UNIVERSITÉ DU QUÉBEC À MONTRÉAL

ADAPTATION D'UN PROTOCOLE DE DÉCOUVERTE DE SERVICES POUR LES RÉSEAUX AD-HOC

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR
HAKIM BOUKOUNA

SEPTEMBRE 2008

REMERCIEMENTS

Je tiens d'abord à remercier chaleureusement mon directeur de recherche, monsieur Abdellatif Obaid, pour la confiance qu'il m'a accordée. Son encadrement, sa patience et ses pertinents conseils qui ont été précieux pour moi.

Je remercie également monsieur Azzedine Khir, maîtrise en informatique à l'Université du Québec à Montréal, de m'avoir aidé à comprendre le fonctionnement de son protocole de découverte de service « SEDIRAN ».

Finalement, je désire souligner le soutien inconditionnel de ma femme dans les bons tout comme dans les mauvais moments, je vous remercie chaleureusement.

Je dédie ce mémoire à mes parents.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

ACRONYMES

AODV	Ad-hoc On Demand Distance Vector
API	Application Programming Interface
ARP	Address Resolution Protocol
DCP	Device Control Protocols
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DNS-SD	DNS Service Discovery
GENA	General Event Notification Architecture
GSD	Group-based Service Discovery
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPMU	HTTP Multicast over UDP
HTTPU	HTTP Unicast over UDP
HVAC	Heating, ventilation and air-conditioning
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IGMP	Internet Group Management Protocol
IP	Internet Protocol

IRDA	Infrared Data Association
JVM	Java Virtual Machine
Manet	Mobile Ad-hoc Network
OWL	Ontology Web Language
PDA	Personal digital assistant
RMI	Remote Method Invocation
RPC	Remote Procedure Calls
SDK	Software Development Kit
SDP	Service Discovery Protocol
SEDIRAN	Service Discovery and Interaction with Routing Protocols in Ad-hoc Network
SLP	Service Location Protocol
SOAP	Simple Object Access Protocol
SPDP	Secure Pervasive Discovery Protocol
SSDP	Simple Service Discovery Protocol
TCP	Transmission Control Protocol
UDA	Universal Data Access
UDP	User Datagram Protocol
UPnP	Universal Plug and Play
URL	Uniform Resource Location
UUID	Universal Unique Identifier
VCR	video cassette recorder
WSDL	Web Service Definition Language
XML	Extensible Markup Language

RÉSUMÉ

Le but de ce mémoire est de concevoir une nouvelle version du protocole de découverte de services UPnP (*Universal Plug and Play*) qui pourra fonctionner dans les réseaux ad-hoc.

UPnP est un protocole de découverte et de contrôle automatiques de services sur les réseaux fixes filaires et sans fil. Il est auto configurable et basé sur des protocoles standards.

Le SEDIRAN (*Service Discovery and Interaction with Routing Protocols in Ad-hoc Network*) est un protocole de découverte de services dans les réseaux ad-hoc qui est au-dessus du protocole de routage réactif AODV (*Ad-hoc On Demand Distance Vector*).

Dans ce mémoire, nous exposons les différents protocoles de découverte de services dans les réseaux filaires ainsi que la problématique de la découverte de services dans les réseaux ad-hoc et quelques protocoles existants. Puis, nous détaillons le protocole de découverte de services dans les réseaux filaires et sans fil avec points fixes UPnP et qui fait l'objet d'une modification pour étendre son champ d'utilisation dans les réseaux ad-hoc et nous détaillons le protocole de découverte de services dans les réseaux ad-hoc SEDIRAN.

Finalement, nous présentons la stratégie, le principe de fonctionnement et le prototype du nouveau protocole de découverte de services dans les réseaux ad-hoc avec le protocole UPnP.

CHAPITRE I

INTRODUCTION

1.1 Réseaux ad-hoc et la découverte de services

Un réseau mobile est un système composé de terminaux mobiles qui permet à ses utilisateurs d'accéder aux services indépendamment de leur position géographique. Il existe deux catégories de réseaux mobiles ou sans fil : les réseaux avec infrastructure et les réseaux sans infrastructure.

- Les réseaux avec infrastructure utilisent des routeurs pour la connexion par des liens radios aux nœuds mobiles environnants.
- Les réseaux sans infrastructure (réseaux ad-hoc), dont les nœuds sont reliés par des liens radio comme l'IEEE 802.11, n'utilisent aucun point d'accès.

Certains sites fixes possèdent une interface de communication sans fil pour la communication directe avec les nœuds mobiles proches (voir Figure 1.1) [4].

Un réseau ad-hoc est un réseau sans fil à multi-saut composé d'un ensemble de nœuds mobiles qui n'exigent pas une infrastructure préexistante du réseau. Il est employé dans les situations où la création d'un réseau ad-hoc est nécessaire (voir Figure 1.2).

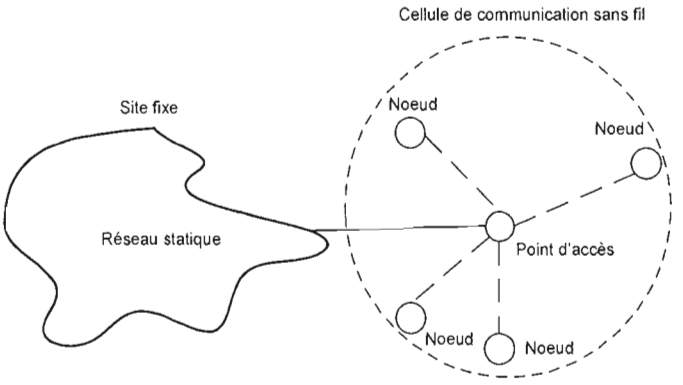

Figure 1.1 - Modèle des réseaux mobiles avec infrastructure

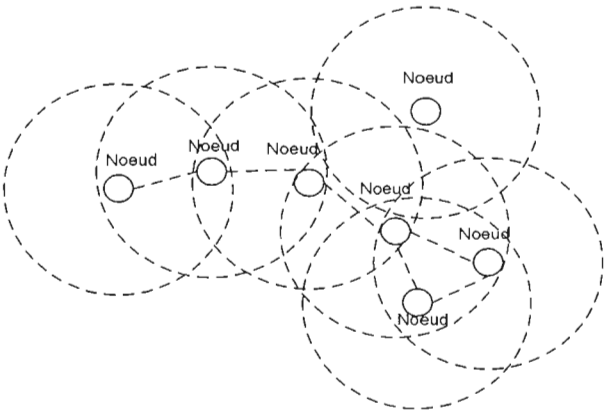

Figure 1.2 - Modèle des réseaux mobiles sans infrastructure (réseau ad-hoc)

Les utilisations possibles des réseaux ad-hoc sont nombreuses. On peut citer, à titre d'exemple, leurs applications dans le domaine militaire, les opérations de secours, les missions d'exploration et le milieu de l'éducation. Par exemple, les étudiants utilisant les ordinateurs portables pour échanger l'information des cours.

Dans ce genre de réseau, chaque nœud est libre de se déplacer aléatoirement et se relie arbitrairement à d'autres nœuds. Chaque nœud joue le rôle de routeur et de serveur. Ainsi, une telle topologie du réseau sans fil peut changer rapidement et de manière imprévisible.

La contrepartie des réseaux filaires est les réseaux ad-hoc qui ont une bande passante limitée et sont plus sujets à des interférences, de l'effacement et du bruit.

La bande passante devrait être utilisée efficacement et la puissance de batterie devrait être conservée. Toute contrainte mentionnée ci-dessus apporte de nouveaux défis non seulement au routage du réseau, mais également aux protocoles de couches applications tels que les protocoles de découverte de services. Au cours des dernières années, beaucoup d'efforts ont été mis pour inventer des protocoles de routages ad-hoc efficaces sous ces contraintes.

La découverte de services dans les réseaux est un des objectifs les plus importants dans le développement des réseaux informatiques. Grâce aux progrès remarquables des technologies de la communication, les terminaux peuvent être désormais connectés les uns aux autres pour annoncer et rechercher des services.

La découverte de services est définie pour résoudre le problème de localisation automatique des différents services dans un réseau. Les services sont des entités offertes à l'utilisateur par les nœuds du réseau. Comme par exemple, le service peut être une imprimante, un scanneur d'images ou un logiciel de comptabilité.

Il y a actuellement une variété de protocoles de découverte de services dans le monde des réseaux. Les plus connus sont : SLP (*Service Location Protocol*) défini par IETF [1]; Jini, défini par Sun Microsystems [2]; SDP, défini par Bluetooth Forum [5]; et UPnP défini par Microsoft [14] [15].

Le protocole de découverte de services UPnP a des caractéristiques (zéro-configuration, basé sur des protocoles standards, indépendant de la plate-forme et indépendant de la couche physique) qui lui permettent d'élargir son champ d'action en devenant fonctionnel dans les réseaux ad-hoc. Le protocole est décrit en détail au chapitre III.

La découverte de services dans un environnement ad-hoc exige une approche décentralisée, où chaque terminal doit être autonome pour annoncer ses services. Le rôle d'un protocole de découverte de services est de faire l'annonce, la recherche et l'invocation de services. Chaque service doit avoir une description détaillée.

1.2 Problématique

UPnP permet la découverte et le contrôle automatiques des services disponibles sur le réseau par d'autres dispositifs sans intervention d'utilisateurs. Les dispositifs qui agissent en tant que serveurs peuvent annoncer leurs services aux clients. Les clients, connus sous le nom de points de contrôle, peuvent rechercher des services spécifiques sur le réseau. Quand ils trouvent les dispositifs avec les services désirés, les points de contrôle peuvent rechercher des descriptions détaillées des dispositifs et des services et, agir en conséquence. Ce processus de découverte et de contrôle est fonctionnel sur les réseaux fixes filaires et sans fil.

Les réseaux sans fil ad-hoc, ou Manet (*Mobile Ad-hoc Network*), ont une topologie qui n'a aucune infrastructure préexistante. Cette dernière se forme pendant l'apparition et le mouvement des nœuds. Par exemple, les participants d'une réunion, les intervenants des opérations de secours menées sur un site en ruine, les éléments engagés dans une opération militaire peuvent tirer profit des caractéristiques de tels réseaux pour échanger de l'information.

L'évolution rapide des performances des réseaux locaux sans fil et leur utilisation de plus en plus importante par les utilisateurs mobiles devraient bénéficier au développement des Manets. Si les Manets se différencient des réseaux classiques, par les caractéristiques de leurs topologies, les services demandés par les utilisateurs au réseau restent identiques.

UPnP est un protocole de découverte et de contrôle de services qui existe seulement sur des réseaux fixes et qui ne bénéficie pas des avantages des réseaux ad-hoc, comme la mobilité des nœuds. C'est-à-dire, la topologie du réseau peut changer rapidement, de façon aléatoire et non prévisible. Les nœuds dans un réseau ad-hoc peuvent être amenés à assurer des fonctions de routage.

UPnP est destiné aux équipements. Il permet de les rechercher, les contrôler et visualiser leur état. Notre objectif est d'avoir un protocole destiné pour la découverte et l'interaction avec les services qu'ils soient matériels ou logiciels.

Les nœuds constituant un réseau ad-hoc peuvent avoir des plates-formes et des capacités différentes, comme pour les téléphones mobiles, ordinateurs portables, PDAs,

imprimantes, etc. Le protocole de découverte de services doit tenir compte de cette variété de matériel en facilitant l'interopérabilité entre les différentes plates-formes. La faiblesse de la bande passante dans les réseaux ad-hoc nécessite une optimisation du trafic. Les ressources des terminaux (mémoire, processeur et batterie) dans un réseau ad-hoc sont limitées, ce qui implique une obligation de minimiser la consommation de ces ressources.

Donc, il faut prendre en considération les avantages et les contraintes des réseaux ad-hoc pendant la conception et la réalisation du protocole UPnP pour réseau ad-hoc.

1.3 Objectifs

Le but de notre recherche, dans le cadre de ce mémoire, est la conception d'un protocole de découverte de services dans les réseaux ad-hoc en rendant le protocole UPnP fonctionnel dans les réseaux ad-hoc.

Nous visons plusieurs objectifs :

1. Établir une stratégie pour rendre le protocole de découverte de services UPnP fonctionnel dans les réseaux ad-hoc.

2. Établir une stratégie pour intégrer le protocole de découverte de services SEDIRAN dans le protocole UPnP.

3. Proposer une architecture tout en sauvegardant les avantages du protocole UPnP et du protocole SEDIRAN.

1.4 Organisation du mémoire

Nous décrirons dans le chapitre II les différents protocoles de découverte de services dans les réseaux filaires dont le mécanisme de découverte a déjà été mis en place depuis quelques années. Puis, nous présenterons la problématique de la découverte de services dans les réseaux ad-hoc et quelques protocoles existants.

Dans le chapitre III, nous détaillerons le protocole de découverte de services dans les réseaux filaires UPnP. Ainsi, le protocole de découverte de services dans les réseaux ad-hoc

SEDIRAN réalisé à l'Université du Québec à Montréal, fera partie intégrante de notre solution.

Dans le chapitre IV, nous détaillerons notre propre solution en expliquant notre stratégie de découverte de services dans les réseaux ad-hoc avec le protocole UPnP qui fera l'objet d'une modification. Nous y présenterons les détails de notre proposition.

Le chapitre V présente la mise en œuvre de notre proposition.

Enfin, le dernier chapitre conclut le travail du mémoire.

CHAPITRE II

LES PROTOCOLES DE DÉCOUVERTE DE SERVICES

2.1 Introduction

La communauté de gestion de réseaux d'ordinateurs a senti le besoin de découvertes de services, il y a quelques années. Plusieurs compagnies, consortiums et un groupe de travail d'IETF (*Internet Engineering Task Force*) ont commencé à faire de la recherche dans ce domaine. Une variété de protocoles de découvertes de services est actuellement en cours de développement. Les plus connus jusqu'ici sont :

- protocole de localisation de services (SLP), développé par l'IETF [1] ;
- Jini, de Sun Microsystems, basé sur Java [2] ;
- Salutation [3] ;
- *Universal plug and play* de Microsoft (UPnP) [4] ;
- protocole de découverte de service de Bluetooth (SDP) [5] ;
- DNS-SD [7].

Plusieurs compagnies contribuent activement au développement de ces protocoles de découverte de services (voir Figure 2.1).

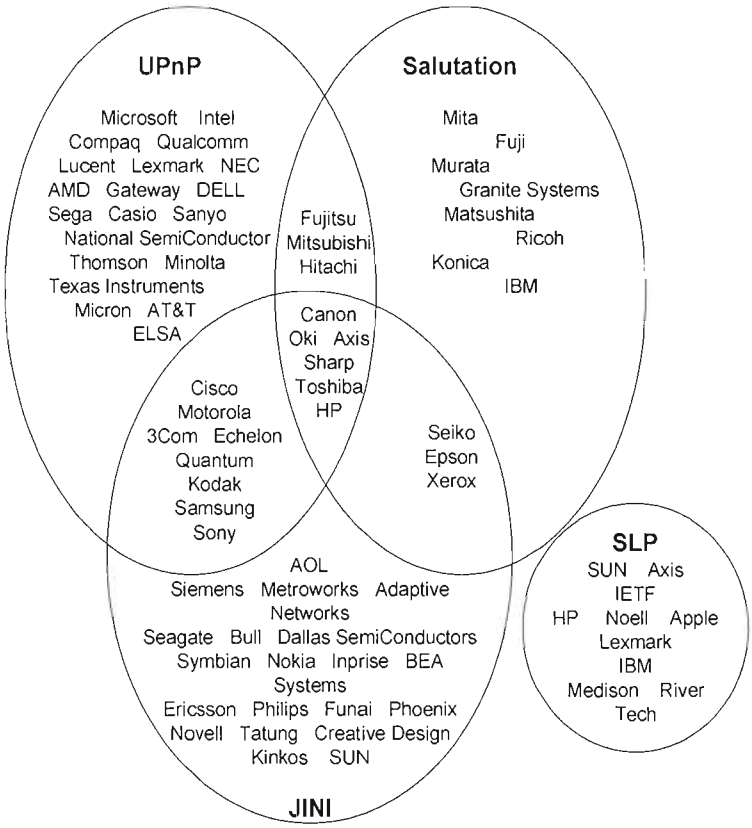

Figure 2.1 - Compagnies participant au développement de UPnP, Salutation, Jini et SLP [8]

2.2 Définition des protocoles de découverte de services dans les réseaux fixes

2.2.1 Service location protocol (SLP)

Le protocole de localisation de services SLP est développé par le groupe de travail IETF SvrLoc et est actuellement disponible dans la version 2. SLP vise à être une norme indépendante du fournisseur. Il est conçu pour des réseaux de TCP/IP et est extensible jusqu'à de grands réseaux d'entreprises.

L'architecture de SLP est constituée de trois composantes principales :

- les agents utilisateurs (UA) effectuent la découverte de services au nom du client (utilisateur ou application) ; [8]
- les agents de services (SA) annoncent la localisation et les caractéristiques des services, au nom des services ; [8]

- les agents de répertoires (DA) rassemblent des adresses et les informations de service reçues des agents de services (SAs) dans leur base de données et répondent aux demandes de services des agents utilisateurs (UAs). [8]

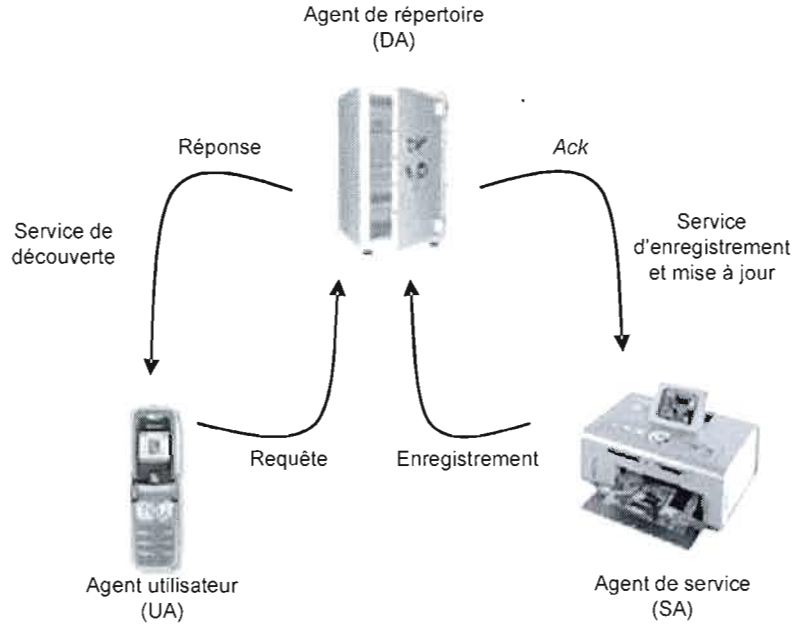

Figure 2.2 -Les agents du protocole SLP

Quand un nouveau service se connecte à un réseau, l'agent de service (SA) contacte l'agent de répertoire (DA) pour annoncer son existence (enregistrement de service).

Quand un utilisateur a besoin d'un certain service, l'agent utilisateur (UA) questionne l'agent de répertoire (DA) au sujet des services disponibles dans le réseau (demande de service). Après la réception de l'adresse et des caractéristiques du service désiré, l'utilisateur peut finalement utiliser le service (voir Figure 2.2).

Avant qu'un client (agent utilisateur (UA) ou agent de service (SA)) puisse entrer en contact avec l'agent de répertoire (DA), il doit découvrir son existence.

Il y a trois méthodes différentes pour découvrir l'agent de répertoire (DA) : découverte statique, découverte active et découverte passive.

La découverte statique : les agents utilisateurs (UAs) et les agents de services (SAs) de SLP obtiennent l'adresse de l'agent de répertoire (DA) par DHCP (protocole de configuration dynamique du serveur).

La découverte active : les agents utilisateurs (UAs) et les agents de services (SAs) envoient des demandes de services à l'adresse (239.255.255.253) du groupe de multicast de SLP. L'agent de répertoire (DA) écoutant sur ce port recevra éventuellement une demande de services et répondra directement (par l'intermédiaire de l'unicast) à l'agent demandeur.

La découverte passive : les agents de répertoires (DAs) envoient périodiquement des annonces multicast pour leurs services. Les agents utilisateurs (UAs) et les agents de services (SAs) apprennent l'adresse de l'agent de répertoire (DA) des annonces reçues et peuvent maintenant entrer en contact avec l'agent de répertoire (DA) individuellement, par l'intermédiaire d'unicast.

Il est important de noter que le DA n'est pas obligatoire. En fait, il est employé particulièrement dans de grands réseaux avec beaucoup de services, puisqu'il laisse classer des services par catégories dans différents groupes.

Dans de plus petits réseaux (par exemple, maison ou réseaux de voiture ou réseaux ad-hoc), il est plus efficace de déployer le SLP sans l'agent de répertoire (DA).

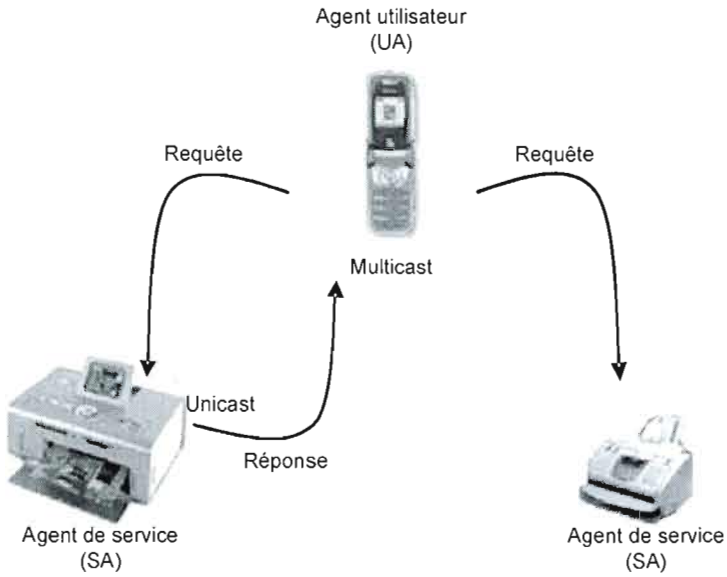

Figure 2.3 - Les agents du protocole SLP sans l'agent de répertoire (DA)

S'il n'existe pas de DA, tel qu'illustré à la figure 2.3, les agents utilisateurs (UAs) envoient leur demande de service à l'adresse de multicast de SLP à plusieurs reprises. Tous les agents de services (SAs) détectent, en écoutant sur le port, ces demandes de multicast et, s'ils possèdent le service demandé, ils enverront des réponses d'unicast aux agents utilisateurs (UAs).

Les agents de services (SAs) envoient périodiquement des annonces au port multicast de SLP pour annoncer les services. Par conséquent, les agents utilisateurs (UAs) peuvent se renseigner sur l'existence de nouveaux services.

Des services sont annoncés en utilisant un URL de service et un modèle de service (*template*). L'URL de service contient l'adresse IP du service, le numéro de port et le chemin. Les modèles de service (*templates*) indiquent les attributs qui caractérisent le service et ses valeurs par défaut.

