qui fait que les protocoles décentralisés ne sont généralement pas déployables dans de grands réseaux est que les annonces et requêtes envoyées en multicast causent une charge importante. Cependant avec le mécanisme du *"Push dirigé"* que nous proposons, nous pourrons réduire cette charge.

5

Conclusions et Perspectives

"En toute chose, il faut considérer la fin." J. de la Fontaine

5.1 Bilan

La configuration du plan de gestion est aujourd'hui une charge importante pour les administrateurs des réseaux modernes. Cette charge ne peut aller qu'en augmentant au regard de la dynamicité et du nombre croissant de noeuds fixes et surtout mobiles dans ces réseaux.

Afin de libérer les administrateurs au maximum de ces tâches et de faire face à ces challenges, les éléments des réseaux et le plan de gestion lui-même devront se doter de capacités d'auto-configuration avancées.

Nous proposons de placer la configuration du plan de gestion au niveau des autres services fournis par un réseau, ce qui nous permet d'utiliser les technologies des protocoles de découverte de services et de les adapter à nos besoins.

De notre étude comparative des principaux protocoles de découverte de services existants, nous avons relevé les critères que le protocole de découverte de services doit fournir pour satisfaire la tâche de l'auto-configuration du plan de gestion, à savoir la prise en compte des capacités du réseau, des entités qui le composent et de la dynamicité du système etc. Cette étude nous a permis de conclure que DNS-SD est le plus approprié dans une perspective d'auto-configuration

Nous avons ensuite proposé un modèle de couplage entre DNS-SD et le protocole SNMP qui lui n'intègre pas les modalités d'auto-configuration. Ce modèle permet à un agent SNMP qui vient de se connecter de détecter de manière automatique le manager du réseau. Le manager, quant à lui, garde une vue cohérente sur le système qu'il gère sans générer un surcoût de gestion en terme de charge CPU et de paquets pour maintenir la topologie du plan de gestion.

Nous avons implémenté une première version de ce modèle qui nous permet l'auto-configuration d'un agent SNMP (sur Linux).

5.2 Perspectives

Il est important de finaliser ce travail en ajoutant à ce modèle des fonctionalités de sécurité devenues incountournables dans la gestion.

Notons également que le modèle que nous avons proposé est générique. Nous l'avons instancié dans le cadre du protocole SNMP, mais il peut s'étendre pour d'autres protocoles de gestion (NetConf ...). Pour cela, il faut identifier les données nécessaires pour chaque protocole et définir les valeurs des différents champs DNS-SD. Ceci permet aux acteurs de choisir les protocoles de gestion correspondants à leurs besoins et leurs capacités.

Pour finir, notre approche d'auto-configuration peut s'appliquer à des réseaux ad-hoc qui sont caractérisés par une grande mobilité et par l'absence d'infrastructure fixe. Cependant une étude du calibrage du bail doit être bien définie en fonction de la stabilité du réseau.

Bibliographie

[1] G.H. Forman and J. Zahorjan. The challenges of mobile computing. vol 27, pp 38-47. Number 4. IEEE Computer Society Press, 1994.

[2] P. Charas. Peer-to-peer mobile network architecture. In *1st International Conference on Peer-to-Peer Computing, pp.55-61.* IEEE Computer Society, August 2001.

[3] C. Bettstetter and C. Renner. A comparison of service discovery protocols and implementation of the service location protocol. In *EUNICE 2000, Sixth EUNICE Open European Summer School, Twente, Netherlands*, September 2000.

[4] California Software Laboratories. Upnp, jini and salutation - a look at some popular coordination frameworks for future networked devices. http ://www.cswl.com/whiteppr/tech/upnp.html.

[5] R.E. McGrath, M.D. Mickunas, and R.H. Campbell. Semantic discovery for ubiquitous computing. (citeseer.ist.psu.edu/485904.html).

[6] C. Lee and S. Helal. Protocols for services discovery in dynamic and mobile networks. In *International Journal Computer Research. Vol 11, No. 1*, 2002.

[7] C. Dabrowski and K. Mills. Analyzing properties and behavior of service discovery protocols using an architecture-based approach, proceedings of working conference on complex and dynamic systems architecture, December 2001. National Institute of Standards and Technology -Gaithersburg USA.

[8] M. Robert. Discovery and its discontents : Discovery protocols for ubiquitous computingg, department of computer science university of illinois urbana-champaign, urbana uiucdcs-r-99-2132, March 2000.

[9] C. Perkins E. Guttman. RFC 2610 : "DHCP Options for Service Location Protocol ", June 1999. Status : Standards Track.

[10] Sun Microsystems : Jini Network Technology. White papers : Jini technology architectural overview, January 1999. http ://www.sun.com/jini/.1999.

[11] Sun Microsystems. Jini technology core platform specification. http ://wwws.sun.com/software/jini/specs/.