2.2.2 JINI

L'architecture Jini est basée sur le modèle des clients recherchant des services. Ces services peuvent être des ressources matérielles, logicielles ou autres. Dans une fédération Jini, différentes instances d'objets représentent un service.

Il y a différentes approches pour implémenter la technologie Jini adaptée aux services. Chaque approche prend un chemin différent pour interagir avec le *lookup* service et fournit une interface écrite dans le langage de programmation Java aux clients de ce service.

Dans chaque cas, différents compromis sont faits entre la complexité du dispositif, la flexibilité du dispositif et la communication directe entre le client voulant utiliser le service et le dispositif qui met en application le service.

Le but de l'architecture Jini est de fédérer des groupes de dispositifs et de composants de logiciel dans un système réparti simple et dynamique.

Les services, le *Lookup*, le *Leasing* ou *bail* et les événements constituent les principaux composants de l'architecture Jini.

2.2.2.1 Les services

Les services sont des entités qui sont utilisées par des clients. Ces derniers peuvent être des personnes, des programmes ou d'autres services. Un service peut être matériel ou logiciel comme, par exemple, une imprimante, une caméra, un PDA, un système de stockage, un ordinateur, une imprimante réseau, etc.

Jini n'impose pas la méthode de communication entre un service et un utilisateur. Une interface du langage de programmation Java représentant un service est un ensemble de prototypes de méthodes, dont l'implémentation est laissée au programmeur du service [13].

2.2.2.2 *Lookup* service

Le « *lookup* » est le service de base de Jini. Il fonctionne comme une table associant les interfaces aux objets offrant ces interfaces.

Le « *lookup* » gère l'apparition et la disparition de services ainsi que la propagation de ces événements aux différents clients. Les différents « *lookups* » pouvant communiquer entre eux, il est aisé de créer un réseau de « *lookups* » et donc de propager les tables de correspondances. Par défaut, toutes les communications entre les services et le « *lookup* », ainsi qu'entre les clients et le « *lookup* », se font via le protocole RMI inclus avec le langage Java.

Il existe des solutions qui permettent de créer une interface entre un langage de programmation, comme le C/C++, et Jini. Ces solutions permettent l'utilisation de Jini sur des systèmes embarqués ne disposant pas de Java Virtual Machine.

2.2.2.3 *Leasing* (Bail)

Le mécanisme mis en œuvre pour détecter la disparition d'un service est le « *leasing* », qui est un bail entre le service et le *lookup*. Il a deux caractéristiques :

- **une durée de vie :** qui peut être limitée ou non dans le temps et qui définit la durée du service dans la table de *lookup* ;
- **une durée de renouvellement :** le service doit périodiquement renouveler son bail.

2.2.2.4 Les événements

Jini définit une série d'événements associés au fonctionnement du *lookup*. Ces événements sont assez simples : apparition ou disparition d'un service et modification des attributs d'un service. Un client peut demander au « *lookup* » de lui envoyer une partie ou tous ces événements. Cela lui permet de réagir à la situation du réseau.

2.2.2.5 Les composants du système Jini

Les composants du système Jini se partagent en trois catégories : l'infrastructure, le modèle de programmation et les services. (voir Tableau 2.1)

	Infrastructure	**Modèle de programmation**	**Services**
Base Java	JVM RMI Java Sécurité	Java APIs JavaBeans	JNDI *Enterprise Beans* JTS
Java + **Jini**	Découverte/Adhésion Sécurité distribuée *Lookup*	*Leasing* (bail) Transactions Événements	Impression Gestionnaire de transaction Service JavaSpaces

Tableau 2.1 - Segmentation de l'architecture de Jini [12]

2.2.2.6 L'enregistrement d'un service auprès du *lookup*

Un service doit s'enregistrer auprès d'un *lookup* pour se faire connaître dans le réseau. Donc, il doit soit :

- connaître l'adresse d'un *lookup* ;
- ou rechercher un *lookup* sur le réseau. Cette recherche se fait par l'envoie de paquets multicast sur le réseau. Ces *lookups* écoutent le réseau sur un port connu et répondent si nécessaire.

Pour s'enregistrer auprès du *lookup* découvert, le service envoi un ensemble d'attributs et un objet réalisant les interfaces désirées qui se retrouveront dans la table de correspondance du *lookup*. Un numéro d'identification unique affecté au service et un bail sont nouvellement créés.

2.2.2.7 La recherche d'un service par un client

Un client qui recherche un service, doit d'abord rechercher un *lookup*. La découverte de celui-ci se fait de la même façon que pour le fournisseur de service : soit en connaissant l'adresse d'un *lookup* ou en faisant une recherche par multicast.

Lorsqu'un client découvre un *lookup*, deux solutions s'offrent à lui :

- Utiliser directement le *lookup* et faire une recherche sur une interface et un ensemble d'attributs pour trouver le service voulu.
- S'enregistrer auprès du *lookup* pour recevoir une série d'événements comme l'ajout, le retrait ou la modification d'un service. On peut spécifier, pour chaque événement, le type d'interface à prendre en compte et, éventuellement, une série d'attributs qui serviront de filtres.

Dans le cas d'un événement, le client peut récupérer l'objet *ServiceItem* correspondant au service qui a déclenché cet événement avec son identificateur et ses attributs. Il existe alors deux *ServiceItems*, l'un correspondant au service avant l'événement, l'autre après l'événement. On peut ainsi savoir, par exemple, quel est le service qui a disparu, en récupérant son numéro d'identification.

Pour chaque service correspond un objet *ServiceItem* qui contient:

- Le numéro d'identification unique du service ;
- L'objet réalisant l'interface souhaitée ;
- L'ensemble des attributs associés à ce service.

2.2.2.8 La connexion d'un client et d'un service

Dès que le client récupère l'objet du service réalisant l'interface souhaitée, il se met à dialoguer avec le service et devient indépendant du *lookup*.

Il y a plusieurs types de services :

- Un service qui est réellement un objet et qui peut exécuter les méthodes de l'interface demandée chez le client. Par exemple, on peut crypter des documents. Le service donne alors une méthode de cryptage, mais l'exécution de celle-ci se fait chez le client.
- Un service qui sert de passerelle. L'objet, qui est alors stocké dans le *lookup*, sert à créer une connexion entre le client et le service. Les méthodes implémentées sont

alors appelées à distance, du client vers le service. L'objet transféré sera, par exemple, un *stub* RMI.

- Un service qui est un hybride des deux premiers types. Il s'agit d'un objet, dont une partie est exécutée par le client, et l'autre par le service.

2.2.3 SALUTATION

Salutation [3] est une autre approche pour la découverte de services. L'architecture de Salutation est développée par un consortium ouvert d'industries, appelé le consortium de Salutation [8].

L'architecture de Salutation est composée de *Salutation Managers* (SLMs). Les services enregistrent leurs caractéristiques avec un SLM, et les clients questionnent le SLM quand ils ont besoin d'un service. Après avoir découvert le service désiré, les clients peuvent demander au SLM l'utilisation du service.

2.2.4 *Universal plug and play* (UPnP)

Universel Plug and Play (UPnP) est développé par un consortium industriel qui a été fondé par Microsoft [4]. On peut indiquer qu'il étend la technologie *Plug and Play de* Microsoft au cas où les unités sont accessibles par une utilisation de TCP/IP [4].

Il est suggéré pour de petits réseaux d'ordinateurs personnels de bureau. Il permet des mécanismes de poste à poste (*peer to peer*) pour la configuration automatique des unités, la découverte de services et la commande des services. Dans la version en cours de UPnP (version 0.91), il n'y a aucun registre central de services, tels que le DA dans SLP ou la table de consultation dans Jini. Le protocole simple de découverte de services (*The Simple Service Discovery Protocol : SSDP*) est employé dans UPnP pour découvrir des services. SSDP emploie HTTP au-dessus de UDP et est ainsi conçu pour l'utilisation dans des réseaux IP.

2.2.5 Bluetooth service discovery protocol (SDP)

Bluetooth est une technologie de transmission sans fil à courte portée [5]. Le protocole Bluetooth contient le protocole de découverte de services (SDP) qui est employé pour localiser des services fournis ou disponibles par l'intermédiaire d'un dispositif de Bluetooth.

Il est basé sur la plate-forme Piano développée par Motorola et qui a été modifiée pour convenir à la nature dynamique des communications ad-hoc.

Il est consacré à la découverte de services spécifiquement pour cet environnement et se concentre ainsi pour découvrir des services. Il permet aux clients :

- de faire la recherche des services par types de services ;
- de faire la recherche des services par attributs de services ;
- de passer en revue tous les services sans connaître a priori leurs caractéristiques.

Le SDP n'inclut pas la fonctionnalité pour des services d'accès. Une fois que des services sont découverts avec le SDP, ils peuvent être choisis, consultés et employés par des mécanismes hors de la portée du SDP, par exemple, par d'autres protocoles de découverte de services tels que SLP et Salutation. Le SDP peut coexister avec d'autres protocoles de découvertes de services, mais il ne l'exige pas.

2.3 Comparaison entre SLP, Jini, Salutation et UPnP

Bien que ces protocoles emploient des architectures semblables, il y a plusieurs différences entre eux [8]. Le tableau 2.2 récapitule les caractéristiques principales des quatre protocoles de découvertes de services SLP, Jini, Salutation et UPnP.

UPnP : *Universal Plug and Play* est le plus jeune de ces protocoles. Il est toujours en cours de développement. Jusqu'ici, il n'existe aucune réalisation commerciale de UPnP mais, l'objectif de Microsoft est de le mettre en application pour toutes les plates-formes de Windows. Les spécifications et les codes sont disponibles gratuitement. UPnP est conçu seulement pour les réseaux TCP/IP.

Salutation : La découverte de services avec le protocole Salutation est définie sur une couche plus élevée, et la couche de transport n'est pas indiquée. Ainsi, le protocole Salutation est indépendant de la technologie de réseau et peut s'exécuter sur de multiples infrastructures finies, telles que TCP/IP fini et IrDA. Il n'est pas limité à HTTP via UDP via IP, comme UPnP. Le protocole Salutation est indépendant du langage de programmation. Son avantage principal vis-à-vis de UPnP et de Jini est qu'il existe déjà des réalisations commerciales.

Jini : Jini est également une approche plutôt nouvelle et aucun produit n'est encore sur le marché. Jini se distingue des autres approches principalement par le fait qu'il est basé sur Java.

D'une part, ce concept rend Jini indépendant de la plate-forme Jini et du système d'exploitation s'exécutant dessus. Jini emploie Java RMI pour déplacer le code de programme dans le réseau.

Un avantage principal, par rapport aux concepts de découverte de service qui ne sont pas basés sur Java, est qu'il présente la possibilité de déplacer des modules de gestion de dispositifs vers les applications client.

D'autre part, le fait que Jini est étroitement lié au langage de programmation Java le rend dépendant de l'environnement de programmation. Il exige également de ses unités d'exécuter la JVM, qui consomme la mémoire et la capacité de traitement. Ceci peut être une condition difficile à réaliser pour de grands modules de gestion de dispositifs et ne pourraient pas être satisfaits dans les systèmes embarqués.

En raison de la nature dynamique des réseaux ad-hoc, Jini utilise le concept du crédit-bail. Chaque fois qu'un dispositif joint le réseau et que ses services deviennent disponibles sur le réseau, il s'enregistre seulement pendant une certaine période, qui est appelée bail. C'est particulièrement utile pour des scénarios de réseaux ad-hoc très dynamiques.

SLP : ce protocole de localisation de services est standardisé et bien documenté par l'IETF. Il existe plusieurs réalisations de référence aussi bien que des produits commerciaux.

SLP offre une architecture flexible et extensible. L'utilisation des modèles (*templates*) de services rend la lecture de services rapide et l'interaction humaine possible.

Puisque SLP peut être utilisé avec ou sans l'agent de répertoire (DA), il convient aux réseaux de tailles différentes. Il s'étend de la très petite connectivité ad-hoc à de grands réseaux d'entreprises.

SLP inclut également un concept de crédit-bail avec un temps de vie limité pour savoir combien de temps l'agent de répertoire (DA) stockera un enregistrement de service.

SLP est indépendant du langage de programmation.

Protocole	SLP	Jini	Salutation	UPnP
Développeur	IETF	Sun Microsystems	Salutation Consortium	Microsoft
Licence	*Open source*	*Open licence*, utilisation commerciale payante	*Open source*	*Open source* seulement pour les membres
Version	2	1.0	2.1	0.91
Protocole de transport	TCP/IP	Indépendant	Indépendant	TCP/IP
Langage de programmation	Indépendant	Java	Indépendant	Indépendant
L'OS et la plate-forme	Dépendant	Indépendant	Dépendant	Dépendant
Transfert de code	Non	Oui (RMI)	Non	Non
Attributs des services	Oui	Oui	Oui	Non
Cache central pour dépôt	Oui (optionnel)	Utilisation optionnelle de SLP	Oui (optionnel)	Non
Opération w/o de répertoire	Oui	La table *Lookup* nécessaire	Oui	--
Concept du bail	Oui	Oui	Non	Oui
Sécurité	Dépendant de IP	Basé sur Java	authentification	Dépendant de IP

Tableau 2.2 - Comparaison entre les protocoles de découvertes de services

2.4 Définition des protocoles de découverte de services dans les réseaux ad-hoc

2.4.1 Protocole Konark

Ce protocole de découvertes de services a été réalisé spécialement pour les réseaux ad-hoc [16]. Avant, les protocoles de découvertes de services se concentraient sur les services qui étaient fournis par des dispositifs, tels que les imprimantes, les caméras, etc. Mais, avec l'apparition de plusieurs sortes de dispositifs manuels sans fil et mobiles, comme le PDA, et qui supportent le commerce électronique, il y a eu l'apparition de plusieurs autres sortes de services comme les jeux vidéo, la musique, les cartes géographiques, la météo, la vente des billets de cinémas, etc.

La description de services dans le protocole Konark est basée sur un langage de description fait en XML et qui est similaire au WSDL. Konark utilise les standards comme HTTP et SOAP. Il fournit un serveur micro-HTTP pour chaque dispositif. Les requêtes et les réponses sont basées sur SOAP.

Le protocole Konark est basé sur le modèle *peer-to-peer*, dont chaque nœud a la possibilité d'héberger ses services locaux et de les fournir à l'aide de son serveur local micro-HTTP.

Le protocole Konark permet l'annonce et la recherche de services d'une façon périodique, Il peut réagir à un événement et l'annonce peut être basée sur des informations géographiques ou temporelles.

Le réseau ad-hoc formé supporte l'envoi multicast des messages de découvertes. Les clients peuvent sauvegarder les descriptions des services annoncés dans une mémoire cache pour une utilisation ultérieure. À l'aide d'un registre de services organisé sous forme d'arborescence, les dispositifs peuvent stocker leurs services et maintenir l'information des services découverts ou annoncés. L'arbre a un nombre de niveaux représentant la classification des services. En se déplaçant de la racine vers le bas, les services deviennent de plus en plus spécifiques (voir Figure 2.4).

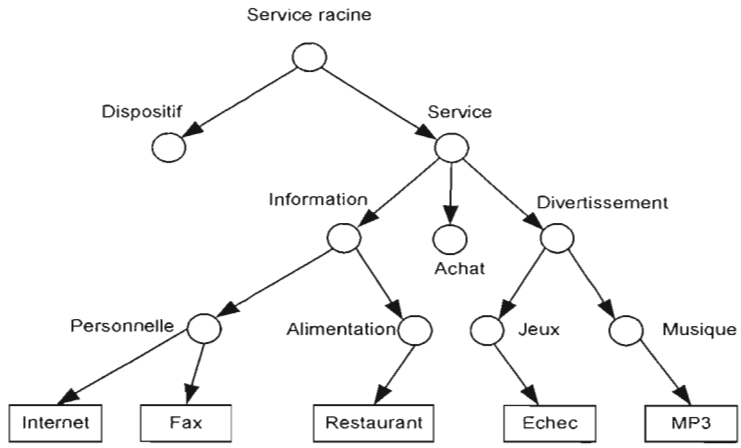

Figure 2.4 - Arbre des services

Konark permet à chaque dispositif d'agir en tant que serveur et client simultanément. Chaque dispositif inclut une application Konark, qui facilite l'interaction avec les usagers, afin d'initier et de contrôler l'annonce, la découverte et l'utilisation de services. En plus, il inclut un gestionnaire et un registre SDP qui maintiennent les objets des services et leurs descriptions. Un serveur micro-HTTP est présent pour manipuler des requêtes afin de livrer les services. Konark est fait de deux parties : découverte et livraison de services (voir Figures 2.5 et 2.6).

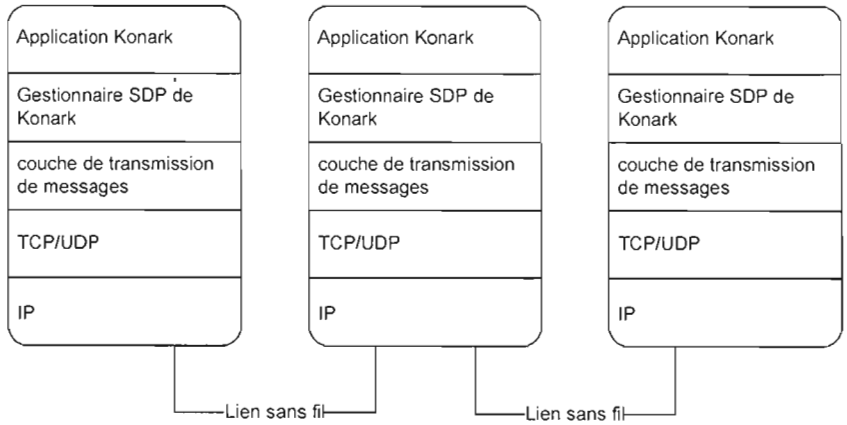

Figure 2.5 - Les couches de découvertes de services du protocole Konark

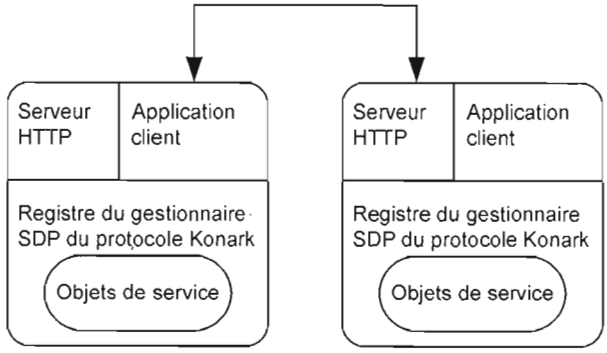

Figure 2.6 - Les composantes de la livraison du service

2.4.2 Protocole SPDP *(Secure Pervasive Discovery Protocol)*

Comme les objectifs de SPDP [19] sont de minimiser l'utilisation de l'énergie des dispositifs, par conséquent, la transmission des messages de découvertes doit être réduite au minimum. Avec ce protocole, les dispositifs annoncent leurs services seulement quand les autres dispositifs lancent des requêtes de recherches de ces services.

Le protocole SPDP permet le partage des services d'une façon sécuritaire, à travers un modèle de confiance entre les dispositifs, qui repose sur une Autorité de Certification établie à cette fin. Donc, les clients et les serveurs sont protégés contre les dispositifs douteux.

Le réseau ad-hoc est composé de D dispositifs et chaque dispositif offre S services. Ces dispositifs restent dans le réseau T secondes.

Chaque dispositif a un agent utilisateur SPDP (*SPDP_UA*) et un agent de service SPDP (*SPDP_SA*). L'agent utilisateur (*SPDP_UA*) est un processus fonctionnant au nom de l'utilisateur (Client) pour rechercher les services offerts dans le réseau par les pairs. Alors, l'agent de service (*SPDP_SA*) est un processus fonctionnant au nom du dispositif (serveur) pour annoncer ses services avec un temps T de disponibilité dans le réseau.

À chaque dispositif est associé un cache contenant une liste des services qu'il a écoutés dans le réseau (dont il a reçu les annonces). Chaque élément du cache associé au SPDP_UA a trois champs : la description du service, la durée de vie du service et le temps d'expiration du service.

Chaque dispositif possède une liste locale de dispositifs fiables ayant chacun un degré de confiance, on retrouve les degrés de confiance dans la liste locale et dans le cache. Dépendamment du degré de confiance, le dispositif choisit de mettre le service offert par le serveur dans son cache ou non.

Quand le client veut utiliser un service, il choisit celui appartenant au dispositif ayant le degré de confiance le plus élevé.

Quand une application ou l'utilisateur final d'un dispositif veut un service, la requête est envoyée par SPDP_UA.

Si la recherche concerne un service particulier, alors le SPDP_UA le cherche dans la liste locale de services et dans le cache. S'il est trouvé, il offre la description du service à l'application, sinon le SPDP_UA diffuse (*broadcast*) la requête du service dans le réseau. S'il n y a pas de réponse après un temps déterminé, le SPDP_UA répond à l'application que le service n'est pas disponible dans le réseau. S'il y a une réponse, le SPDP_UA met à jour son cache et envoie la description du service trouvé à l'application.

Le protocole permet de faire une recherche parmi tous les services existants dans le réseau. SPDP_UA envoie une requête recherchant un service de type *ALL*. S'il n'y a pas de réponse après un temps prédéfini, le SPDP_UA efface tous les services stockés dans son cache et répond à l'application en listant seulement les services locaux. Mais s'il y a une réponse, le SPDP_UA met à jour le cache et répond à l'application en listant les services du cache et les services locaux.

Les SPDP_UAs de tous les dispositifs du réseau continuent à faire leur recherche de tous les types de services pour mettre à jour leur cache. Si leur cache se remplit, ils effacent les services qui appartiennent à des dispositifs qui ont un degré de confiance inférieur ou un temps d'expiration restant court.

Le SPDP_SA annonce les services offerts par le dispositif. Il doit traiter les messages de recherche pour générer les réponses correspondantes.

Quand SPDP_SA reçoit une requête de recherche d'un service spécifique S, alors il doit vérifier si le service fait partie de ses services locaux ou s'il est dans le cache. Si le service est dans l'un ou l'autre, le SPDP_SA génère un temps aléatoire t inversement proportionnel au temps T de disponibilité du dispositif qui a initié la requête de recherche (t = random ($1/T$)). Pendant ce temps t, le SPDP_SA écoute sur le réseau les réponses des autres dispositifs à la même requête de recherche pour mettre à jour son cache. Quand le temps t expire, le SPDP_SA répond avec le service S s'il sait qu'il n'a pas été annoncé.

Quand SPDP_SA reçoit une requête de recherche d'un service de type *ALL*, alors le SPDP_SA génère un temps aléatoire t inversement proportionnel au temps T de disponibilité du dispositif qui a initié la requête de recherche (t = random ($1/T$)). Pendant ce temps t, le SPDP_SA écoute sur le réseau les réponses des autres dispositifs à la même requête de recherche pour mettre à jour le cache. Quand le temps t expire, le SPDP_SA répond avec toute sa liste de services locaux et ceux du cache, s'il sait qu'aucun dispositif n'a donné de réponse pour de nouveaux services.