[12] Sun Microsystems. Java remote methode invocation rmi. http ://java.sun.com/docs/books/tutorial/rmi.

[13] J. Veizades E. Guttman, C. Perkins and M. Day. RFC 2668 : "Service Location Protocol, version 2 ", June 1999. Status : Standards Track - http ://www.rfc-editor.org/rfc/rfc2608.txt.

[14] J. Govea and M. Barbeau. "results of comparing bandwith usage and latency : Service location protocol and jini". *"Workshop on Ad hoc Communication"*, September 2001. held in conjunction with the Seventh European Conference on Computer Supported Cooperative Work (ECSW 2001)",Bonn, Germany.

[15] A. Miller T. Nixon C. Tai M.D. Wood. Home networking with universal plug and play. *IEEE Communication Magazine, no 12. pp 104-109*, December 2001.

[16] Universal Plug and Play Forum. Upnp device architecture v1.0.1 draft, December 2003. http ://www.upnp.org/resources/documents.asp.

[17] W3C. Simple object access protocol (soap), May 2000. http ://www.w3.org/TR.

[18] Salutation Consortium. Salutation architecture specification. http ://www.salutation.org/spec.

[19] G.G.Richard. Service advertisement and discovery : Enabling universal device cooperation.vol 4. no 5. pp 18-26. *IEEE Internet Computing*, October 2000.

[20] S.E. Czerwinski, Ben Y. Zhao, Todd D. Hodes, Anthony D. Joseph, and Randy H. Katz. An architecture for a secure service discovery service. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 24–35. ACM Press, 1999.

[21] Zero configuration networking. http ://www.zeroconf.org/.

[22] S. Cheshire and M. Krochmal. Dns-based service discovery,internet-draft, draft-cheshire-dnsext-dns-sd-02.txt, February 2004.

[23] B.Wellington. RFC 3007 : Secure Domain Name System (DNS) Dynamic Update, November 2000. Status : Standards Track.

[24] S.Cheshire and M. Krochmal. Multicast dns, internet-draft draft-cheshire-dnsext-multicastdns-03.tx, February 2004. Status : Standards Track.

[25] UDDI. Uddi technical white paper, September 2000. http ://www.uddi.org.

[26] UDDI. Uddi version 2.03 data structure referance uddi committee specification, July 2002.

[27] E. Christensen et al. Web services description language (wsdl)1.1 .technical report, world wide web consortium (2001). (http ://www.w3.org/tr/wsdl), Mars 2001.

[28] J. Beatty G. Kakivaya D. Kemp B. Lovering B. Roe J. Schlimmer G. Simonnet J. Weast. Web service dynamic discovery (ws-discovery). copyright (c) 2004 microsoft corporation, inc. 35 pages (http ://schemas.xmlsoap.org/ws/2004/02/discovery), February 2004.

[29] B. ZHAO. Xset. http ://www.cs.berkeley.edu/ ñavenben/xset.

[30] D. Eastlake. Rfc 2535 :domain name system security extensions, March 1999. Status : Standards Track.

[31] A.Nadalin et al. Oasis web services security : Soap message security - working draft, August 2003.

[32] J.Rosenberg H.Schulzrinne and B.Suter. Ietf : Internet draft : Wide area network service location, November 1997. Status : Standards Track.

[33] A. Friday N. Davies E. Catterall. Supporting service discovery, querying and interaction in ubiquitous computing environments. In *the 2nd ACM international workshop on Data engineering for wireless and mobile access*, 2001.

[34] J.P. Martin-Flatin. La gestion des réseaux ip basée sur les technologies web et le modèle *push. Proc. 3ème Colloque Francophone sur la Gestion de Réseaux et de Services (GRES'99), Montreal, QC, Canada*, Juin 1999.

[35] A. Duda. Ambient networking.tutoriel à l'école d'été rhdm 2002. LSR-IMAG Laboratory. INP Grenoble and CNRS.

[36] J. Karlsson A. Persson. Master thesis mee 01-28, device and service discovery in bluetooth networks. Master's thesis, Australian Telecommunications Reasearch Institute -Blekinge Institute of Technology, June 2002.

[37] Y. Yuan and A. Agrawala. A Secure Service Discovery Protocol for MANET.UMIACS-TR-2003-67. Technical report, MIND Laboratory, Department of Computer Science. University of Maryland, 2003.

[38] M. Subramanian. Network management : Principles and practice, May 2000. Georgia Institute of Technology- Addison-Wesley Publishers. May 2000.ISBN 0-201-35742-9.

[39] R. Enns. Netconf configuration protocol.internet-draft : draft-ietf-netconf-prot-02, February 2004. Network Working Group.