2.4.3 Protocole GSD (*Group-based Service Discovery*)

Le protocole GSD est un protocole de découvertes de services dans les réseaux Manet [17][18]. Il est basé sur les concepts de mémoire cache *peer-to-peer* des annonces de services et des groupes à qui les services appartiennent. Les services sont décrits par le langage de l'ontologie du web OWL (*Ontology Web Language*) [25]. Le protocole exploite la hiérarchie classe/sous-classe de OWL pour décrire les groupes de services et utilise ces informations sémantiques pour sélectionner les requêtes de services à renvoyer (voir Figure 2.7).

L'OWL, qui est basé sur XML et la description de ressources *framework* [20], est également employé comme standard pour décrire l'information ou le service sur l'infrastructure filaire et le Web. Ceci rend la description de services interopérable avec d'autres infrastructures sémantiques du Web.

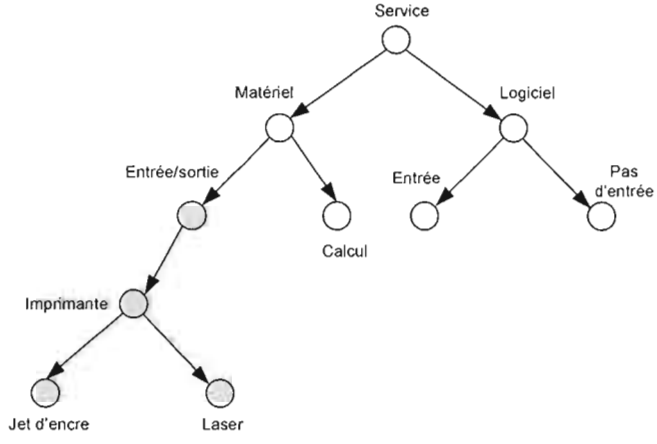

Figure 2.7 - La hiérarchie des groupes de services

Chaque fournisseur de services annonce périodiquement une liste de ses services à tous les nœuds du réseau Manet. Ce message est sous la forme suivante :

<Packet-type, Source-Address, Service-Description, Service-Groups, Other-Groups, Hop-Count, Lifetime, ADV DIAMETER>

- ***Service-Description*** : contient la description des services locaux du nœud émetteur.
- ***Service-Groups*** : contient les noms des groupes qui correspondent aux services locaux.
- ***Other-Groups*** : contient la liste des groupes des services non locaux, qui est construite à partir de la mémoire cache.
- ***ADV DIAMETER*** : détermine le nombre de sauts à effectuer quand un nœud reçoit un message d'annonce et qu'il doit le retransmettre.
- ***Hop-Count*** : détermine le nombre de sauts déjà effectués.
- ***Lifetime*** : correspond à la durée de vie d'une annonce dans le cache du nœud récepteur.

Le nœud qui reçoit un message d'annonce enregistre le service dans sa mémoire cache. Chaque entrée contient les champs suivants :

<Source-Address, Local, Service-Description, Service-Groups, Other-Groups, Lifetime>

- ***Local*** **:** la mémoire cache stocke aussi la description des services locaux identifiés par le champ Local, qui est de type booléen.
- ***Other-Groups*** **:** contient une liste des groupes que le nœud correspondant au « *Source-Address* » a pu voir dans son voisinage.
- ***Lifetime*** : la durée de vie d'une entrée dans le cache est déterminée par le *Liftime* correspondant au message d'annonce.

Une requête de découverte de services provient d'une source de requêtes (RS) dont la couche application demande le service. Une demande comprend une description basée sur l'ontologie du service demandé et inclut optionnellement les groupes de descriptions

de services auxquels le service demandé appartient. Avant sa diffusion dans le réseau, la requête est tout d'abord assortie aux services du cache local. Cette requête de service est sous la forme suivante :

<Packet-type, BroadcastId, Service-Description, Request-Groups, Source-Address, Last-Address, Hop-Count>

- ***Request-Groups*** : contiennent le(s) groupe(s) de services à qui le service demandé appartient.
- ***Hop-Count*** : ce paramètre, contrôlé par l'utilisateur, détermine la limite maximale de la durée de propagation de la requête.

Le champ *Other-Groups est* dans le cache de service de chaque nœud et permet aux nœuds qui traitent la requête de découverte de services de la retransmettre si elle est absence du cache local.

La réponse à une requête de découverte de services *Service Reply* est générée par chaque nœud qui a trouvé des entrées dans le cache assorties à la requête de découverte.

Un mécanisme de routage inverse *Reverse Routing* est utilisé pour envoyer le message de réponse. À la réception d'une requête, le nœud met à jour la table de routage avec un temps d'expiration pour permettre le nettoyage des routes expirées en se basant sur les champs « *Source-Address* », « *BroadcastId* » et « *Previous-Address* » de la table de routage inverse et dont voici la forme :

<Source-Address, BroadcastId, Previous-Address>

Dans le prochain chapitre nous détaillerons le protocole UPnP de découverte de services dans les réseaux filaires et sans fil avec points fixes, ainsi que protocole SEDIRAN de découverte de services dans les réseaux ad-hoc.

CHAPITRE III

UNIVERSAL PLUG AND PLAY ET LE SEDIRAN

Universal plug and play (UPnP)

Universal Plug and Play utilise le standard TCP/IP et les protocoles de l'Internet. UPnP est distribué, c'est une architecture ouverte et défini par un ensemble de protocoles. Il est indépendant des systèmes d'exploitation et des langages de programmation. UPnP ne spécifie pas les APIs, mais laisse le soin aux vendeurs de systèmes d'exploitation de créer les APIs répondant aux besoins.

Universal Plug and Play Forum qui se nommait à l'origine Device Control Protocols (DCPs), a défini les descriptions des services et des dispositifs UPnP. C'est un groupe de compagnies et d'individus qui ont joué un rôle important pour définir les spécifications des dispositifs et services UPnP.

Mis en place le 18 octobre 1999, ce forum regroupe plus de 200 vendeurs qui sont des leaders dans l'industrie de l'électronique, l'informatique, l'automatisation et la sécurité maison, les réseaux et les dispositifs mobiles.

3.1 Composantes de UPnP

UPnP est construit principalement de dispositifs (*devices*), de services et de points de contrôle. (voir Figure 3.1)

3.1.1 Dispositifs (*Devices*)

Le dispositif UPnP est un conteneur de services et d'autres dispositifs imbriqués. Par exemple, le VCR est un dispositif qui peut contenir un service de transport de bande, un service de *tuner* et un service d'alarme.

Différentes catégories de dispositifs UPnP sont associées aux différents ensembles de services et de dispositifs imbriqués. Par exemple, les services d'un VCR sont différents de ceux d'une imprimante.

3.1.2 Services

Dans un réseau UPnP, la plus petite unité de contrôle est le service. Ce dernier garde son état d'actions et de modèles dans des variables d'états. Comme le service horloge qui a une variable d'état appelée « current_time » et deux actions « set_time » et « get_time », cette information est une partie de la description du service XML standardisé par le forum UPnP. Et l'URL qui pointe vers ce fichier XML de description de services se trouve dans le fichier XML de description du dispositif.

Un service dans un dispositif UPnP a une table d'état, un serveur de contrôle et un serveur d'événements.

La table d'état : modélise l'état du service via les variables d'états et les met à jour quand l'état change.

Le serveur de contrôle : reçoit des requêtes comme « set_time », les exécute, met à jour la table d'état et retourne les réponses.

Le serveur d'événements : publie les événements aux clients intéressés lorsque l'état de service change. Par exemple, le service du système d'incendie envoie un événement aux abonnés intéressés quand son état change pour « sonner ».

3.1.3 Points de contrôle

Le point de contrôle dans UPnP est capable de découvrir et de contrôler d'autres dispositifs. Après la découverte d'un dispositif, un point de contrôle pourrait :

- Rechercher la description du dispositif et obtenir la liste des services associés.
- Rechercher les descriptions des services qui sont intéressants.
- Invoquer les actions pour contrôler le service.
- S'inscrire à la source de l'événement du service. Lorsque l'état du service change, le serveur d'événements envoie un événement au point de contrôle.

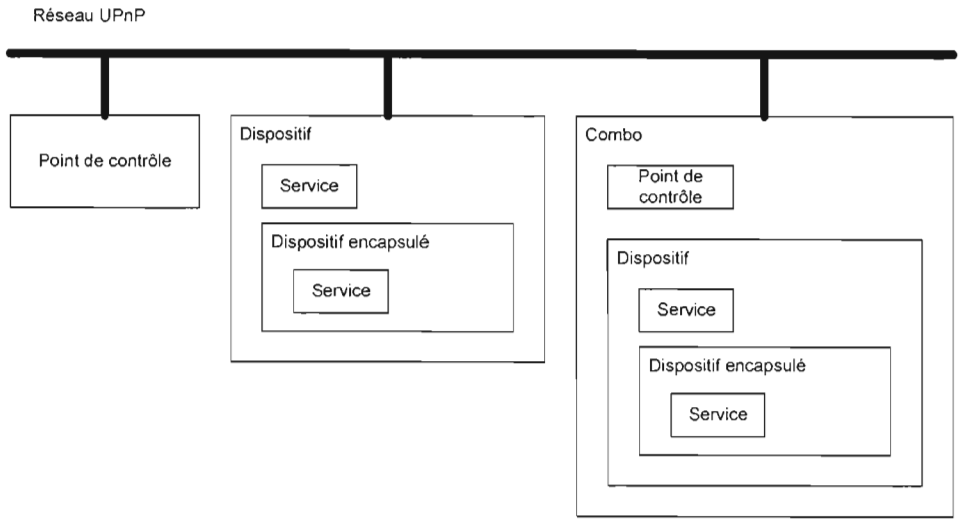

Figure 3.1 - Les composantes UPnP [21]

3.2 Vue générale sur le protocole UPnP

Les dispositifs dans le réseau UPnP peuvent se connecter en utilisant n'importe quelle communication de média incluant des radios fréquences, des lignes téléphoniques, des lignes d'alimentation, IRDA, Ethernet et IEEE 1394. Des protocoles standards sont utilisés dans UPnP comme TCP/IP, HTTP et XML. (voir Figure 3.2)

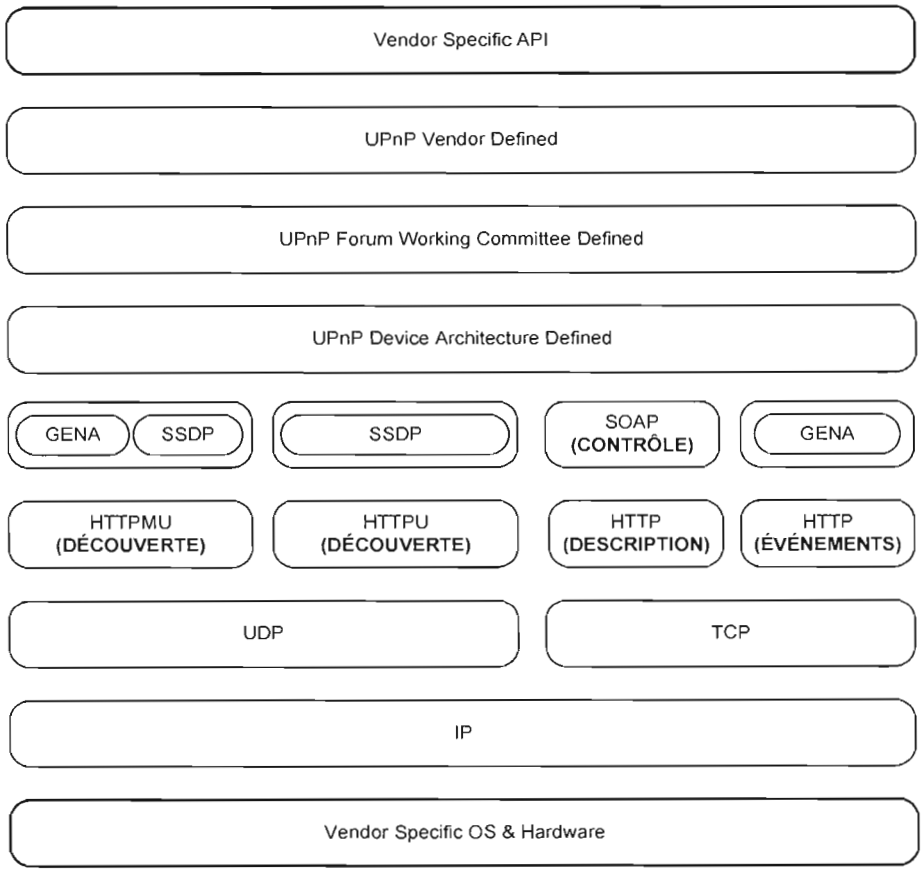

Figure 3.2 - Les couches du protocole UPnP. [14]

Il y a trois protocoles spécifiques au UPnP et qui définissent les couches de protocoles les plus élevées, UPnP *vendor defined*, UPnP *Forum Working Committee Defined et* UPnP *Device Architecture Defined.*

3.2.1 UPnP vendor defined

Il définit les données spécifiques aux dispositifs ou services telles que le nom du dispositif, le numéro du modèle, le nom du manufacturier et l'URL vers la description du service, etc. Ces données sont encapsulées dans les protocoles spécifiques de UPnP définis dans le document *UPnP Device Architecture*, tel que le modèle XML de description.

3.2.2 UPnP *Forum Working Committee Defined*

Basé sur *UPnP Device Architecture*, il définit de l'information globale des types de dispositifs comme les VCRs, les systèmes HVAC (chauffage, ventilation et climatisation), les lave-vaisselle, etc. Il crée un modèle pour chaque type de dispositif ou service.

3.2.3 UPnP *Device Architecture Defined*

Il définit le schéma pour créer les descriptions des dispositifs et des services pour n'importe quel type de dispositif ou de service.

Puis, nous poursuivons en parlant des autres couches du protocole UPnP.

3.2.4 TCP/IP

La pile de protocoles TCP/IP sert de base pour le reste des protocoles UPnP. Les dispositifs UPnP peuvent utiliser plusieurs protocoles dans les couches TCP/IP incluant TCP, UDP, IGMP, ARP et IP.

3.2.5 HTTP, HTTPU, HTTPMU

HTTP est responsable du succès de l'Internet et est au cœur de UPnP. Tous les aspects de UPnP sont construits sur la couche HTTP ou une de ses variantes. HTTPU et HTTPMU sont des variantes de HTTP définies pour livrer des messages au-dessus d'UDP/IP à la place de TCP/IP. HTTPMU (HTTP multicast UDP) pour les messages « Notify » et « Discover » et HTTPU (HTTP unicast UDP) pour les réponses. Ces protocoles sont utilisés par SSDP. La base des formats des messages utilisés par ces protocoles s'adapte avec celui de HTTP.

3.2.6 SSDP

Simple Service Discovery Protocol décrit comment les services sont découverts dans le réseau. Il est construit sur HTTPU et HTTPMU et définit des méthodes pour que le point de contrôle découvre les ressources, et aussi pour que les dispositifs annoncent leur disponibilité dans le réseau.

Le point de contrôle UPnP peut envoyer une requête de recherche en multicast via HTTPMU pour découvrir les dispositifs et les services disponibles dans le réseau.

D'une façon similaire, un dispositif qui vient de se brancher au réseau envoie des annonces SSDP en multicast pour avertir de sa présence et de ses services.

Les messages d'annonces, de présences et de réponses unicast des dispositifs contiennent un pointeur au document de description des dispositifs contenant les propriétés et les services supportés par le dispositif.

3.2.7 GENA (*General Event Notification Architecture*)

C'est une extension de HTTP qui fournit la capacité d'envoyer et de recevoir des annonces de changements d'états de services. C'est un protocole utilisé pour la découverte de services qui comporte trois sortes de messages :

- *SUBSCRIBE*: pour s'abonner au service ;
- *UNSUBSCRIBE* : pour se désabonner du service ;
- *NOTIFY* : pour la disponibilité de services et dispositifs et pour le changement d'état des variables.

3.2.8 SOAP (*Simple Object Access Protocol*)

SOAP définit l'utilisation de XML et HTTP pour exécuter les RPCs (*Remote Procedure Calls*). Il est devenu le standard pour la communication basée sur RPC via internet. En utilisant l'infrastructure existante d'Internet, il fonctionne avec les pare-feu et les serveurs mandataires (*proxies*).

UPnP utilise SOAP pour livrer les messages de contrôle aux dispositifs et retourner les résultats ou erreurs aux points de contrôle.

Chaque requête de contrôle est un message SOAP qui contient l'action à invoquer avec un ensemble de paramètres. La réponse est également un message SOAP contenant l'état, la valeur de retour ainsi que les paramètres de retour.

3.3 Différentes phases de fonctionnement du protocole UPnP

L'adressage IP est la base du protocole UPnP. Chaque dispositif reçoit, à sa première connexion, une adresse IP d'un serveur DHCP. Si le réseau n'est pas doté d'un serveur DHCP, alors le dispositif choisit automatiquement une adresse IP par le biais de « *Auto IP* ».

Le dispositif UPnP passe par la suite en phase de découverte, c'est-à-dire qu'il doit informer les autres dispositifs des services offerts et doit également pouvoir rechercher les dispositifs du réseau. Cette phase d'annonce/recherche se base sur le protocole SSDP.

La phase de découverte est suivie d'une phase de description qui permet à un dispositif de connaître les caractéristiques des autres dispositifs telles que le type de service offert, le nom du fabricant, le modèle, le numéro de série, etc.

Vient, par la suite, la phase de contrôle. Le dispositif utilise les fonctions déjà connues pendant la phase de description, pour ses interactions qui se font via des messages XML à l'aide de SOAP.

La phase événement permet d'adapter les dispositifs UPnP en analysant au fur à mesure les messages d'états et d'actions des autres dispositifs.

Les points de contrôle peuvent demander à un fournisseur de services de leur envoyer des annonces en cas d'activités ou de changements d'états.

La présentation est la phase qui termine ce processus de fonctionnement UPnP. Elle permet à un utilisateur de se connecter à un dispositif via un navigateur, en utilisant une URL, afin de consulter et contrôler l'état du dispositif.

3.3.1 Phase adressage

Le réseau UPnP est fondé sur le protocole TCP/IP et la clé de ce protocole est l'adressage. Chaque dispositif doit avoir un client DHCP. Quand il se connecte pour la première fois au réseau, il doit faire une recherche du serveur DHCP. S'il est disponible, il assignera une adresse IP au dispositif qui vient de se connecter au réseau, sinon le dispositif utilisera Auto-IP pour acquérir une adresse.

Une fois l'adresse IP obtenue, un nom DNS lui sera assigné. C'est avantageux car même si un dispositif change d'adresse IP, il peut toujours garder son nom DNS, ce qui facilite la tâche aux utilisateurs.

3.3.2 Phase découverte

3.3.2.1 Message d'annonce

Quand un dispositif est ajouté au réseau, il annonce ses services aux points de contrôle. Ces annonces se font sous la forme d'envois de messages d'annonce en mode multicast à l'adresse standard 239.255.255.250 et au numéro de port 1900. Les points de contrôle écoutent ce port afin de détecter les dispositifs et les services disponibles (voir Figure 3.3).

Si le dispositif incorpore d'autres dispositifs et plusieurs services, il doit envoyer pour chacun d'eux un message de d'annonce. Également, chaque message doit contenir l'information spécifique au dispositif ou au service incorporé auquel il est lié.

Les messages doivent inclure la durée d'expiration des annonces. Si le dispositif reste disponible, alors les annonces doivent être renvoyées avec une nouvelle durée d'expiration. Mais, si le dispositif devient indisponible, alors il doit explicitement annuler ses annonces s'il est capable. Sinon, après l'écoulement de la durée d'expiration, les annonces s'annulent.

Un message d'avertissement contient quatre composantes majeures :

- L'en-tête NT (*Notification Type*) spécifie le type d'un dispositif ou d'un service.
- L'en-tête USN (*Unique Service Name*) contient l'identificateur d'un dispositif ou d'un service.
- L'en-tête LOCATION et un URL vers la description du dispositif.
- L'en-tête CACHE-CONTROL pour déterminer la durée de vie de l'annonce.

Quand un dispositif veut quitter le réseau, il envoie un message ssdp:byebye en multicast pour chaque annonce ssdp:alive déjà multicastée et encore non expirée.

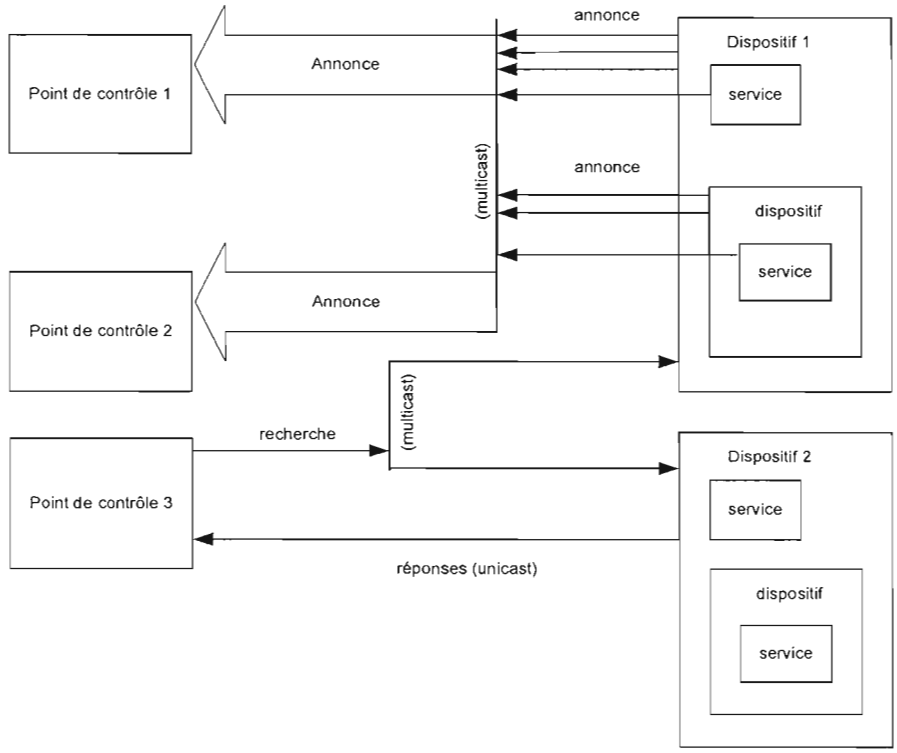

Figure 3.3 - Recherche et annonce de dispositifs et services [15]

Dispositif disponible

Dans UPnP, un dispositif (*Device*) qui vient de joindre un réseau annonce sa présence (c'est-à-dire la présence du dispositif racine, des dispositifs embarqués et des services, voir la sous-section 3.1.1) en envoyant des messages HTTP à l'adresse multicast « Host : 239.255.255.250 :1900 », l'annonce a une durée maximale de « CACHE-CONTROL », le type du service recherché est « NT », avec un identificateur unique « USN » en utilisant le protocole SSDP avec l'en-tête « NTS », la description du service se trouve à l'URL désignée dans l'en-tête « LOCATION » et « SERVER » qui détermine la version du protocole UPnP, le nom et la version du service afin d'éviter les incompatibilités (voir Figure 3.4).

NOTIFY * HTTP/1.1

HOST: 239.255.255.250:1900

CACHE-CONTROL: max-age = *secondes jusqu'à l'expiration de l'annonce*

LOCATION: *URL de la description UPnP pour un dispositif racine*
NT: *cible recherchée*
NTS: ssdp:alive
SERVER: *OS/version* **UPnP/1.0** *produit/version*
USN: *UUID annoncé*

Figure 3.4 - Format du message d'annonce de disponibilité

Le nombre de messages envoyés dépend du nombre de dispositifs embarqués dans le dispositif racine et du nombre de services contenus dans ces dispositifs. Le dispositif racine envoie trois requêtes, le dispositif embarqué en envoie deux, et le service en envoie un. Un dispositif qui a *e* dispositifs embarqués et *s* services enverra donc $3+2e+s$ messages.

Dispositif non disponible

Dans UPnP, un dispositif (*Device*) qui quitte un réseau annonce qu'il doit quitter en envoyant un message HTTP à l'adresse multicast « Host : 239.255.255.250 :1900 » pour chaque message d'annonce de disponibilité envoyé, le type du service recherché est « NT », avec un identificateur unique « USN » en utilisant le protocole SSDP avec l'en-tête « NTS », (voir Figure 3.5).

NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
NT: *cible recherchée*
NTS: ssdp:byebye
USN: *UUID annoncé*

Figure 3.5 - Format du message d'annonce de non disponibilité

Si le dispositif quitte le réseau brusquement sans annoncer sa sortie, alors le message d'annonce de disponibilité restera dans les caches des points de contrôle, jusqu'à son expiration telle que spécifie l'en-tête CACHE-CONTROL.

NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
CACHE-CONTROL: max-age = 5000
LOCATION: http://coffemaker1.mykitchen.de/description
NT: ka:coffe-maker
NTS: ssdp:alive
SERVER: KOS/0.1 **UPnP/1.0** CoffeMaker/6.0
USN: uuid:coffem123-01jan-2001-0099-12345678

Figure 3.6 - Exemple d'annonce de disponibilité d'une cafetière dans UPnP

La figure 3.6 présente un exemple d'annonce qui informe de la disponibilité d'une cafetière. Envoyé à l'adresse multicast « Host : 239.255.255.250 :1900 », l'annonce a une durée de 5000 secondes. Le type du service est « ka:coffe-maker » avec un identificateur unique « uuid:coffem123-01jan-2001-0099-12345678 ». La description du service se trouve à l'URL http://coffemaker1.mykitchen.de/description.

3.3.2.2 Message de recherche

Quand un point de contrôle est ajouté au réseau, le protocole de découverte UPnP lui permet de faire la recherche de dispositifs spécifiques dans le réseau. Il utilise le mode multicast à l'adresse et au port (239.255.255.250 :1900). Le message de recherche contient un **modèle**, ou une **cible, égale à** un type ou identificateur d'un dispositif ou service.

Les réponses des dispositifs contiennent des messages de découvertes essentiellement identiques à ceux annoncés par des dispositifs récemment ajoutés au réseau. Les réponses se font en mode unicast (voir Figure 3.3).

Dans UPnP, un point de contrôle (client) qui vient de joindre un réseau recherche la disponibilité de services en envoyant un message HTTP multicast à l'adresse « HOST : 239.255.255.250:1900 ». Le message comporte d'autres champs: « MX » indique que le dispositif doit attendre un temps aléatoire entre 0 et MX secondes pour envoyer la réponse; « MAN : SSDP :discover » spécifie le protocole utilisé et « ST » indique le type de service recherché (voir Figure 3.7).

M-SEARCH * HTTP/1.1

HOST: 239.255.255.250:1900

MAN: "ssdp:discover"

MX: *secondes pour retarder la réponse*

ST: *cible recherché*

Figure 3.7 - Format du message de recherche

M-SEARCH * HTTP/1.1

HOST: 239.255.255.250:1900

MAN: "ssdp:discover"

MX: 3

ST: ka:coffe-maker

Figure 3.8 - Exemple de recherche d'une cafetière dans UPnP

La figure 3.8 représente un message de recherche à l'adresse multicast « Host : 239.255.255.250 :1900 » d'un service cafetière, spécifiant le temps maximum 3 pour retarder la réponse, le type du service est « ka :coffe-maker » et utilisant le protocole « SSDP :discover » pour la recherche de service.

3.3.2.3 Message de réponse

Si l'en-tête (ST) du message de recherche « ssdp:all » ou « upnp:root device » ou « uuid:UUID » concorde exactement avec le dispositif ou le préfixe du dispositif ou le type du service supporté par le dispositif que le point de contrôle veut rechercher, alors le dispositif répond avec un message unicast au point de contrôle.

Le dispositif doit attendre un temps aléatoire entre 0 et MX secondes pour envoyer la réponse. MX a déjà été envoyé par le point de contrôle dans l'en-tête de son message.

Le MX ne doit pas dépasser 120 secondes. Autrement, 120 secondes seront considérées. Si MX n'existe pas dans le message de recherche, alors le dispositif ou le service concerné ignore ce message.

Le message de réponse unicast à une recherche de disponibilité d'un service, le type du service recherchée se trouve dans « ST », la durée de disponibilité du service est dans « CACHE-CONTROL », avec un identificateur unique qu'on trouve dans « USN », la description du service se trouve à l'URL dans « LOCATION », « SERVER » détermine la version du protocole UPnP, le nom et la version du service, la date de la génération de la réponse est dans l'en-tête « DATE » (voir Figure 3.9).

HTTP/1.1 200 OK
CACHE-CONTROL: max-age = *secondes jusqu'à l'expiration de l'annonce*
DATE: date de la génération de la réponse
EXT:
LOCATION: *URL de la description UPnP pour un dispositif racine*
SERVER: *OS/version* UPnP/1.0 *produit/version*
ST: *cible recherchée*
USN: *UUID annoncé*

Figure 3.9 - Format du message de réponse

HTTP/1.1 200 OK
CACHE-CONTROL: max-age = 5000
EXT:
LOCATION: http://coffemaker1.mykitchen.de/description
SERVER: KOS/0.1 **UPnP/1.0** CoffeMaker/6.0
ST: ka:coffe-maker
USN: uuid:coffem123-01jan-2001-0099-12345678

Figure 3.10 - Exemple de réponse à une recherche de disponibilité d'une cafetière

La figure 3.10 représente un message de réponse unicast à une recherche de disponibilité d'un service cafetière dans UPnP, le type du service est « ka :coffe-maker », la durée de disponibilité du service est 5000 secondes, avec un identificateur unique « uuid:coffem123-01jan-2001-0099-12345678 », la description du service se trouve à l'URL http://coffemaker1.mykitchen.de/description.

3.3.3 Phase description

Après que le point de contrôle ait découvert un dispositif, les informations qu'il a à son sujet sont limitées à l'essentiel pour la découverte, comme le type du dispositif (ou du service) UPnP, identificateur universel unique du dispositif, et un lien URL à la description du dispositif UPnP. Ce dernier permet au point de contrôle d'en apprendre davantage sur le dispositif et sur ses capacités, ou comment interagir avec lui (voir Figure 3.11).

La description du dispositif ou du service UPnP est écrite par *UPnP Vendor* avec le langage XML et est basée sur un modèle (*template*) du dispositif UPnP standard. Ce modèle est produit par *UPnP Forum Working Committee.* Ils ont tiré ce modèle de *UPnP Template Language*.

La description du service UPnP inclut une liste d'actions auxquelles le service répond et leurs paramètres. La description du service inclut aussi une liste de variables. Les variables modélisent l'état du service au temps d'exécution.

Si un dispositif veut changer une de ses descriptions, il doit annuler ses annonces et ré-annoncer une autre fois. Le point de contrôle suppose que les descriptions du dispositif et de ses services ont changé.

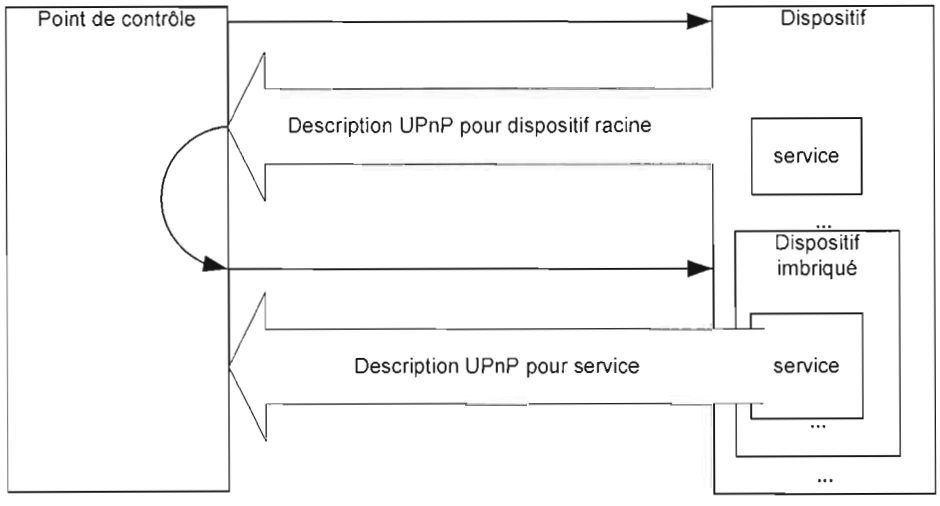

Figure 3.11 - Description UPnP pour dispositifs et services

3.3.4 Phase contrôle

Une fois que le point de contrôle a récupéré la description d'un service, il peut invoquer les actions offertes en envoyant un message de contrôle vers l'URL de contrôle du service. Le service retourne le résultat ou une erreur en cas d'échec. L'invocation peut causer un changement de variables d'état, et dans ce cas le service envoie des notifications aux points de contrôle intéressés. Pour invoquer une action, un message SOAP est crée et livré par HTTP via TCP/IP.

Le contrôle est l'étape 3 dans le réseau UPnP. Le point de contrôle invoque des actions des dispositifs qui sont une sorte de RPC. Le point de contrôle peut questionner les services en invoquant les actions et recevoir des réponses indiquant le résultat de l'action ou une erreur (voir Figure 3.12).

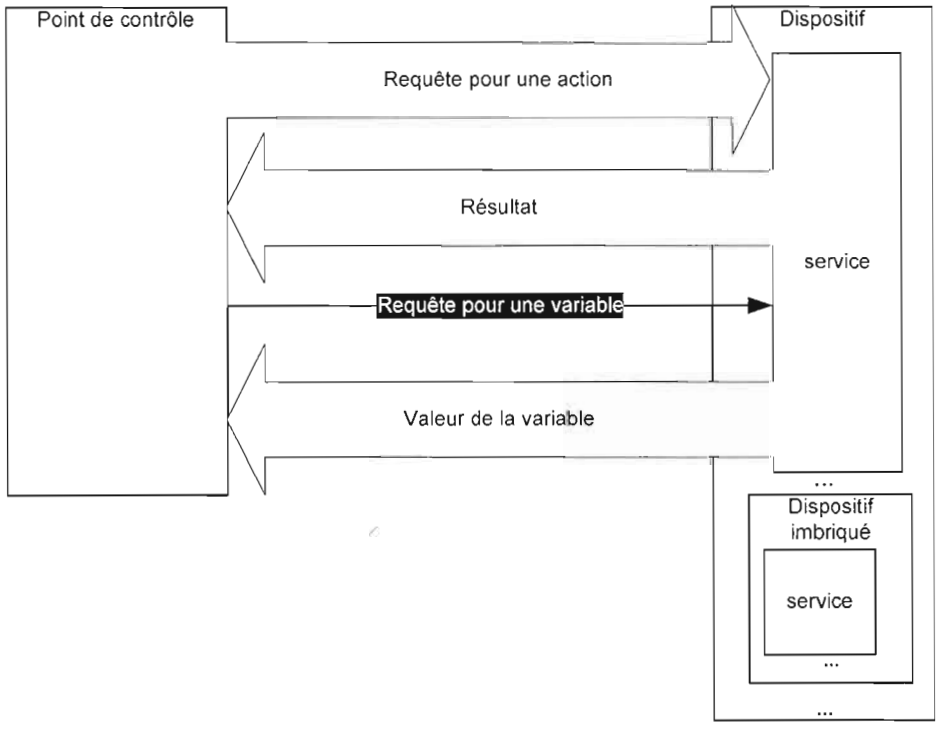

Figure 3.12 - Contrôle d'un service

3.3.4.1 Les protocoles de la phase contrôle

À la couche supérieure, les messages de descriptions contiennent des informations spécifiques du vendeur, par exemple, les valeurs d'arguments. Dans la couche inférieure, des informations de *UPnP Forum Working Committee* sont rajoutées au contenu de *UPnP Vendor*, par exemple, les noms d'actions, les noms d'arguments ou les noms de variables.

Les messages des couches supérieures reposent sur *UPnP-Specific Protocols*. Ils sont formatés en utilisant les éléments d'en-têtes et de corps de SOAP. Ces messages sont livrés via HTTP au-dessus de TCP et IP.

3.3.4.2 Invocation des actions et les réponses

Le client invoque les actions des services du serveur et reçoit en retour les résultats ou les messages d'erreurs. Les actions, les résultats et les erreurs sont encapsulés dans SOAP.

SOAP définit l'utilisation de XML et de HTTP pour RPC. UDA (*Universal Data Access*) utilise SOAP pour livrer les messages de contrôle aux serveurs et retourner les résultats ou erreurs aux clients.

Pour rendre obligatoire certains champs dans une requête HTTP, les méthodes HTTP doivent être préfixées par M-, afin de mieux contrôler le passage des différentes requêtes à travers un pare-feu.

Pour fournir une plus grande flexibilité administrative aux pare-feu et aux serveurs mandataires (proxies), SOAP indique que les requêtes doivent d'abord être essayées sans en-tête MAN et sans préfixe M-. Si la requête est rejetée avec la réponse "405 Method Not Allowed", alors une deuxième requête doit être envoyée en utilisant l'en-tête MAN et le préfixe M-. Si cette requête est rejetée avec la réponse "501 Not Implemented" ou "510 Not Extented", la requête échoue.

Si l'invocation d'une action est optionnelle alors elle commence avec la méthode POST défini par HTTP, par contre si l'invocation d'une action est obligatoire alors elle commence avec la méthode M-POST. Les figures 3.13 et 3.14 représentent les méthodes POST ou M-POST suivies de l'URL de contrôle avec la version de HTTP. Cette invocation se

fait vers l'adresse « HOST : adresse URL : numéro de port », le message comporte d'autres champs: « CONTENT-LENGTH » indique la longueur en octet du corps du message; l'en-tête « CONTENT-TYPE » est *text/xml* avec un code caractère *utf-8;* « SOAPACTION » est un en-tête définie par SOAP qui comporte le type du service et le nom de l'action; le corps du message comporte une enveloppe SOAP, qui contient en XML l'action à exécuter et ses arguments. Si la requête utilise M-POST, l'en-tête « MAN » devient obligatoire suivi de http://schemas.xmlsoap.org/soap/envelope/ avec un espace nom pour chaque en-tête SOAP.

POST *path of control URL* **HTTP/1.1**
HOST: *host of control URL:port of control URL*
CONTENT-LENGTH: *bytes in body*
CONTENT-TYPE: text/xml; charset="utf-8"
SOAPACTION: "urn:schemas-upnp-org:service:*serviceType:v#actionName***"**
<?xml version="1.0"?>
<s:Envelope
xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<s:Body>
<u:*actionName* xmlns:u="urn:schemas-upnp-org:service:*serviceType:v***">**
<argumentName>*in arg value****</argumentName>***
other in args and their values go here, if any
</u:*actionName*>
</s:Body>
</s:Envelope>

Figure 3.13 : Requête d'invocation d'une action avec POST

M-POST *path of control URL* **HTTP/1.1**
HOST: *host of control URL:port of control URL*
CONTENT-LENGTH: *bytes in body*
CONTENT-TYPE: text/xml; charset="utf-8"
MAN: "http://schemas.xmlsoap.org/soap/envelope/"; ns=01
01-SOAPACTION: "urn:schemas-upnp-org:service:*serviceType:v#actionName***"**

Figure 3.14 : Requête d'invocation d'une action avec M-POST et MAN

Le service doit compléter l'invocation de l'action et la réponse dans les 30 secondes, incluant le temps de transmission des exceptions.

La figure 3.15 représente la réponse à une requête de contrôle commence avec « HTTP/1.1 200 OK » qui signifie la version de HTTP et un code HTTP de succès, Le message comporte d'autres champs: « CONTENT-LENGTH » indique la longueur en octet du corps du message; l'en-tête « CONTENT-TYPE » contient *text/xml*; et un code caractère *utf-8;* « DATE » défini la date de la génération de la réponse, il est optionnel; l'en-tête « EXT » sans valeurs confirme que le « MAN » a été compris; « *SERVER* » détermine la version du protocole UPnP, le nom et la version du service afin d'éviter les incompatibilités, le corps du message comporte une enveloppe SOAP, qui contient en XML la réponse à l'action à exécuter et ses arguments.

HTTP/1.1 200 OK
CONTENT-LENGTH: *bytes in body*
CONTENT-TYPE: text/xml; charset="utf-8"
DATE: *when response was generated*
EXT:
SERVER: *OS/version* **UPnP/1.0** *product/version*
<?xml version="1.0"?>
<s:Envelope
xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<s:Body>
<u:actionNameResponse xmlns:u="urn:schemas-upnp-org:service:*serviceType:v***">**
<argumentName>*out arg value***</argumentName>**
other out args and their values go here, if any
</u:actionNameResponse>
</s:Body>
</s:Envelope>

Figure 3.15 : La réponse du service

3.3.5 Phase événement

La phase événement est liée à la phase contrôle dont le point de contrôle envoie des actions aux dispositifs. À travers la phase événement, le point de contrôle peut écouter les changements d'états des dispositifs.

La description du service UPnP inclut une liste d'actions et une liste de variables qui modélisent l'état du service au temps d'exécution.

Si une ou plusieurs de ces variables ont changé, alors le service publie des mises à jour et le point de contrôle s'abonne pour recevoir ces informations.

Le service prend note des changements des variables d'états en envoyant des messages d'événements. Ces derniers contiennent les noms d'une ou plusieurs variables d'états et leurs valeurs courantes. Le premier message d'événement spécial est envoyé quand le point de contrôle s'abonne pour la première fois.

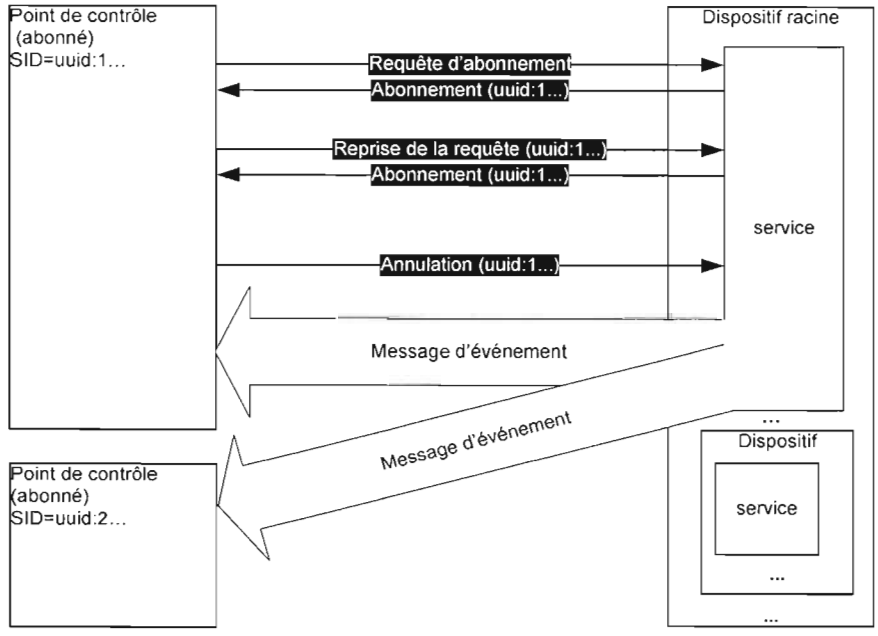

Figure 3.16 - Les événements d'un service

Pour s'abonner à l'événement, le point de contrôle envoie un message d'abonnement au service. L'identificateur du service (*serviceId element*), l'URL du service et l'URL d'événement (*eventSubURL element*) sont dans le message de description. Si l'abonnement est accepté, le service répond avec un identifiant unique et un temps *t* d'expiration de l'abonnement. Pour garder l'abonnement actif il faut que le point de contrôle relance la demande d'abonnement avant que la durée d'abonnement expire. Un point de contrôle peut annuler un abonnement quand il n'en a plus besoin (voir Figure 3.16).

Dès que l'abonnement est accepté, le service envoie le message d'événement initial à l'abonné. Ce message inclut les noms et les valeurs courantes des variables d'événements. Ce message est toujours envoyé, même si le point de contrôle se désabonne avant sa livraison.

Le service doit s'assurer que le point de contrôle a reçu la réponse à la requête d'abonnement (SID) avant l'envoi du message d'événement initial. Si l'abonnement expire, l'identificateur d'abonnement devient invalide, et le service arrête d'envoyer les messages d'événements à l'abonné (point de contrôle) puis le retire de sa liste d'abonnés.

3.3.5.1 Les protocoles de la phase événement

À la couche supérieure, l'abonnement et les messages d'événements contiennent des informations spécifiques du vendeur, par exemple, les URLs pour l'abonnement et la durée de l'abonnement, ou les valeurs spécifiques des variables. Dans la couche inférieure, des informations de *UPnP Forum Working Committee* sont ajoutées au contenu de *UPnP Vendor*, par exemple, les identificateurs de services ou les noms de variables.

Les messages des couches supérieures reposent sur des protocoles spécifiques à UPnP et sont livrés via HTTP en utilisant des méthodes et en-têtes de GENA, au-dessus de TCP/IP.

Le service maintient une liste d'abonnés (points de contrôle) ayant pour chaque abonné les informations suivantes :

- identificateur d'abonnement unique (*Unique Subscription Identifier*) ;
- URL pour les messages d'événements ;
- clé d'événement (*Event Key*) ;

- durée d'abonnement (*Subscription Duration*).

Le message d'abonnement contient la méthode « SUBSCRIBE » qui inscrit pour la première fois un abonné ou un renouvellement de son inscription. Le message est envoyé à l'adresse URL de l'événement « HOST », l'en-tête « CALLBACK » contient une à plusieurs URLs où le service envoie ses messages d'événements, le type de notification « NT » doit être *upnp:event*, « TIMEOUT » qui est optionnel contient la durée de la requête jusqu'à expiration de l'inscription, « SID » identificateur unique d'une inscription qui concerne le message du renouvellement d'une inscription d'un abonné (voir Figure 3.17, 3.18 et 3.19).