[40] J. Jaen-Martinez. The java management extensions (jmx) : is your cluster ready for evolution ? *J. Parallel Distrib. Comput.*, 60(10) :1341–1353, 2000.

[41] H.G. Hegering S. Abeck B. Neumair. Integrated management of networked systems : concepts, architectures, and their operational application, 1998.morgan kaufmann publishers inc. isbn 1-55860-571-1, 1998.

[42] M.T. Rose and K. McCloghrie. RFC 1155 : Structure and identification of management information for TCP/IP-based internets, May 1990. See also STD0016. Obsoletes RFC1065 . Status : STANDARD.

[43] Abstract syntax notation one (asn.1). International Organization for Standardization and International Electrotechnical Committee, 1987. Draft Addendum 8824/DAD 1.

[44] Y. Yemini A. V. Konstantinou D. Florissi. Nestor : An architecture for network self-management and organization. *IEEE :SELECTED AREAS IN COMMUNICATIONS*, 2000.

[45] M. Bakke. Dhcp option for snmp notifications - internet-draft, draft-bakke-dhc-snmp-trap-01.txt, November 2002. Status : Standards Track.

[46] D. Galand and O. Marcé. A functional architecture for self-aware routers. *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC), pp 352-356*, March 2004.

[47] S. Boros B. Helthuis A. Pras. Distributed mib object information service, proceedings of the high speed networking 2001 spring workshop, May 2001. University of Twente, The Netherlands.

[48] G. Aschemann and P. Hasselmeyer. A looosely coupled federation of distributed management services. ito tr-00-09. Technical report, Darmstadt University of Technology, March 2001.

[49] Jmdns. http ://sourceforge.net/projects/jmdns.

[50] netsnmp. http ://netsnmp.sourceforge.net.

6

Annexe

6.1 Comparaison générale des protocoles de découverte de services

	SLP	**Jini**	**UPnP**	**SSDS**
Développé par	IETF	Sun Microsystem	Microsoft	Ninja
Langage de programmation	Indépendant	Java	Indépendant	Java
Réseau de transport	TCP/IP	Indépendant	TCP/IP	Indépendant
Approche suivie	DA (optionnel)	LUS (optionnel)	Décentralisé	Serveurs SSDS
Modèle utilisé dans le cas distribué	Pull	-	Pull/Push	Push bien que c'est un protocole centralisé
Adresse du protocole et numéro de port	239.255.255.255 Port :427	Jini-announcement : 224.0.1.84 Jini-request : 224.0.1.85 Port :4160	239.255.255.250 Port :1900	-
Description du service	Service template	Interface + attributs	Document XML	Document XML
Protocole d'invocation du service	Mentionné dans le service URL	Java RMI	SOAP	Authenticated RMI
Protocole pour la recherche de services	SLP	Java RMI	SSDP	Authenticated RMI
Passage à l'échelle	WASRV	Non spécifié	Non	Serveurs hiérarchique
Sécurité	Intégrité des descriptions des services	Restreinte à celle offerte par Java	Non	Intégrité, confidentialité et authentification
Leasing	Life time	Lease	Oui	Oui
Notification	Non	Distributed Event	GENA	Non
Requête selon les attributs	Oui	Oui	Non	Oui

TAB. 6.1 – Comparaison générale des protocoles de découverte de services

	DNS-SD	**WS-Discovery**	**Salutation**	**UDDI**
Développé par	IETF	Microsoft Canon, Intel, BEA	Consortium d'entreprises	IBM, Microsoft et Ariba
Langage de programmation	Indépendant	Indépendant	Indépendant	Indépendant
Réseau de transport	TCP/IP	TCP/IP	Indépendant	TCP/IP
Approche suivie	Serveurs DNS (optionnel)	Discovery Proxy (optionnel)	SLM (comme proxy)	UBRs IBM, Microsoft et SAP
Modèle utilisé dans le cas distribué	Pull/Push	Pull/Push	Pull	-
Adresse du protocole et numéro de port	224.0.0.251 Port : 5353	239.255.255.250 Port :3702	-	-
Description du service	Record SRV Record TXT	Document XML	Functional Unit Description Record	WSDL
Protocole d'invocation du service	Mentionné dans le record SRV	Mentionné dans la description du service	Dépend du mode utilisé (native ou Salutation)	Mentionné dans le WSDL
Protocole pour la recherche de services	DNS	SOAP	SunRPC	SOAP
Passage à l'échelle	Serveurs hiérarchiques	Non	Non	Nuage UDDI
Sécurité	DNS SEC	-	Authentification	Intégrité des descriptions des services
Notification	Non	Implicite (hello, bye)	Oui	Dans la version bêta
Leasing	RR TTL	Implicite (hello, bye)	Explicite	Non
Requête selon les attributs	Non	Non	Oui	Non

TAB. 6.2 – Comparaison générale des protocoles de découverte de services