SUBSCRIBE *publisher path* HTTP/1.1
HOST: *publisher host:publisher port*
CALLBACK: <*delivery* URL>
NT: *upnp:event*
TIMEOUT: *Second-requested subscription duration*

Figure 3.17 - Abonnement : SUBSCRIBE avec NT et CALLBACK

SUBSCRIBE publisher path HTTP/1.1
HOST: publisher host:publisher port
SID: uuid:subscription UUID
TIMEOUT: Second-requested subscription duration

Figure 3.18 - Renouvellement : SUBSCRIBE avec SID

UNSUBSCRIBE publisher path HTTP/1.1
HOST: publisher host:publisher port
SID: uuid:subscription UUID

Figure 3.19 - Annuler l'abonnement : UNSUBSCRIBE

3.3.5.2 Messages d'événements

Le service publie le changement survenu sur les variables d'états en envoyant des messages d'événements en utilisant la méthode « NOTIFY » suivie du chemin de l'URL de l'événement avec la version de HTTP. Ce message contient dans son corps les noms d'une ou plusieurs variables d'états et leurs valeurs courantes. Ils doivent être envoyés le plus tôt possible pour obtenir l'information exacte du service aux abonnés et leur permettre l'affichage de l'interface utilisateur. Si plusieurs valeurs de variables sont changées en même temps, le service introduit ces changements dans un même message d'événement.

Pour un abonnement, la clé d'événement « SEQ :*Event Key* » est initialisée à « 0 » avec le message initial d'événement. Pour les messages d'événements subséquents, le service incrémente la clé d'événement et inclut la mise à jour de la clé dans le message d'événement. L'en-tête « NT » désigne le type de notification qui doit être *upnp:event*, L'en-tête « NTS » désigne le sous type de notification qui doit être *upnp:propchange*, « SID » identificateur unique d'une inscription. Le corps du message a une longueur en octet « CONTENT-LENGTH », l'en-tête « CONTENT-TYPE » doit être text/xml; et de code caractère utf-8,

S'il n'y a pas de réponse de la part des abonnés, le service doit continuer à leurs envoyer des messages d'événements « NOTIFY » jusqu'à l'expiration de l'abonnement. Et si un abonné répond à un message d'événements alors le service arrête de lui envoyer ce même message (voir Figure 3.20).

NOTIFY delivery path HTTP/1.1
HOST: delivery host:delivery port
CONTENT-TYPE: text/xml
CONTENT-LENGTH: Bytes in body
NT: upnp:event
NTS: upnp:propchange
SID: uuid:subscription-UUID
SEQ: event key

```
<?xml version="1.0"?>
<e:propertyset xmlns:e="urn:schemas-upnp-org:event-1-0">
 <e:property>
  <variableName>new value</variableName>
 </e:property>
 Other variable names and values (if any) go here.
</e:propertyset>
```

Figure 3.20 - Messages d'événements : NOTIFY

3.3.6 Phase présentation

Si un équipement dispose d'une URL pour la présentation, le point de contrôle peut charger la page dans son navigateur, ce qui permet à l'utilisateur de contrôler l'équipement et visualiser son état (voir Figure 3.21).

L'URL pour la présentation est contenue dans l'élément *presentationURL* qui se trouve dans le fichier de description du dispositif.

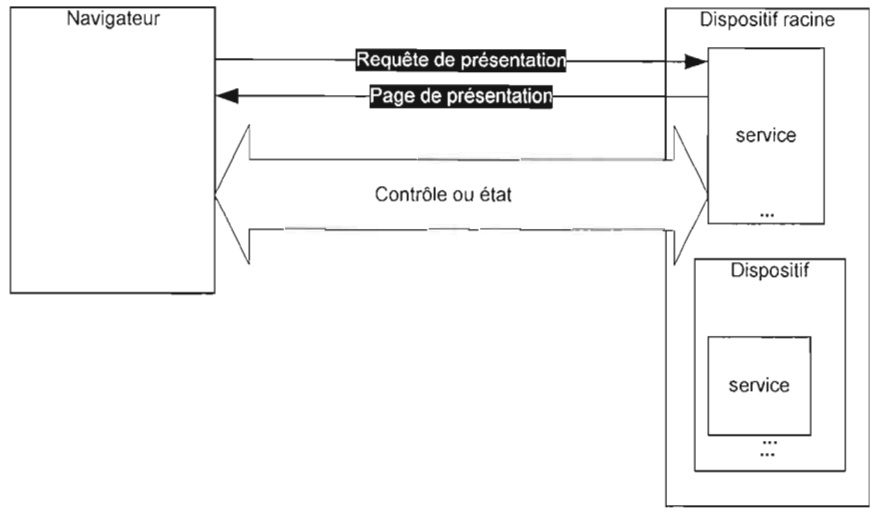

Figure 3.21 - Visualisation pour contrôler et savoir l'état du dispositif

3.3.6.1 Les protocoles de la phase présentation

À la couche supérieure, la page de présentation est spécifiée par le vendeur. Dans la couche inférieure, le *UPnP Device Architecture* spécifie que cette page est écrite en HTML. Cette page est livrée via HTTP par-dessus TCP/IP. Pour retrouver une page de présentation, le point de contrôle lance une requête HTTP GET vers l'URL de la page de présentation, et le dispositif retourne la page de présentation.

SEDIRAN

SEDIRAN (*Service Discovery and Interaction with Routing Protocols in Ad-hoc Network*) [22] est un protocole de découverte de services qui est au-dessus du protocole de routage réactif AODV. Sa stratégie est donc l'annonce et la recherche de services en couplage avec les protocoles de routage.

3.4 Protocole de routage AODV

Le protocole AODV (*Ad-hoc On Demand Distance Vector*) est un système d'acquisition de route à la demande. C'est un protocole réactif. Il n'y a pas d'information de routage maintenue sur les nœuds en dehors des chemins actifs. La recherche de route se fait uniquement lors d'une communication ou si le nœud est sur un chemin actif, c'est-à-dire s'il y a une retransmission de paquets.

Il utilise un numéro de séquence pour assurer la fraîcheur des routes et un mécanisme d'expiration de routes pour nettoyer la table de routage [20].

Parmi les objectifs du protocole AODV, on peut citer :

- Diffuser les paquets de découvertes uniquement lorsque cela est nécessaire.
- Faire une distinction entre la gestion de la connectivité locale et la maintenance de la topologie générale.
- Diffuser les informations lors de changements dans la connectivité locale aux nœuds voisins qui en ont besoin.

Il existe trois types de requêtes principales dans AODV : RREQ (*Route Request*), RREP (*Route Reply*) et RERR (*Route ERRor*).

La découverte de chemins est un processus qui se déclenche dès qu'un nœud source désire communiquer avec un nœud au sujet duquel il n'a aucune information de routage. Il diffuse une requête RREQ via le réseau et, qui contient les champs suivants : (*RREQ ID, Adresse IP Destination, Numéro de séquence Destination, Adresse IP Source, Numéro de séquence Source, Nombre de Sauts*)

La réaction des voisins se fait par une réponse RREP au nœud source, si le nœud répondeur est concerné par la requête. Il incrémente son propre numéro de séquence destination pour assurer la fraîcheur des entrées de la table de routage.

Sinon, une retransmission de la requête est faite aux voisins, avec une sauvegarde d'une trace de l'information pour la construction du chemin inverse. Cette trace de l'information sera l'adresse source, le numéro de séquence source, l'adresse de destination, le numéro de séquence destination, le dernier saut, le RREQ ID et la durée de validité pour l'entrée de routage du chemin inverse.

Si un nœud reçoit un RREQ, et qu'il possède une route correspondante valide où que le paquet arrive à destination avec un numéro de séquence supérieur ou égal à celui de RREQ reçu, il envoie un RREP au nœud source. Le RREP utilise le chemin inverse mis en place. Les nœuds le long du chemin mettent à jour leur table de routage (*timer*, numéro de séquence et prochain saut). Sinon, il rediffuse le RREQ. RREP contient les champs suivants : (*Adresse IP Destination, Numéro de séquence Destination, Adresse IP Source, durée de vie*)

Lorsque le nœud source reçoit le message RREP, il peut commencer à émettre les paquets de données. Mais, si le nœud source reçoit un RREP contenant un numéro de séquence supérieur à celui enregistré dans la table de routage, ou égal avec un nombre de sauts plus petit, alors il met à jour l'information de routage vers cette destination.

Les informations des tables de routage doivent être gardées même pour les routes de courte durée qui sont créées pour stocker temporairement les chemins de retour vers les nœuds sources. Une table de routage contient les champs suivants : (Adresse IP Destination, Numéro de Séquence Destination, Nombre de sauts, Prochain saut, Liste des voisins actifs, Temps d'expiration de la route)

Les nœuds enregistrent les adresses IP sources et les identificateurs des requêtes RREQ IDs dans une table de diffusion, pour savoir si une requête spécifique a été déjà reçue.

Un lien est considérée actif tant que les paquets de données transitent périodiquement de la source à la destination. Mais, le lien expirera si l'émission de paquets de données ne s'est pas faite pendant un temps défini et, par conséquent, il sera effacé des tables de routages des nœuds intermédiaires.

Si un lien se rompt d'une façon brusque, le nœud qui se trouve à l'extrémité du lien avise les nœuds utilisant ce lien, qu'il n'est plus possible d'atteindre la destination en utilisant le message RERR (*Route ERRor*).

3.5 Services spéciaux et services ordinaires

Dans SEDIRAN, la découverte de services s'appuie sur deux notions de services : les services spéciaux et les services ordinaires.

Les services ordinaires sont des services connus par les nœuds du réseau. Leurs descriptions sont stockées dans une base de données qui a la forme d'un arbre de type de services. Chaque service ordinaire appartient à un type de service.

Les services spéciaux ne sont pas connus des nœuds du réseau. Ils ne font pas partie de la base de données de types de services mais chacun est décrit par une chaîne de caractères afin que les utilisateurs puissent connaître l'utilité du service.

L'annonce de services se fait avec ceux qui ne sont pas a priori connus des nœuds du réseau. C'est aux services spéciaux de répondre à cette exigence. La découverte de services se fait sur les services ordinaires déjà connus des nœuds du réseau.

Dans SEDIRAN, il y a un gestionnaire de cache pour les services ordinaires et un autre pour les services spéciaux.

3.6 Découverte de services dans SEDIRAN

SEDIRAN utilise le message RREP du protocole de routage AODV pour encapsuler tous ses types de messages (DREQ, DREP et ADVM). Le message RREP a subi quelques modifications pour s'adapter à la nouvelle situation (voir Figure 3.22).

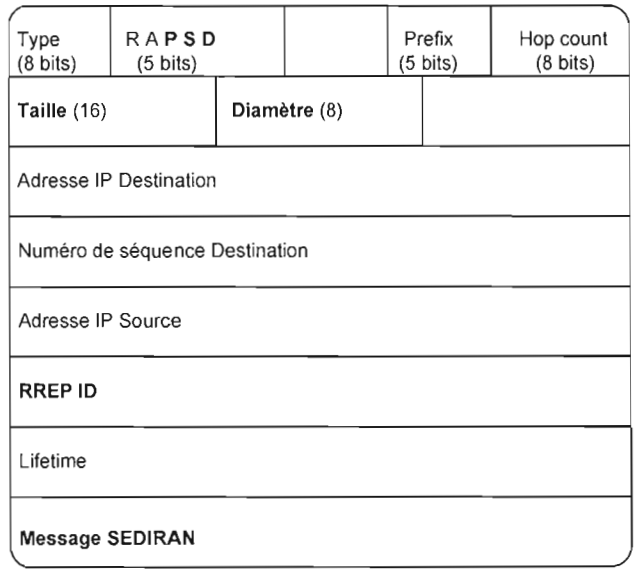

Figure 3.22 - Paquet RREP de AODV modifié

3.6.1 Annonce de services dans SEDIRAN

Un nœud, qui veut annoncer un service spécial, diffuse (*broadcast*) un message de d'annonce ADVM (*ADVertisement Message*) aux nœuds du réseau ad-hoc, lorsque le champ P de RREP vaut 1. Il contient l'adresse du nœud source (*serveur*), l'identificateur du service, sa description et son bail. Ce dernier permet de délimiter la durée de vie du service stocké dans le cache.

3.6.2 Recherche de service dans SEDIRAN

Un nœud qui veut rechercher un service ordinaire diffuse (*broadcast*) un message de recherche DREQ (*Discovery REQuest*) aux nœuds du réseau ad-hoc. Il y a un champ dans le message qui désigne le nombre maximal de sauts (*Hop Limit*), c'est-à-dire le nombre de nœuds que le message peut traverser. La procédure de recherche commence toujours par *Hop-Limit* égal à 1. Si les découvertes ne sont pas satisfaisantes ou l'utilisateur veut trouver plus, il passe à *Hop-Limit* égal à 2 et ainsi de suite. Cependant, seuls les nœuds qui offrent des services différents de ceux découverts aux niveaux précédents répondent. Cette procédure s'applique aussi pour quelqu'un qui veut rechercher tous les services du réseau.

Les nœuds intermédiaires mettent à jour leurs tables de routage vers le nœud source et rediffusent le message de requête. Quand ils reçoivent des réponses, ils mettent à jour la table de routage vers le fournisseur de services et ajoutent les services découverts dans le cache des services ordinaires.

Le message DREQ contient les champs suivants : le champ « *Type* » vaut DREQ, le champ « *Hop-Limit* » contient le nombre de sauts entre le nœud source et les nœuds cibles, le champ « *SourceIPAddr* » contient l'adresse source de l'émetteur de la requête et le champ « *ServiceTypeID* » identifie le type de service ordinaire recherché. Le champ « *ServiceCount* » indique le nombre de services déjà découverts par le nœud client, les champs « *ServicesID* » contiennent la liste des identificateurs des services.

3.6.3 Message de réponse dans SEDIRAN

Le message de réponse DREP répond à une requête de découverte de services. Il contient les champs suivants : le champ « *Type* » contient la valeur DREP, le champ « *ServiceCount* » indique le nombre de services ordinaires découverts, le champ « *ResponderIPAddr* » indique l'adresse IP du fournisseur du service, le champ « *OriginatorIPAddr* » indique l'adresse IP du nœud demandeur de service, et puis la liste des services ordinaires découverts décrite par trois champs, le champ « *ServiceID* » qui indique l'identificateur du service, le champ « *ServiceTypeID* » indique l'identificateur du type de service et le champ « *LifeTime* » indique la durée de vie du service dans le cache.

3.7 Fonctionnement de SEDIRAN

Un message d'annonce « ADVM » est passé au protocole de routage « AODV » afin de l'encapsuler dans un paquet « RREP » et le diffuser (*broadcast*) sur le réseau. Lorsqu'un nœud reçoit le message, grâce au champ « RREP ID », le protocole « AODV » vérifie si le message a été déjà reçu. Dans le cas contraire, il met à jour la route vers le nœud source et rediffuse le message. Il passe ensuite au protocole SEDIRAN le message ADVM afin de mettre à jour le cache des services spéciaux (voir Figure 3.23).

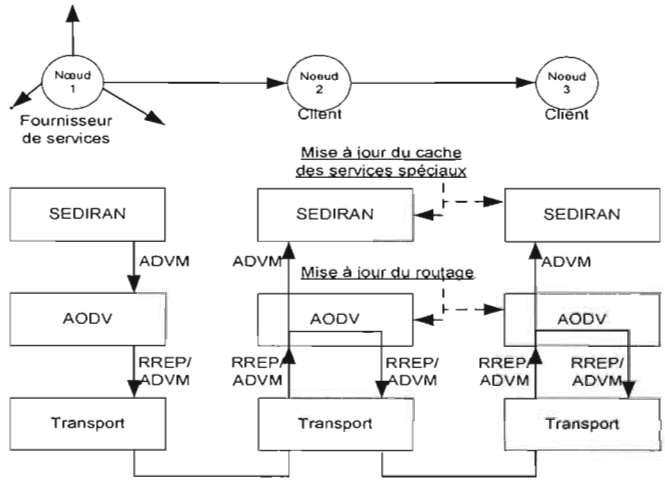

Figure 3.23 - Propagation des messages ADVM

Un message de recherche « DREQ » est passé au protocole de routage « AODV » afin de l'encapsuler dans un paquet « RREP » et le diffuser (*broadcast*) sur le réseau. Lorsqu'un nœud reçoit le message, le protocole « AODV » met à jour la route vers le nœud source, et si le nœud est le dernier du nombre de sauts *Hop-Limit* choisi, alors il passe le message « DREQ » au protocole SEDIRAN qui va vérifier si le service cherché est dans la table des services locaux (voir Figure 3.24).

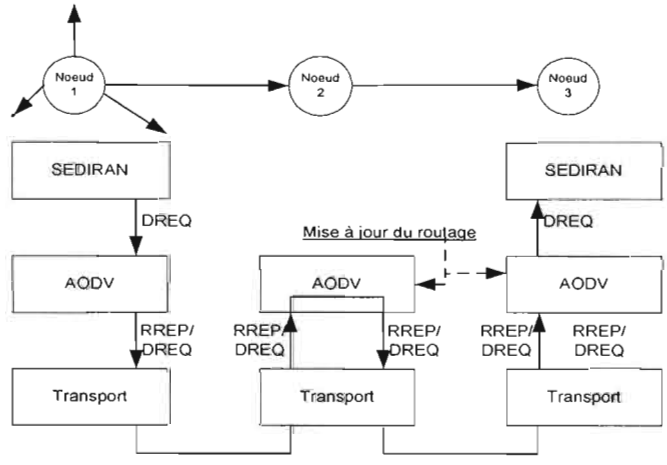

Figure 3.24 - Propagation de messages DREQ

Un message de réponse « DREP » est passé au protocole de routage « AODV » afin de l'encapsuler dans un paquet « RREP » et l'envoyer au nœud demandeur (*client*) en unicast. Lorsqu'un nœud intermédiaire ou source reçoit le message, le protocole « AODV » met à jour la route vers le nœud fournisseur du service. Pour le nœud intermédiaire, une invocation ultérieure du service ne produira pas une découverte de la route. Le nœud demandeur passe le message DREP au protocole SEDIRAN afin de mettre à jour le cache des services ordinaires (voir Figure 3.25).

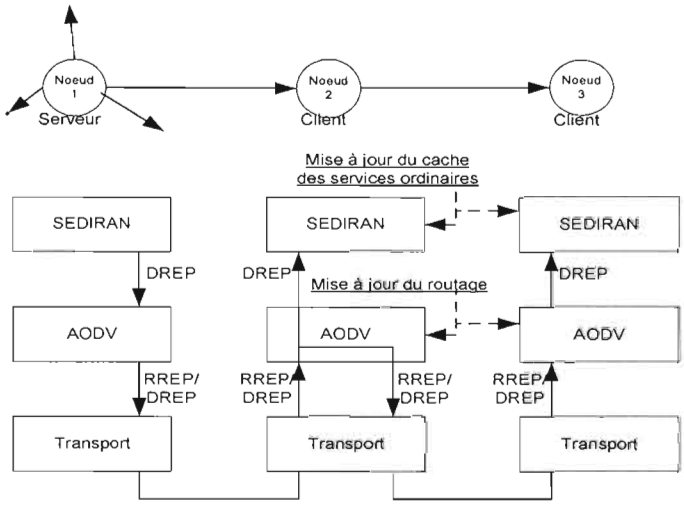

Figure 3.25 - Envoie du message DREP

CHAPITRE IV

UNIVERSAL PLUG AND PLAY POUR RÉSEAU AD-HOC

Dans ce chapitre, nous décrivons notre proposition pour faire fonctionner le protocole de découverte de services UPnP dans un réseau spontané *(ad-hoc)*.

4.1 Introduction

Cette étude concerne principalement la stratégie de découverte de services dans les réseaux ad-hoc. On propose d'intégrer le protocole de découverte de services SEDIRAN à UPnP pour qu'il devienne fonctionnel aussi bien dans les réseaux classiques que dans les réseaux ad-hoc.

Cependant, nous pensons que le protocole de découverte de services SEDIRAN ne sera pas le seul à pouvoir intégrer l'UPnP.

UPnP va bénéficier des avantages du protocole SEDIRAN qui sont :

- Il y a un couplage entre le protocole de découverte de services dans les réseaux ad-hoc SEDIRAN et le protocole de routage AODV. L'architecture est sous forme de couche, le protocole SEDIRAN étant au-dessus du protocole réactif AODV.

- Au moment de l'annonce ou de la recherche de services, les tables de routages sont enrichies, et de cette façon, l'invocation de services et les messages de réponses DREP (*Discovery Reply*) ne déclenchent pas le processus de découverte de routes AODV.

- La recherche d'un service ordinaire avec le protocole SEDIRAN se fait avec le message DREQ (*Discovery Request*). Ce dernier contient dans son en-tête *Hop-limit* une valeur désignant le nombre maximal de sauts que le message est autorisé à effectuer entre les nœuds du réseau jusqu'à ce qu'il arrive au niveau de nœuds voulu. Donc, la première recherche cible le niveau 1 des nœuds, c'est-à-dire *Hop-limit=1,* et si les réponses sont satisfaisantes, la recherche s'arrête. Sinon, la recherche passe au niveau 2, et ainsi de suite. Les nœuds qui possèdent des services déjà découverts dans les nœuds des niveaux inférieurs ne répondent pas aux requêtes de recherche.

- SEDIRAN contient un cache qui permet de stocker les services déjà découverts pour une éventuelle utilisation ultérieure. Donc, il améliore considérablement le processus de découverte de services.

- La découverte de services avec le protocole SEDIRAN est basée sur la notion de services spéciaux et services ordinaires. Les services spéciaux ne sont pas connus des utilisateurs (points de contrôle). Donc, ils doivent être annoncés, alors que les services ordinaires sont connus, ce qui veut dire que leurs annonces ne sont pas nécessaires.

UPnP est destiné aux équipements. Il permet de les rechercher, les contrôler et visualiser leurs états, UPnP pour réseau ad-hoc est destiné à la découverte et l'interaction avec les services, qu'ils soient matériels ou logiciels (voir Figure 4.1).

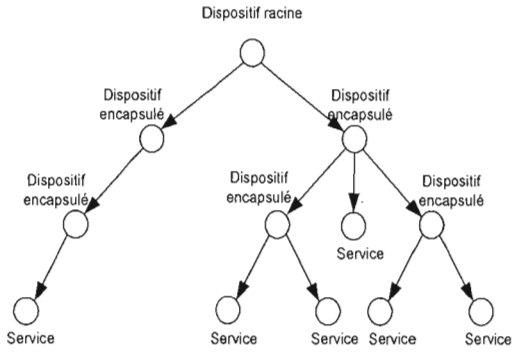

Figure 4.1a - Structure générale d'un dispositif UPnP

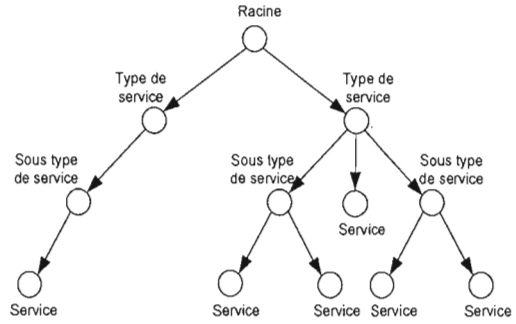

Figure 4.1b - Structure générale d'un dispositif UPnP pour ad-hoc

Dans UPnP pour réseau ad-hoc, les fichiers de descriptions sont enregistrés localement dans les nœuds qui offrent les services correspondants.

4.2 Protocoles de la phase découverte

Comme déjà indiqué dans le chapitre III, *Universal Plug and Play* passe par plusieurs étapes pour son exécution entière.

UPnP fonctionne dans les réseaux filaires et sans fil via des points d'accès. Ce n'est pas le cas dans les réseaux ad-hoc, où son protocole de découverte de service SSDP n'a pas été conçu pour ce genre de réseaux.

Par conséquent, nous proposons d'intégrer dans l'architecture UPnP un protocole de découverte de services fonctionnant dans les réseaux ad-hoc et, de cette façon, UPnP devient aussi fonctionnel dans ce genre de réseaux. Nous avons choisi SEDIRAN comme protocole à intégrer dans UPnP.

C'est la phase découverte qui sera concernée par les modifications pour adapter UPnP au réseau ad-hoc, tandis que les autres étapes restent telles quelles (voir Figure 4.2).

Au-dessus des couches du protocole UPnP, il y a les APIs qui sont spécifiques aux vendeurs. En dessous du protocole UPnP, il y a un système et une plate-forme spécifique

aussi aux vendeurs, c'est-à-dire que les vendeurs peuvent choisir leur propre modèle de programme, leur propre système et leur propre plate-forme pour implémenter UPnP.

Les couches du protocole UPnP sont divisées en deux parties : les protocoles spécifiques à UPnP et les protocoles standards (voir Figure 4.2).

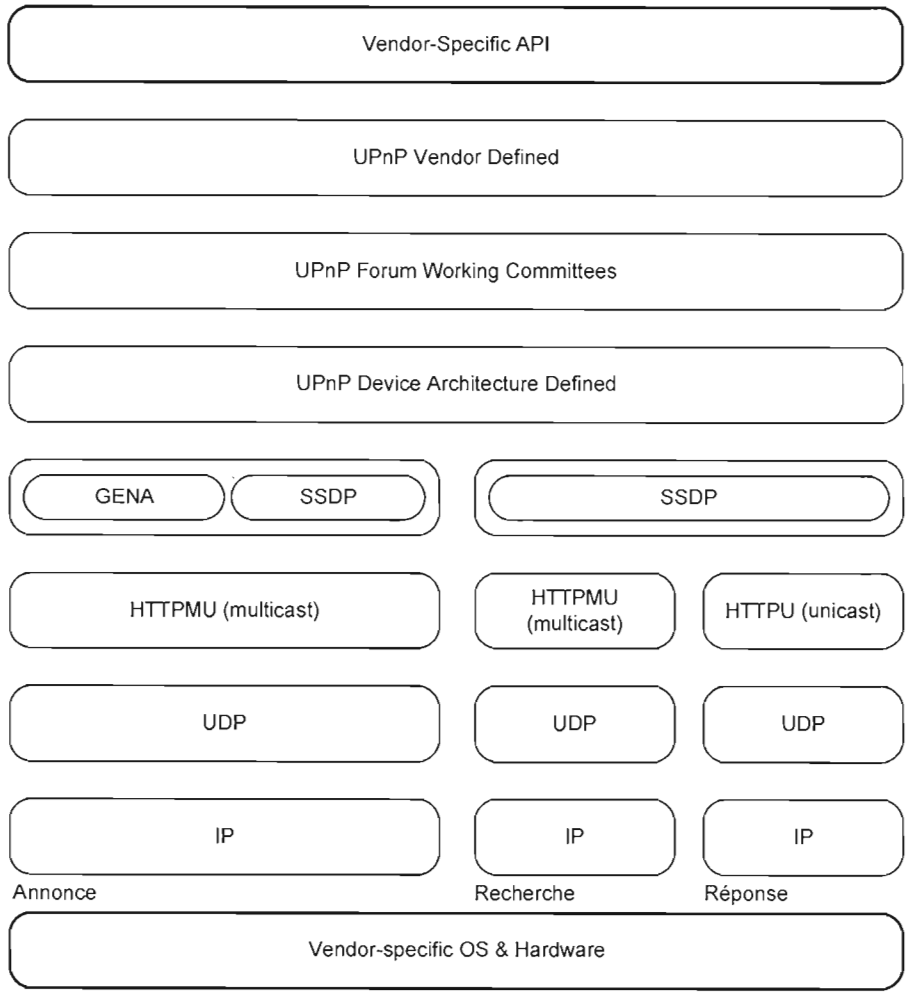

Figure 4.2 - Les couches de la phase de découverte UPnP [14]

Notre protocole choisit SEDIRAN qui sera intégré juste en-dessous des protocoles spécifiques à UPnP et au-dessus de TCP/UDP. Donc, pour le nouveau protocole UPnP pour réseau ad-hoc, SSDP, les protocoles HTTPMU, HTTPU et GENA ne seront pas utiles

dans la phase découverte. Mais, le protocole GENA restera fonctionnel dans la phase événement (voir Figure 4.3 et 4.4).

L'intégration du protocole SEDIRAN dans UPnP nécessite d'intégrer dans ses messages les en-têtes qu'on trouve dans les messages du protocole SSDP et pas dans les messages du protocole SEDIRAN, ces en-têtes sont nécessaire pour la suite des phases de UPnP comme « SERVER » qui détermine la version du protocole UPnP, le nom et la version du service afin d'éviter les incompatibilités et « NT » qui nous informe du type de service et « LOCATION » qui contient l'URL du fichier de description du service.

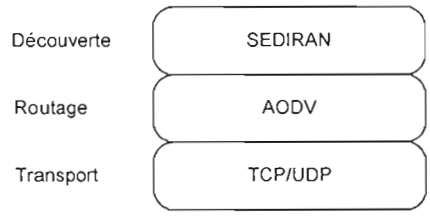

Figure 4.3 - Les couches de protocoles de SEDIRAN

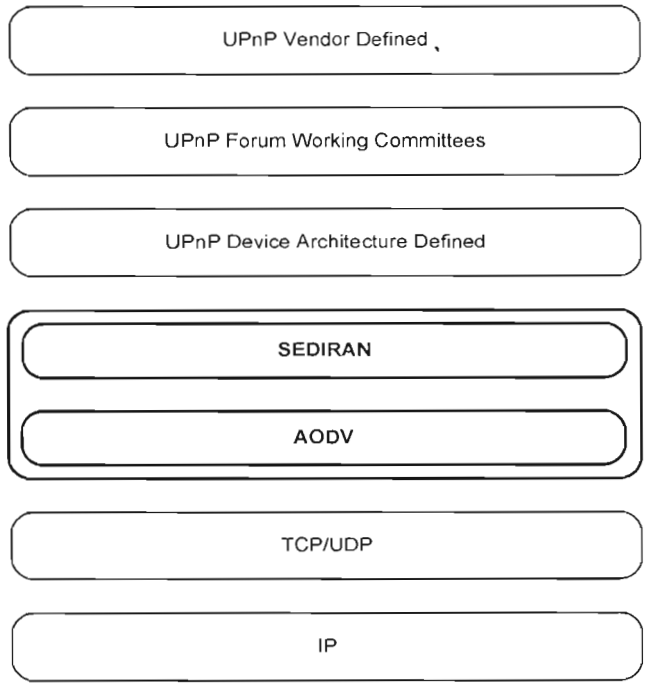

Figure 4.4 - Les couches de la phase de découverte UPnP pour réseau ad-hoc

4.3 Phase Découverte

4.3.1 Message d'annonce dans UPnP

4.3.1.1 Dispositif disponible

Dans UPnP, un dispositif (*Device*) qui vient de joindre un réseau annonce sa présence en envoyant des messages HTTP à l'adresse multicast « Host : 239.255.255.250 :1900 », l'annonce a une durée maximale de « CACHE-CONTROL », le type du service recherché est « NT », avec un identificateur unique « USN » en utilisant le protocole SSDP avec l'en-tête « NTS », la description du service se trouve à l'URL désignée dans l'en-tête « LOCATION » et « SERVER » qui détermine la version du protocole UPnP, le nom et la version du service afin d'éviter les incompatibilités, pour plus de détails voir la sous-section 3.3.2.1.

4.3.1.2 Dispositif non disponible

Dans UPnP, un dispositif (*Device*) qui quitte un réseau annonce qu'il doit quitter en envoyant un message HTTP à l'adresse multicast « Host : 239.255.255.250 :1900 » pour chaque message d'annonce de disponibilité envoyé, le type du service recherché est « NT », avec un identificateur unique « USN » en utilisant le protocole SSDP avec l'en-tête « NTS ».

Si le dispositif quitte le réseau brusquement sans annoncer sa sortie, alors le message d'annonce de disponibilité restera dans les caches des points de contrôle, jusqu'à son expiration telle que spécifie l'en-tête CACHE-CONTROL, pour plus de détails voir la sous-section 3.3.2.1.

4.3.2 Message d'annonce dans UPnP pour réseau ad-hoc

Dans ce qui suit nous allons exposer l'annonce de disponibilité dans UPnP pour réseau ad-hoc adaptée dans notre travail.

4.3.2.1 Message d'annonce de disponibilité dans UPnP pour réseau ad-hoc

Dans UPnP pour réseau ad-hoc, le message d'annonce se fait en diffusion (*broadcast*) et non en multicast comme c'était le cas en UPnP version SSDP (voir la sous-section 3.5.1). C'est SEDIRAN qui se charge d'annoncer la disponibilité en diffusion d'un service ou d'un type de services, ou d'un type racine.

Le message d'annonce ne comportera plus les variables suivantes : « NOTIFY : fonction d'annonce » appartenant au protocole GENA, qui n'est plus utilisé dans la phase découverte, et « Host : 239.255.255.250 :1900 », qui fournit l'adresse multicast. Le numéro de port n'est plus valable dans notre cas, car SEDIRAN fait ses annonces en diffusion (*broadcast*).

Donc, le message d'annonce de UPnP, c'est l'ADVM modifié de SEDIRAN qui contenait avant la modification les champs : Type, Adresse IP Source, SpecialService.ServiceID, SpecialService.ServiceDesc et SpecialService.Lifetime (voir Figure 4.5).

Nous avons adapté le message d'annonce ADVM en ajoutant deux champs : « SERVER » qui détermine la version du protocole UPnP, le nom et la version du service afin d'éviter les incompatibilités et « NT » qui nous informe du type de service. Pour le reste des champs, nous avons fait les correspondances entre ceux du message d'annonce ADVM et ceux de UPnP, sauf pour « Adresse IP Source » qui est spécifique à SEDIRAN (voir Figure 4.6).

Correspondances entre les champs équivalents

Type ~ NTS

SpecialService.ServiceID ~ USN

SpecialService.ServiceDesc ~ LOCATION

SpecialService.Lifetime ~ CACHE-CONTROL

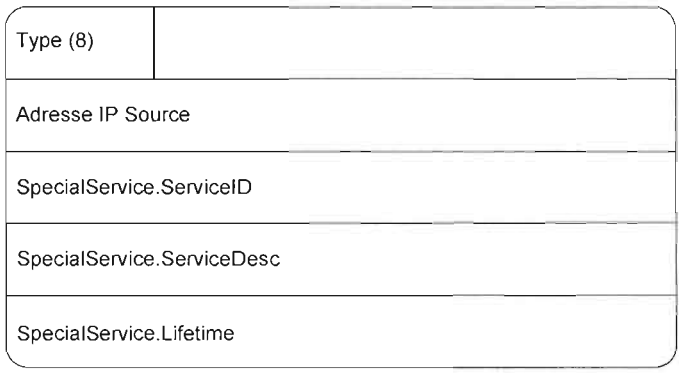

Figure 4.5 - Message d'annonce ADVM du SEDIRAN

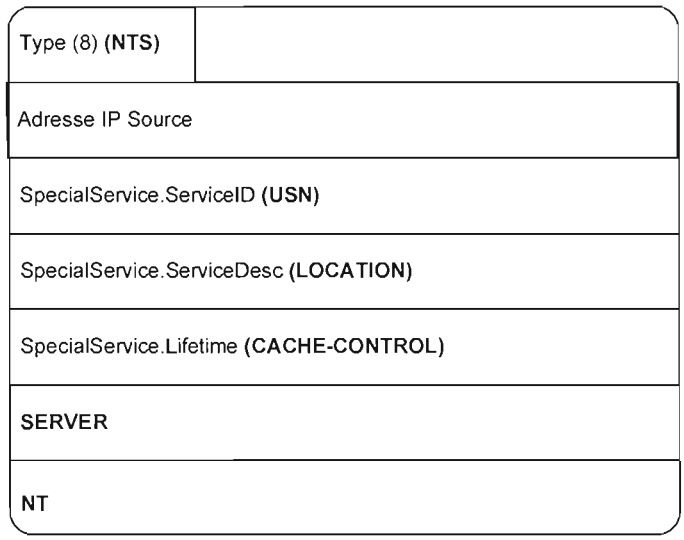

Figure 4.6 - Message d'annonce ADVM du SEDIRAN modifié

4.3.2.2 Message d'annonce de non disponibilité dans UPnP pour réseau ad-hoc

Dans UPnP pour réseau ad-hoc, un serveur (fournisseur de services) qui quitte un réseau d'une façon normale ou brusque ne fait aucune annonce. Alors, le message d'annonce de disponibilité restera dans les caches des clients jusqu'à son expiration, qui est spécifiée dans son en-tête CACHE-CONTROL.

4.3.3 Message de recherche dans UPnP

Dans UPnP, un point de contrôle (client) qui vient de joindre un réseau recherche la disponibilité de services en envoyant un message HTTP multicast à l'adresse « HOST : 239.255.255.250:1900 », avec « MX » le dispositif doit attendre un temps aléatoire entre 0 et MX secondes pour envoyer la réponse, en utilisant le protocole « MAN : SSDP :discover » et cherchant un service de type « ST », pour plus de détails voir la section 3.3.2.2.

Dans ce qui suit nous allons exposer la recherche de service dans UPnP pour réseau ad-hoc adaptée dans notre travail.

4.3.4 Message de recherche dans UPnP pour réseau ad-hoc

Dans UPnP pour réseau ad-hoc, le message de recherche se fait en diffusion (*broadcast*) et non en multicast, comme c'était le cas en UPnP version SSDP. C'est SEDIRAN qui se charge de faire cette recherche d'un service ou d'un type de service, ou d'un type racine.

Le message de recherche ne comportera plus les variables suivantes : « M-SEARCH : fonction de recherche », « MX : Le dispositif doit attendre un temps aléatoire entre 0 et MX secondes pour envoyer la réponse afin de ne pas congestionner le réseau si plusieurs dispositifs répondent au même moment » et « Host: 239.255.255.250 :1900 : adresse multicast » car ils ne se trouvent pas dans SEDIRAN qui fait aussi ses recherches en diffusion (*broadcast*).

Donc, le message de recherche de UPnP, c'est DREQ modifié de SEDIRAN qui contenait avant la modification les champs : *Type*, *Hop-Limit*, *ServiceCount*, *Adresse IP Source*, *ServiceTypeID*, *ServiceID*[1],..., *ServiceID[ServiceCount]* (voir Figure 4.7).

Nous avons fait les correspondances entre les champs du message de recherche DREQ et ceux de UPnP, sauf pour ceux qui sont spécifiques à SEDIRAN dont « Adresse IP Source », « Hop: *nombre de sauts* » et « ServiceCount: *nombre de services découverts* ».

Correspondances entre les champs équivalents

Type ~ MAN
ServiceTypeID ~ ST
ServiceID [1] ~ USN [1]
...
ServiceID [ServiceCount] ~ USN [ServiceCount]

« Hop » est le nombre de sauts maximal qu'un message est autorisé à effectuer. « ServiceCount » est le nombre de services déjà découverts et « USN [1] »,..., « USN [ServiceCount] » sont les noms unique des services ou dispositifs en utilisant UUID (*Universally Unique Identifier*) représentant les services déjà découverts (voir Figure 4.8).

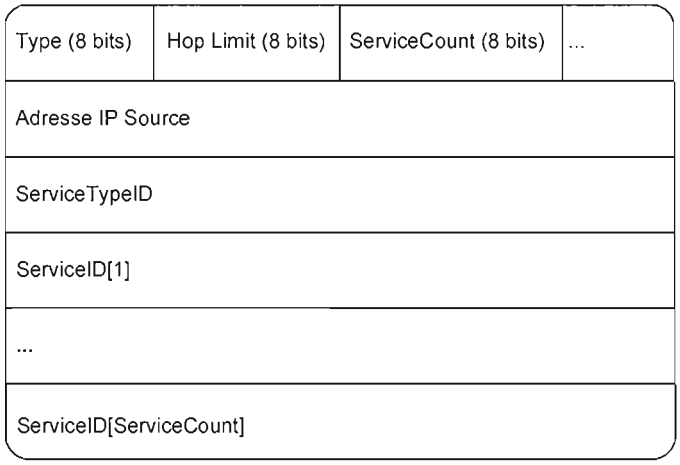

Figure 4.7 - Message de découverte DREQ de SEDIRAN

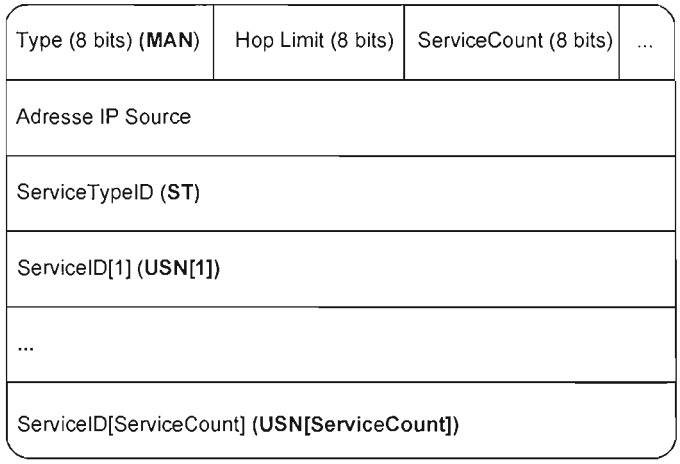

Figure 4.8 - Message de découverte DREQ de SEDIRAN modifié

4.3.5 Message de réponse dans UPnP

Si l'en-tête (ST) du message de recherche « ssdp:all » ou « upnp:root device » ou « uuid:UUID » concorde exactement avec le dispositif ou le préfixe du dispositif ou le type du service supporté par le dispositif que le point de contrôle veut rechercher, alors le dispositif répond avec un message unicast au point de contrôle.

Le dispositif doit attendre un temps aléatoire entre 0 et MX secondes pour envoyer la réponse. MX a déjà été envoyé par le point de contrôle dans l'en-tête de son message.

Le MX ne doit pas dépasser 120 secondes. Autrement, 120 secondes seront considérées. Si MX n'existe pas dans le message de recherche, alors le dispositif ou le service concerné ignore ce message.

Le message de réponse unicast à une recherche de disponibilité d'un service, le type du service recherchée se trouve dans « ST », la durée de disponibilité du service est dans « CACHE-CONTROL », avec un identificateur unique qu'on trouve dans « USN », la description du service se trouve à l'URL dans « LOCATION », « SERVER » détermine la version du protocole UPnP, le nom et la version du service, la date de la génération de la réponse est dans l'en-tête « DATE », pour plus de détails voir la sous-section 3.3.2.3.

Dans ce qui suit nous allons exposer la réponse dans UPnP pour réseau ad-hoc adaptée dans notre travail.

4.3.6 Message de réponse dans UPnP pour réseau ad-hoc

Dans UPnP pour réseau ad-hoc, le message de réponse se fait en unicast comme c'était le cas en UPnP version SSDP. C'est SEDIRAN qui se charge d'envoyer cette réponse de disponibilité d'un service ou d'un type de service ou d'un type racine.

Le message de réponse ne comportera plus les variables suivantes : « DATE : date de génération du message de réponse » et « EXT : confirme que l'en-tête MAN est comprise ».

Donc, le message de réponse de UPnP est le message DREP de SEDIRAN. qui contenait avant la modification les champs : Type, ServiceCount, Adresse IP Répondeur, Adresse IP Source, ServiceID[1], ServiceType[1] et Lifetime[1],...,ServiceID[ServiceCount], ServiceType[ServiceCount] et Lifetime[ServiceCount] (voir Figure 4.9).

Nous avons adapté le message de réponse DREP en ajoutant deux champs : la liste « SERVER[1],...,SERVER[ServiceCount] », chaque champ « SERVER » correspondant à un service cité dans le message de réponse et une autre liste « LOCATION[1],..., LOCATION[ServiceCount] », chaque champ « LOCATION » correspondant à un service cité dans le message de réponse. Pour les autres champs, nous avons fait les correspondances

entre ceux du message de réponse DREP et ceux de UPnP, sauf pour « Type : *type du message* DREP, « Adresse IP Source: *adresse IP source* », « Adresse IP Répondeur » et « ServiceCount: *nombre de services découverts* » qui sont spécifiques au SEDIRAN (voir Figure 4.10).

Correspondances entre les champs équivalents

ServiceID[1] ~ **USN [1]**
ServiceType[1] ~ **ST[1]**
LifeTime[1] ~ **Cache Control[1]**
...
ServiceID[ServiceCount] ~ **USN [ServiceCount]**
ServiceType[ServiceCount] ~ **ST[ServiceCount]**
LifeTime[ServiceCount] ~ **Cache Control[ServiceCount]**

« ServiceCount » est le nombre de services déjà découverts et « ServiceID ~ USN » est le nom unique du service ou dispositif en utilisant UUID représentant les services découverts, le « ServiceType ~ ST » est le type de service découvert et le « LifeTime ~ CacheControl » est la durée d'expiration du service.

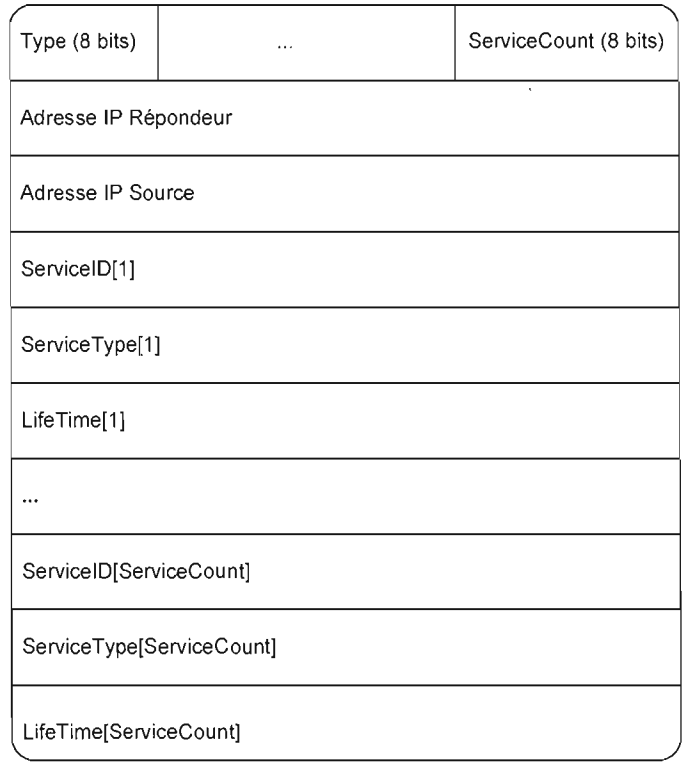

Figure 4.9 - Message de réponse DREP de SEDIRAN

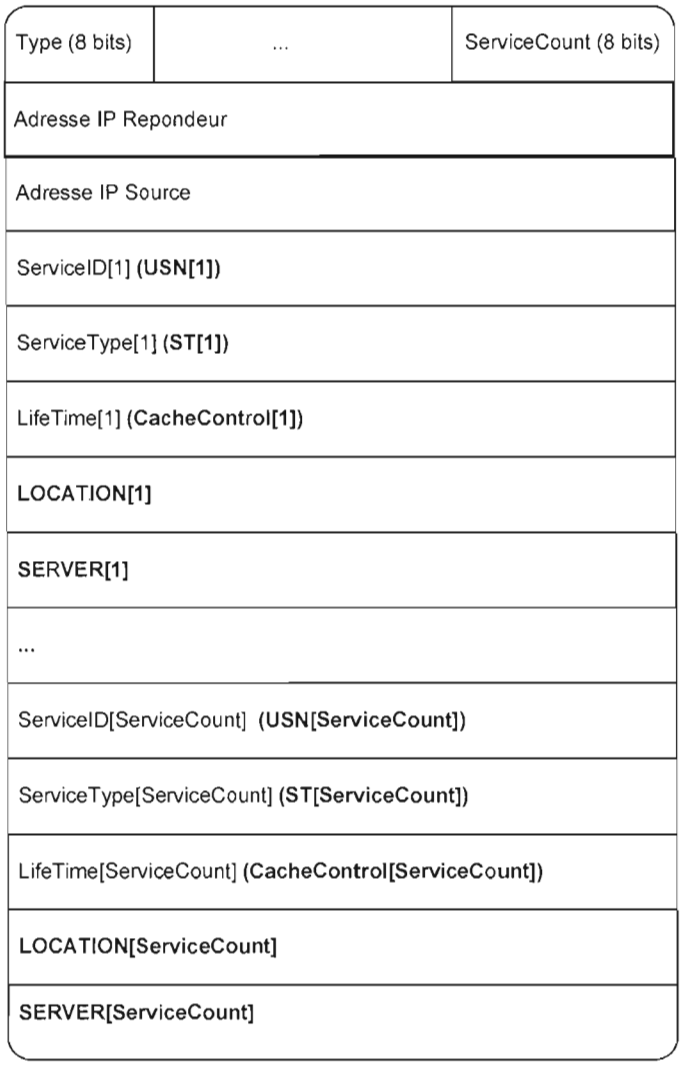

Figure 4.10 - Message de réponse DREP de SEDIRAN modifié

4.4 Principe de fonctionnement

Le client fournit une application client ou serveur qui s'exécute au-dessus du SDK. L'application client ou serveur implémente des fonctionnalités pour des services spécifiques.

Les APIs SDK cachent les détails des protocoles UPnP par rapport aux applications clients ou serveurs et leur fournissent l'accès aux fonctionnalités via une interface unique. Il y a un seul client et un seul serveur qui peuvent être assignés à un même processus.

C'est-à-dire qu'une application peut s'enregistrer au plus une fois comme client et une fois comme serveur.

L'API de UPnP maintient une table de références des clients et des serveurs enregistrés avec SDK. Donc, pour chaque appel à une fonction API, le SDK valide la référence connue en vérifiant la table de références.

4.4.1 Messages d'annonces

Un terminal (serveur) qui vient de joindre un réseau ad-hoc doit tout d'abord construire, pour chacun de ses services et de ses serveurs encapsulés, des documents de descriptions contenant des informations détaillées sur les serveurs et les services encapsulés et leurs vendeurs, ainsi que faire une vérification dans la table des références des serveurs enregistrés avec SDK afin de valider le terminal (serveur) qui annoncera ses services.

Le terminal (serveur) diffuse un nombre de messages d'annonces égal au nombre de ses serveurs et services encapsulés. Pour la première couche « *UPnP Vendor* », le message d'annonce contient de l'information spécifique au vendeur, comme l'identificateur du serveur (ou service) et l'URL de son document de description. Pour la deuxième couche « *UPnP Forum Working Committee* », il y a d'autres informations qui sont ajoutées au message, comme le type du serveur. Ces informations sont traitées par des APIs serveur d'UPnP (voir Figure 4.11).

SEDIRAN se chargera de diffuser les messages ADVM aux nœuds du réseau ad-hoc pour annoncer la présence de services spéciaux. Le message ADVM passe au protocole de routage AODV qui l'encapsulera dans un paquet RREP et puis le diffusera sur le réseau (voir Figure 4.15) [22]. Lorsqu'un client reçoit le message, le protocole de routage AODV doit vérifier à l'aide du champ « *RREP ID* » si le message a été déjà reçu. Sinon, AODV met à jour la route vers le serveur (fournisseur), rediffuse le message « ADVM » et, ensuite, le passe au protocole SEDIRAN pour mettre à jour le cache des services spéciaux [22].

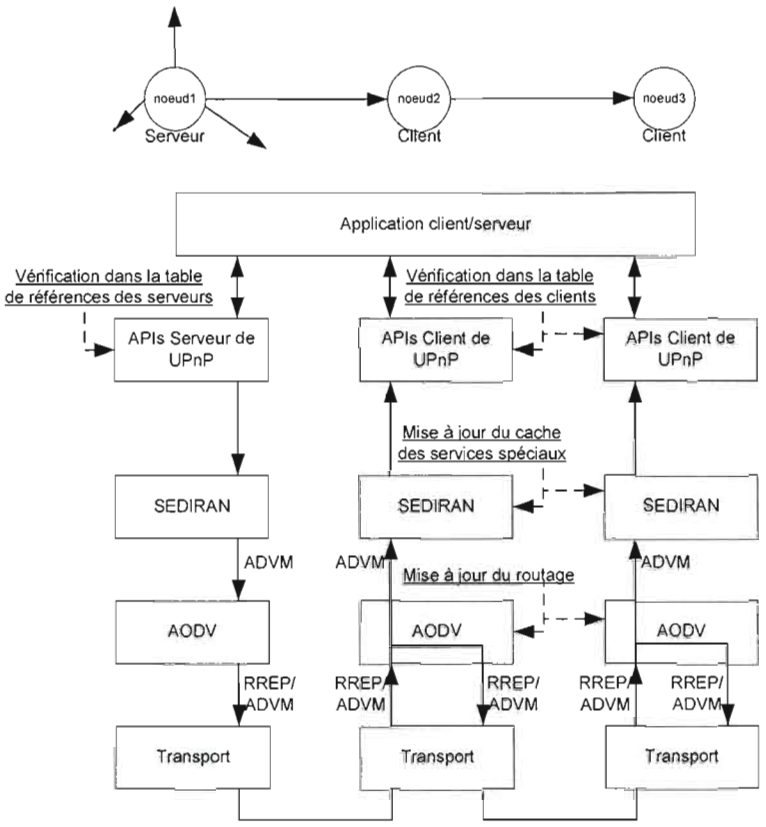

Figure 4.11 - Diffusion des messages d'annonces ADVM

4.4.2 Message de recherche

Un client qui vient de joindre un réseau ad-hoc sera validé par les APIs clients qui se trouvent dans le SDK en faisant la vérification dans la table des références des clients. Après, il lance un message de recherche à l'aide du protocole de découverte de services d'UPnP pour réseau ad-hoc, qui est SEDIRAN. Celui-ci diffuse le message de recherche « *DREQ* » sur le réseau ad-hoc.

Pour la première couche « *UPnP Vendor* », le message de recherche contient de l'information spécifique au vendeur, comme l'identificateur du serveur (dispositif ou service). Pour la deuxième couche « UPnP Forum Working Committee », il y a d'autres informations qui sont ajoutées au message, comme le type du serveur (dispositif ou service). Ces informations sont traitées par les APIs clients UPnP (voir Figure 4.12).

SEDIRAN se chargera de diffuser le message DREQ aux nœuds du réseau ad-hoc pour rechercher des services ordinaires. Donc, le message passe au protocole de routage AODV qui l'encapsulera dans un paquet RREP et, le diffusera sur le réseau (voir Figure 4.12) [22].

Lorsqu'un serveur (dispositif) reçoit le message, le protocole de routage AODV met à jour la route vers le demandeur pour préparer le chemin de retour de la réponse. Ce serveur (dispositif) vérifie la valeur du champ « *Hop-Limit* ». S'il est la cible de la requête de recherche, alors il passe le message DREQ à la couche supérieure, qui est le protocole SEDIRAN, qui chargera dans la table des services locaux les services recherchés (voir Figure 4.12) [22].

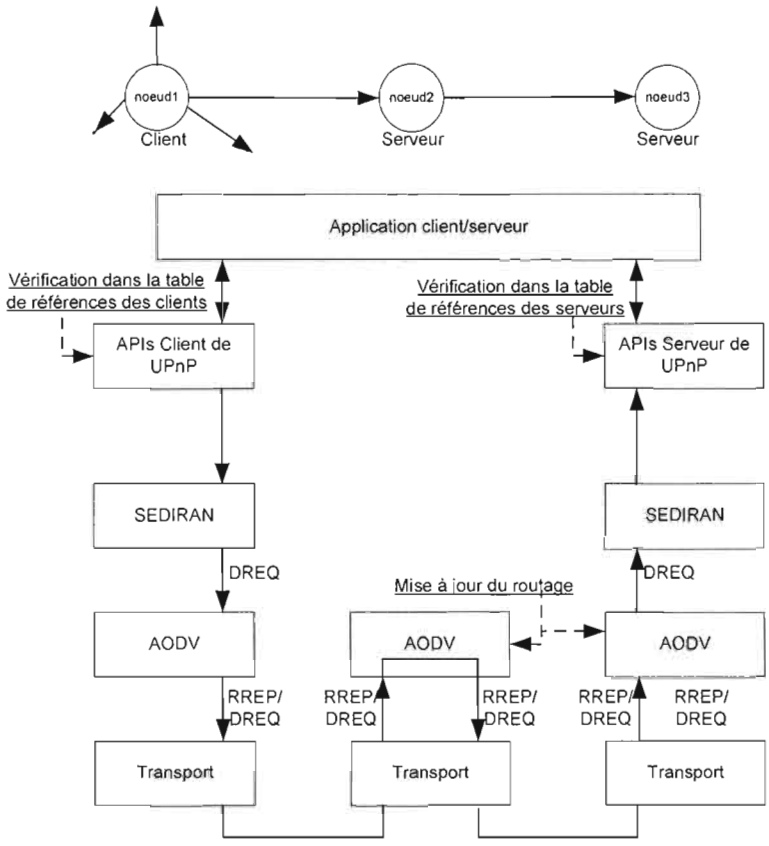

Figure 4.12 - Diffusion des messages de recherche DREQ

4.4.3 Message de réponse

Le serveur (fournisseur de services) sera validé par les APIs serveurs en faisant la vérification dans la table des références des serveurs. Après cela, il envoie un message de réponse en unicast DREP au client (demandeur de services). Pour la première couche « *UPnP Vendor* », le message de réponse contient de l'information spécifique au vendeur, comme l'identificateur du serveur (dispositif ou service) et l'URL de son document de description. Pour la deuxième couche « *UPnP Forum Working Committee* », il y a d'autres informations qui sont ajoutées au message, comme le type du serveur. Ces informations sont traitées par les APIs serveurs UPnP (voir Figure 4.13).

SEDIRAN se chargera d'envoyer le message DREP au nœud demandeur. Ce message DREP passe au protocole de routage AODV qui l'encapsulera dans un paquet RREP et puis l'enverra au client qui a activé la requête d'annonce. Lorsqu'un nœud intermédiaire reçoit le message, le protocole de routage AODV met à jour la route vers le fournisseur et passe le message DREP au protocole SEDIRAN pour mettre à jour le cache des services ordinaires (voir Figure 4.13) [22].

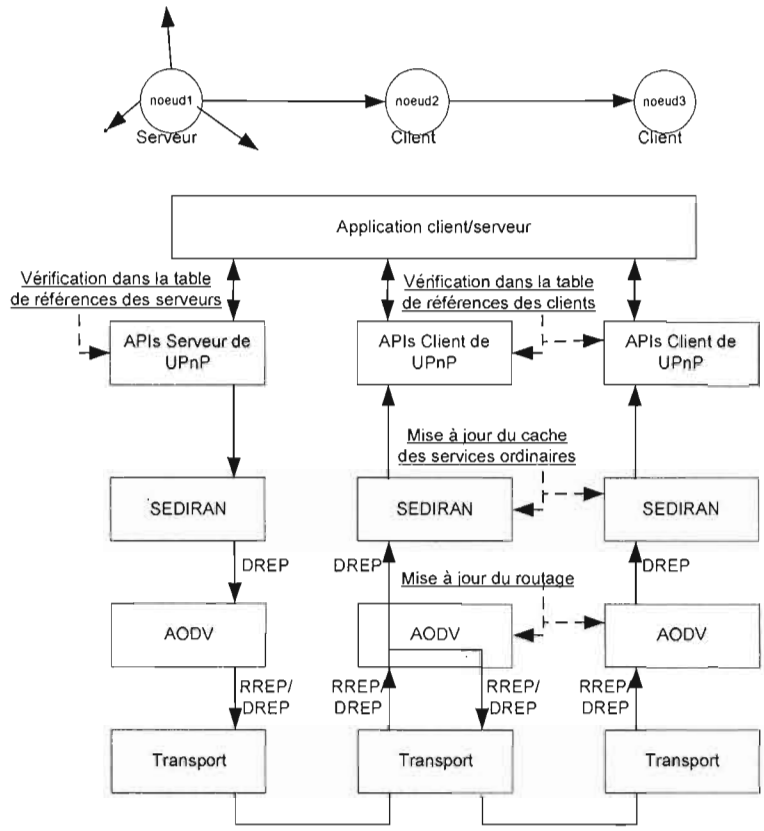

Figure 4.13 - Message de réponse DREP au message de recherche DREQ

CHAPITRE V

MISE EN ŒUVRE

5.1 Introduction

Nous avons implémenté le protocole de découverte de services dans les réseaux adhoc « UPnP pour réseaux Ad-hoc » en faisant une simulation d'un prototype dont nous avons testé le fonctionnement.

Donc, nous avons fait des modifications nécessaires au niveau du protocole SEDIRAN pour interagir avec les APIs de UPnP. Vu qu'une version complète du code de UPnP n'est pas disponible, nous avons préféré passer manuellement les messages vers le SEDIRAN déjà modifié.

5.2 Architecture logicielle de UPnP pour réseau Ad-hoc

L'architecture logicielle de UPnP pour réseau Ad-hoc peut être réalisée en utilisant l'architecture de SDK pour UPnP [26], cette dernière possède le module SSDP que nous avons remplacé par Jsediran le module représentant l'architecture logicielle de Sediran [23].

Les détails du module Jsediran sont résumés dans les APIs SDK pour UPnP pour réseau adhoc (voir Figure 5.1 et 5.2).

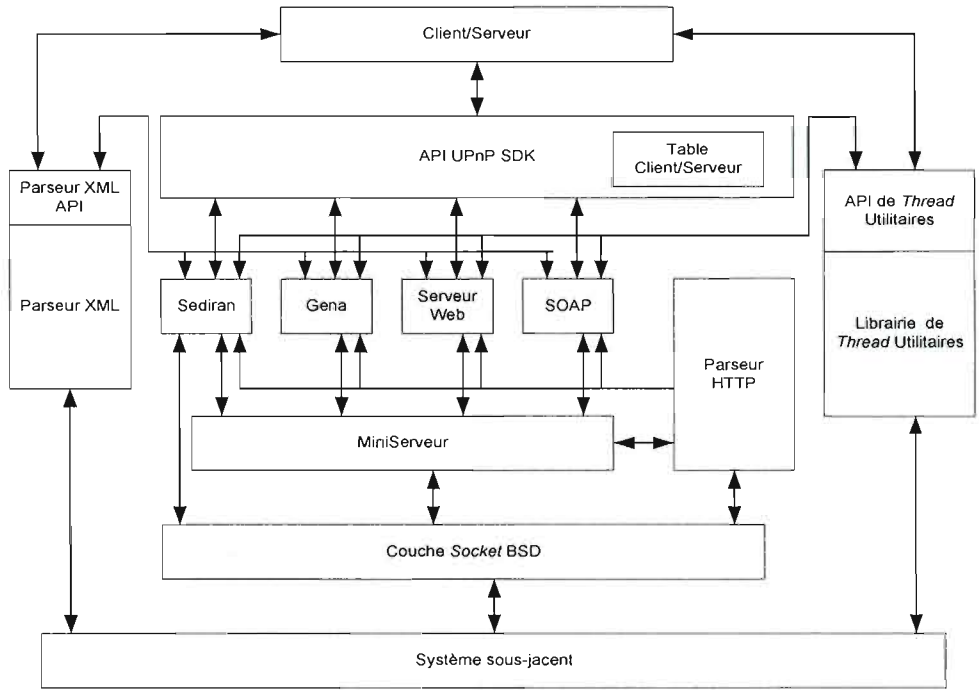

Figure 5.1 - Architecture logicielle de UPnP pour réseaux Ad-hoc [26]

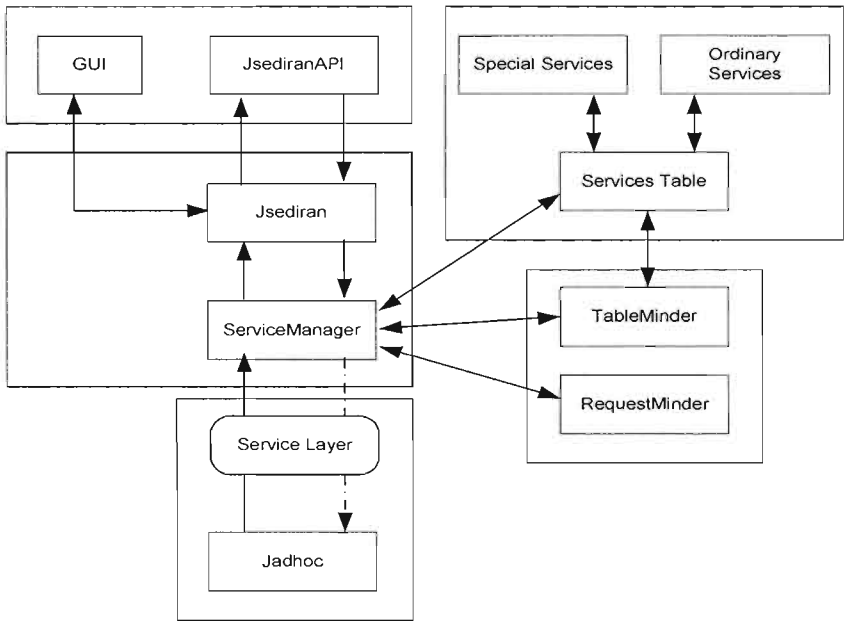

Figure 5.2 - Architecture logicielle de Sediran [23]

5.3 Enregistrer et annoncer un dispositif

Il existe de nombreuses façons d'implémenter un dispositif UPnP en utilisant le SDK pour dispositifs UPnP. Toutefois, toute implémentation doit effectuer quelques étapes de base:

1. Mettre en place et initialiser le dispositif en suivant ces étapes de base:
   a. Initialiser le SDK en utilisant UpnpInit().
   b. Définir un répertoire racine pour le mini serveur Web à l'aide de UpnpSetWebServerRootDir().
   c. Enregistrez le dispositif de description de document en utilisant UpnpRegisterRootDevice() ou UpnpRegisterRootDevice2().
   d. Démarrer l'initialisation d'un dispositif spécifique.
   e. Annoncer le dispositif sur le réseau en utilisant UpnpSendAdvertisement().
      1. La méthode TestADVM() de l'objet StartTest de Jsediran est exécuté pour annoncer le dispositif.
      2. L'annonce d'un service se fait dans un environnement adhoc à l'aide de pktSender.sendMessage(rrep) de l'objet PacketSender crée par l'objet RouteManager() tous appartenant à l'application Jadhoc.
2. Le dispositif doit gérer trois différents types de demandes:
   a. Demande d'abonnement aux notifications des modifications des états de service.
   b. Les demandes de récupérer la valeur actuelle d'une variable d'état de service.
   c. Les demandes de modifier la valeur d'une variable d'état de service.
3. Garder les points de contrôle mis à jour par l'envoi d'événements en utilisant UpnpNotify() ou UpnpNotifyExt().
4. Désactiver en utilisant le SDK UpnpFinish().

5.4 Enregistrer un point de contrôle UPnP et rechercher un dispositif

Le kit de développement Intel pour les dispositifs UPnP traite aussi l'application cliente qui est le point de contrôle UPnP. Les étapes de base pour implémenter une application point de contrôle sont les suivantes:

1. Mettre en place et initialiser le point de contrôle suivant ces étapes de base:
   a. Initialiser le SDK en utilisant UpnpInit().

b. Enregistrer un point de contrôle (également connu sous le nom d'un client) en utilisant la fonction UpnpRegisterClient().

2. Rechercher un dispositif en utilisant UpnpSearchAsync
   a. La méthode TestDREQ() de l'objet StartTest de Jsediran est exécutée pour rechercher le dispositif.
   b. La recherche se fait dans un environnement adhoc à l'aide de pktSender.sendMessage(rrep) de l'objet PacketSender crée par l'objet RouteManager() tous appartenant à l'application Jadhoc.
3. Télécharger la description des documents en utilisant UpnpDownloadXmlDoc() ou UpnpHttp().
4. S'abonner aux services à l'aide de UpnpSubscribe() ou UpnpSubscribeAsync().
5. Un point de contrôle utilise UpnpSendAction() ou UpnpSendActionAsync() pour connaître les changements d'états d'un service à qui il est abonné.
6. Désactiver le point de contrôle suivant les étapes ci-dessous:
   a. Désinscrire le point de contrôle en utilisant le SDK UpnpUnRegisterClient().
   b. Désactiver en utilisant le SDK UpnpFinish().

5.5 Modifications faites au SEDIRAN

Dans cette section, nous présentons les quelques classes et méthodes qui ont été modifiées pour interagir avec les APIs de UPnP. Le langage de programmation java est utilisé.

5.5.1 Classe Service

Cette classe représente un service ordinaire modifié connu par les nœuds du réseau. Les variables « serviceLocation » et « serviceServer » sont ajoutées dans la classe Service afin de permettre leur utilisation dans les phases suivantes de UPnP.

```
public class Service {
public  UUID serviceID;
public  UUID serviceTypeID;
public  int lifeTime;
public  char[]  serviceLocation = new char [ConfigInfo.DescSize]; // ajout H.Boukouna
```

```
public  char[]  serviceServer = new char [ConfigInfo.ServerSize]; // ajout H.Boukouna
public Service(UUID sID, UUID sTID, int lTime, char[] sLocation, char [] sServer) {
      serviceID = sID;
      serviceTypeID = sTID;
      lifeTime = lTime;
      serviceLocation = sLocation; // ajout Hakim Boukouna
      serviceServer = sServer; // ajout Hakim Boukouna
}
public String toString(){
      String str;
      str = " serviceID " + serviceID +
      ", serviceTypeID :" + serviceTypeID +
      ", lifeTime " + lifeTime;
      //auteur Boukouna Hakim     ajout de SERVER et NT
      str = str + " serviceLocation :";
      for(int i=0;i<ConfigInfo.DescSize;i++)
        str = str + serviceLocation[i];
      str = str + " SERVER :";
      for(int i=0;i<ConfigInfo.ServerSize;i++)
        str = str + serviceServer[i];
      // fin ajout
      return str;
}
}
```

Figure 5.3 - Classe Service représentant un service ordinaire modifié

5.5.2 Classe SpecialeService

Cette classe représente un service spécial non connu des nœuds du réseau. Les variables « notificationType » et « serviceServer » sont ajoutées dans la classe Service afin de permettre leur utilisation dans les phases suivantes de UPnP.

```
public class SpecialService {
    ...
    //author Boukouna Hakim    ajout de la variable SERVER et NT
    public  char[]  serviceServer = new char [ConfigInfo.ServerSize];
    public  char[]  notificationType = new char [ConfigInfo.NTSize];
    // fin ajout
    /** Creates a new instance of SpecialService */
public SpecialService(UUID sID, char[] sDesc, int lTime, char [] sServer, char [] nType) {
    serviceID = sID;
    serviceDescription = sDesc;
    lifeTime = lTime;
    serviceServer = sServer; //ajout Hakim Boukouna
    notificationType = nType;//ajout Hakim Boukouna
}
public String toString(){
    String str;
    str = " service ID :" + serviceID;
    str = str + " service Location :";
    for(int i=0;i<ConfigInfo.DescSize;i++)
      str = str + serviceDescription[i];
    str = str + " life Time :" + lifeTime;
    //auteur Boukouna Hakim      ajout de SERVER et NT
    str = str + " SERVER :";
    for(int i=0;i<ConfigInfo.ServerSize;i++)
      str = str + serviceServer[i];
    str = str + " NT :";
    for(int i=0;i<ConfigInfo.NTSize;i++)
```

```
        str = str + notificationType[i];
//fin ajout
    return str;
}}
```

Figure 5.4 - Classe représentant un service spécial non connu des nœuds du réseau

5.5.3 Classe StarTest

La classe StarTest est crée pour tester le protocole SEDIRAN, on introduit manuellement l'URL du fichier de description et la version du système ainsi que le type du service en utilisant les variables « sLocation », « sServer » et « sNotification ».

```
public class startTest {
public startTest() { }
public   static void TestDREQ() throws Exception { ... }
public   static void TestDREP() throws Exception {
    ...
    char[] sLocation = new char [ConfigInfo.DescSize]; // ajout Hakim Boukouna
    char[] sServer = new char [ConfigInfo.ServerSize]; // ajout Hakim Boukouna
    ...
    String sL = "http://coffemaker1.mykitchen.de/description"; // ajout H. Boukouna
    String sS = "KOS/0.1 UPnP/1.0 CoffeMaker/6.0"; // ajout Hakim Boukouna
    sLocation = sL.toCharArray(); // ajout Hakim Boukouna
    sServer = sS.toCharArray(); // ajout Hakim Boukouna
    ...
    s[0] = new Service(sID,sTID,lT,sLocation,sServer);
    ...
    s[1] = new Service(sID,sTID,lT,sLocation,sServer);
    ...
}
public   static void TestADVM() throws Exception {
    ...
```

```
    char[] sServer = new char[ConfigInfo.ServerSize];
    char[] sNotification = new char[ConfigInfo.NTSize];
    char[]  description= new char[ConfigInfo.DescSize];
    String sS = "KOS/0.1 UPnP/1.0 CoffeMaker/6.0"; // ajout Hakim Boukouna
    String sN = "ka:coffe-maker"; // ajout Hakim Boukouna
    String  sL = "http://coffemaker1.mykitchen.de/description"; // ajout H. Boukouna
    description = sL.toCharArray(); // ajout Hakim Boukouna
    sNotification = sN.toCharArray(); // ajout Hakim Boukouna
    sServer = sS.toCharArray(); // ajout Hakim Boukouna
    ...
    specialS = new SpecialService(sID,description,IT,sServer,sNotification);
    ...
}
public static void main(String[] args) throws Exception{
    TestDREP(); // ou TestDREQ() ou TestADVM();   }
}
```

Figure 5.5 - Classe StarTest est crée pour tester le protocole SEDIRAN

5.5.4 Classe ADVM

La classe représente le message ADVM (*Advertissement Message*) qu'un nœud diffuse (*broadcast*) pour annoncer un service spécial aux nœuds du réseau ad-hoc.

public class ADVM extends SDIRAMessage{

Ce constructeur permet de récupérer les champs d'un message ADVM à partir d'un datagramme.

```
public ADVM(DatagramPacket up) throws Exception {
    super(up);
    breakDgramToADVM();
}
```

Ce constructeur permet de construire un datagramme ADVM

```
public ADVM(InetAddress sIPAddr, SpecialService sSrv)  throws Exception {
    sourceIPAddr=sIPAddr;
    specialSrv =sSrv;
    buildADVM();
}
private void breakDgramToADVM() throws Exception {
    ...
    char[] srvDesc = new char[ConfigInfo.DescSize];
    char[] serverDesc = new char[ConfigInfo.ServerSize];
    char[] ntDesc = new char[ConfigInfo.NTSize];
    ...
    //auteur Boukouna Hakim     ajout de SERVER et NT
    k=k+4;
    for(int i=0; i< ConfigInfo.ServerSize; i++){
      serverDesc[i] = 0;
      int u=0;
      u |= (int) ((buf[k+2*i] << 8) & 0x0000FF00) ;
      u |= (int) (buf[k+2*i+1] & 0x000000FF) ;
      serverDesc[i] = (char) u;
    }
    k = k+ConfigInfo.ServerSize*2;
    for(int i=0; i< ConfigInfo.NTSize; i++){
      ntDesc[i] = 0;
      int u=0;
      u |= (int) ((buf[k+2*i] << 8) & 0x0000FF00) ;
      u |= (int) (buf[k+2*i+1] & 0x000000FF) ;
      ntDesc[i] = (char) u;
    }  //fin ajout
    specialSrv = new SpecialService(sID,srvDesc,lTime,serverDesc,ntDesc);
}
private void  buildADVM() throws Exception {
    ...
```

```
int packetSize= 28+ConfigInfo.DescSize*2+ConfigInfo.ServerSize*2+
                ConfigInfo.NTSize*2;
...
//auteur Boukouna Hakim     ajout de SERVER et NT
k=k+4;
for(int i=0; i<ConfigInfo.ServerSize; i++) {
  msg[k+2*i] = msg[k+2*i+1] = (byte) 0;
  msg[k+2*i] |= (byte) (( (int)specialSrv.serviceServer[i] & 0x0000FF00) >>> 8);
  msg[k+2*i+1] |= (byte) (  (int)specialSrv.serviceServer[i] & 0x000000FF);
}
k = k+ConfigInfo.ServerSize*2;
for(int i=0; i<ConfigInfo.NTSize; i++) {
  msg[k+2*i] = msg[k+2*i+1] = (byte) 0;
  msg[k+2*i] |= (byte) (( (int)specialSrv.notificationType[i] & 0x0000FF00) >>> 8);
  msg[k+2*i+1] |= (byte) (  (int)specialSrv.notificationType[i] & 0x000000FF);
}   // fin ajout
javaUDPDgram = new DatagramPacket(msg, msg.length,
                InetAddress.getByName(cfg.BroadcastAddress), SDIRA_PORT);
}
}
```

Figure 5.6 - Classe représentant le message ADVM

5.5.5 Classe DREP

La classe représente le message DREP (Discovery REPly) qui répond à une requête de découverte de services.

public class DREP extends SDIRAMessage{

Ce constructeur permet de récupérer les champs d'un message DREP à partir d'un datagramme.

public DREP(DatagramPacket up) throws Exception {

```
super(up);
```

```
    breakDgramToDREP();
}
```

Ce constructeur permet de construire un datagramme DREP

```
public DREP(InetAddress rIPAddr, InetAddress orgIPAddr,  Service[] srvs)
                    throws Exception {
        serviceCount = (byte)srvs.length;
        responderIPAddr=rIPAddr;
        originatorIPAddr=orgIPAddr;
        services = srvs;
        buildDREP();
    }
private void  breakDgramToDREP() throws Exception {
    ...
    char[] serviceLocation = new char[ConfigInfo.DescSize];
    char[] serverDesc = new char[ConfigInfo.ServerSize];
    for (int i=0 ; i < serviceCount ; i++){
      k=12+(36+ConfigInfo.DescSize*2+ConfigInfo.ServerSize*2)*i;
      //36+DescSize*2+ServerSize*2 est le nombre d'octets dans l'objet service
      ...
      // auteur Boukouna Hakim      ajout de Location et Server
      k=k+36;
      for(int j=0; j< ConfigInfo.DescSize; j++){
        serviceLocation[j] = 0;
        int u=0;
        u |= (int) ((buf[k+2*j] << 8) & 0x0000FF00) ;
        u |= (int) (buf[k+2*j+1] & 0x000000FF) ;
        serviceLocation[j] = (char) u;
      }
      k = k+ConfigInfo.DescSize*2;
      for(int j=0; j< ConfigInfo.ServerSize; j++){
        serverDesc[j] = 0;
```

```
            int u=0;
            u |= (int) ((buf[k+2*j] << 8) & 0x0000FF00) ;
            u |= (int) (buf[k+2*j+1] & 0x000000FF) ;
            serverDesc[j] = (char) u;
          }    // fin ajout
          services[i] = new Service(sID,sTID,lTime,serviceLocation,serverDesc);
} }
private void  buildDREP() throws Exception {
      ...
    int packetSize= 12+serviceCount*(36+ConfigInfo.DescSize*2+ConfigInfo.ServerSize*2);
      // auteur Boukouna Hakim     ajout de Location et Server
      k = k+36;
      for(int j=0; j<ConfigInfo.DescSize; j++) {
      msg[k+2*j] = msg[k+2*j+1] = (byte) 0;
      msg[k+2*j] |= (byte) (( (int)services[i].serviceLocation[j] & 0x0000FF00) >>> 8);
      msg[k+2*j+1] |= (byte) (  (int)services[i].serviceLocation[j] & 0x000000FF);
      }
      k = k+ConfigInfo.DescSize*2;
      for(int j=0; j<ConfigInfo.ServerSize; j++) {
      msg[k+2*j] = msg[k+2*j+1] = (byte) 0;
      msg[k+2*j] |= (byte) (( (int)services[i].serviceServer[j] & 0x0000FF00) >>> 8);
      msg[k+2*j+1] |= (byte) (  (int)services[i].serviceServer[j] & 0x000000FF);
      }    // fin ajout
      javaUDPDgram = new DatagramPacket(msg, msg.length,
                              InetAddress.getByName(cfg.BroadcastAddress), SDIRA_PORT);
      }}  ...}
```

Figure 5.7 - Classe représentant le message DREP

5.6 Tests de SEDIRAN modifié

Pour visualiser le comportement et le bon fonctionnement du prototype, SEDIRAN contient un journal qui permet de garder les traces des exécutions. Nous avons ajouté trois services ordinaires locaux et puis nous avons lancé l'application (voir Figure 5.3).

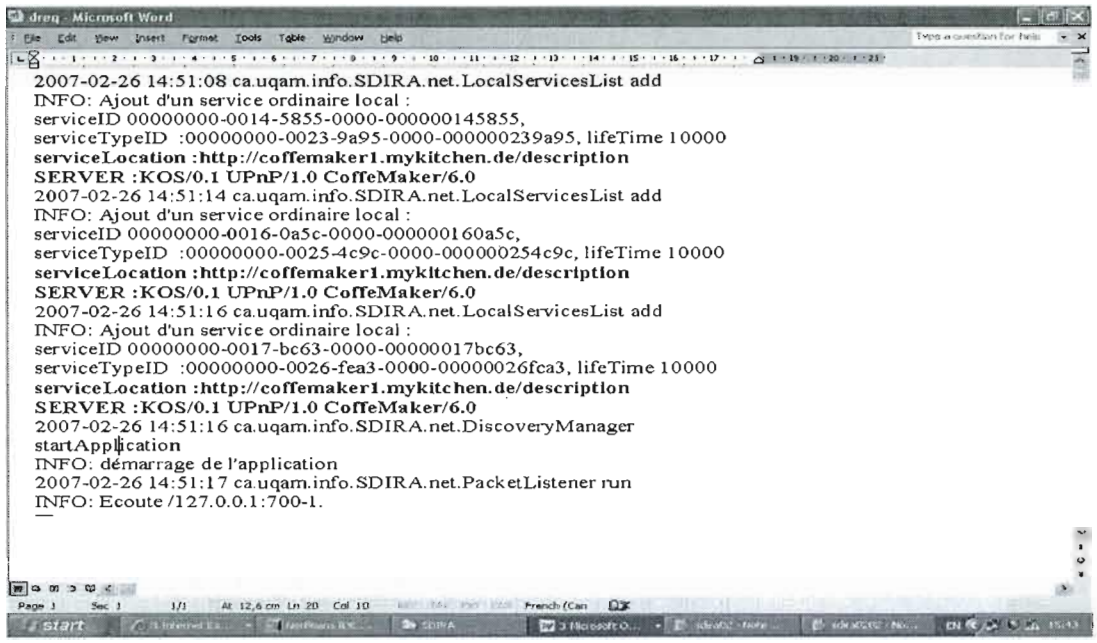

Figure 5.8 - Une capture qui montre le journal de SEDIRAN

5.6.1 Test de DREQ modifié

Nous avons utilisé le programme client pour envoyer une requête de recherche DREQ. Le journal montre que le processus SEDIRAN a bien reçu le message DREQ recherchant un service de type 00000000-0026-fea3-0000-00000026fea3.

Après l'assortiment de la requête reçue avec la liste des services locaux, un service a été retrouvé. Il a le ID=00000000-0000-0017-bc63-0000-00000017bc63, avec un fichier de description localisé à l'adresse « http://coffemaker1.mykitchen.de/description » et la version de son système et produit est « KOS/0.1 UPNP/1.0 CoffeMaker/6 ». Ces informations seront passées aux APIs de UPnP et seront très utiles aux phases restantes de UPnP (voir Figure 5.4).

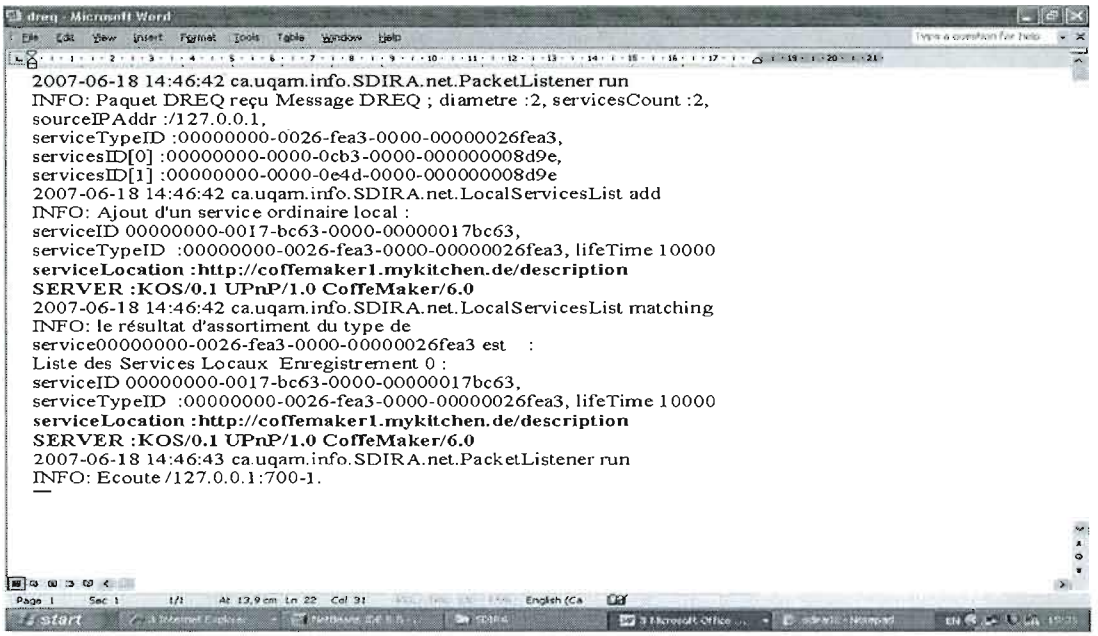

Figure 5.9 - le processus SEDIRAN recevant le message DREQ

5.6.2 Test du DREP modifié

Nous avons utilisé le programme client pour envoyer un message DREP modifié. Le journal montre que le processus SEDIRAN a bien reçu le message DREP modifié, ayant deux services, avec ID=00000000-0014-5855-0000-000000145855 et 00000000-0016-0a5c-0000-000000160a5c répondant à une requête de recherche de type 00000000-0023-9a95-0000-000000239a95. Chacun de ces deux services trouvés a un fichier de description localisé à l'adresse « http://coffemaker1.mykitchen.de/description ». La version de leur système et produit est « KOS/0.1 UPNP/1.0 CoffeMaker/6 ». Ces informations seront passées aux APIs de UPnP et seront très utiles aux phases restantes de UPnP (voir Figure 5.5).

Figure 5.10 - le processus SEDIRAN recevant le message DREP modifié

5.6.3 Test d'ADVM modifié

Nous avons utilisé le programme client pour envoyer un message ADVM modifié. Le journal montre que le processus SEDIRAN a bien reçu le message ADVM modifié annonçant

un service avec ID=00000000-001e-847f-0000-000001e847f. Son fichier de description est localisé à l'adresse « http://coffemaker1.mykitchen.de/description », son type est « ka:coffe-maker » et la version de son système et produit est « KOS/0.1 UPNP/1.0 CoffeMaker/6 ». Ces informations seront passées aux APIs de UPnP et seront très utiles aux phases restantes de UPnP (voir Figure 5.6).

Figure 5.11 - le processus SEDIRAN recevant le message ADVM modifié

CONCLUSION

Dans ce mémoire, nous avons proposé une nouvelle version du protocole de découverte de services UPnP qui devient désormais fonctionnel dans les réseaux ad-hoc.

UPnP pour réseau ad-hoc est destiné à la découverte et à l'interaction avec les services, qu'ils soient matériels ou logiciels, ce qui nous a conduit à opter pour des nominations plus générales comme type de services, sous type de services et services au lieu de dispositif racine, dispositif encapsulé et services.

Dans UPnP pour réseau ad-hoc, les fichiers de descriptions sont enregistrés localement dans les nœuds qui offrent les services correspondants.

L'architecture de UPnP pour réseau ad-hoc pourra utiliser les services web vu qu'ils partagent des protocoles communs comme XML, HTTP et SOAP. Donc, notre système pourra adopter le langage de description des services web et le UDDI. Ceci permettra de l'intégrer, par exemple, dans les applications du commerce mobile. Ce point important pourrait faire l'objet d'extensions futures de notre travail.

Dans ce travail, nous avons tenté de proposer une solution et une architecture qui permettent de mettre en œuvre une pile de protocoles basée sur UPnP et le protocole SEDIRAN. Ces deux protocoles offrent chacun des avantages :

- UPnP est un standard industriel généralisé qui permet de gérer des services et des équipements.
- SEDIRAN est un protocole qui est couplé avec les protocoles de routage ad-hoc.

Leur combinaison permet de construire un environnement riche pour la découverte des services de manière plus globale tel que décrit dans ce mémoire.

Nous avons également proposé les grandes lignes d'une mise en œuvre en Java en vue de montrer la faisabilité de notre solution. Du travail reste à faire, notamment, de montrer par un modèle de performance que notre solution est performante en terme de temps et nombre de messages échangés.

BIBLIOGRAPHIE

[1] E. Guttman, C. Perkins, *Service Location Protocol, Version 2*, RFC 2608 Sun Microsystems, juin 1999.

[2] *Jini™ Architecture Specification*, Sun Microsystems, Version 2.0, juin 2003.

[3] *Salutation Architecture Spécification (Part-1)*, Version 2.0c, The Salutation Consortium, juin 1999.

[4] T. Lemlouma, *Le routage dans les réseaux mobiles ad-hoc*, mémoire de Magistère Université des Sciences et de la Technologie Houari Boumèdiene Alger, Septembre 2000.

[5] James Kardach,, *Bluetooth Architecture Overview*, Mobile Computing Group, Intel Corporation, 2000.

[6] E. Corson, J. Macker, *Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations*, RFC2501 Université de Maryland, janvier 1999.

[7] S. Cheshire, *DNS-Based Service Discovery*, draft-cheshire-dnsext-dns-sd-2.txt, Apple Computer inc. février 2004.

[8] C. Bettstetter, C. Renner, *A Comparison of service discovery protocols and implementation of service location protocol*, Université Technique de Munich, 2000.

[9] Jidong Wu and Martina Zitterbart, « Service Awareness and its challenges in mobile ad hoc networks », Institute of Telematics, University of Karlsruhe, Germany.

[10] Charles E. Perkins, *Service Location Protocol for Mobile Users,* 1998.

[11] E. Guttman, *Service Location Protocol Automatic Discovery of IP Network Services*, Sun Microsystems, juillet-août 1999.

[12] *Jini Device Architecture Specification*, Sun Microsystems, Version 2.0, juin 2003

[13] *Découverte de données dans les réseaux mobiles*, Université Joseph Fourier, Grenoble, juin 2003.

[14] *Understanding Universal Plug and Play*, White Paper, Microsoft, Juin 2000.

[15] *UPnP Device Architecture 1.0*, UPnP forum, Version 1.0.1, mai 2003.

[16] N. Desai, *Konark – An Ad-Hoc Service Discovery Protocol,* mémoire de maitrise, Université de Floride, Aout 2002.

[17] D. Chakraborty, A. Joshi, Y. Yesha et Tim Finin, *GSD: A novel group-based service discovery protocol for MANETS,* Conférence IEEE sur les réseaux de communications Mobile et sans fil, Stockholm, Suède, Septembre 2002.

[18] D. Chakraborty, A. Joshi, Y. Yesha et Tim Finin, *Towards Distributed Service Discovery in Pervasive Computing Environment,* IEEE Transactions on Mobile Computing, Juillet 2004.

[19] F. Almenarez et C. Campo, *SPDP: A Secure Service Discovery Protocol for Ad-hoc Networks*, Dept. Telematic Engineering - Université Carlos III, Madrid Eunice 2003.

[20] C. Perkins, E. Belding-Royer, S. Das, *Ad-hoc On-Demand Distance Vector (AODV) Routing*, RFC3561 (Experimental) Juillet 2003.

[21] Lan Wu, *Service Discovery for Personal Networks*, Rapport de thèse, Université de Stuffgard, Décembre 2004.

[22] A. Khir, *Un Protocole de Découverte de Services dans les réseaux ad-hoc*, mémoire de Maîtrise, UQAM, 2005.

[23] H. Labiod, « Réseaux mobiles ad-hoc et réseaux de capteurs sans fil », chapitre 7, Hermès science, Lavoisier 2006.

[24] M. Jeronimo, J. Weast, *UPnP Design by Example*, Intel press, Avril 2003.

[25] D. -L. McGuinness, F. -V. Harmelen, *OWL Web Ontology Language Overview*, recommandation W3C, Février 2004.

[26] *Intel SDK for UPnP Devices Version 1.2.1*, Intel, Novembre 2002.