



Titre: Architecture de découverte des services web géolocalisés dans les réseaux mobiles de prochaine génération
Title:

Auteur: André Claude Bayomock Linwa
Author:

Date: 2005

Type: Mémoire ou thèse / Dissertation or Thesis

Référence: Bayomock Linwa, A. C. (2005). Architecture de découverte des services web géolocalisés dans les réseaux mobiles de prochaine génération [Thèse de doctorat, École Polytechnique de Montréal]. PolyPublie.
Citation: <https://publications.polymtl.ca/8193/>

 Document en libre accès dans PolyPublie
Open Access document in PolyPublie

URL de PolyPublie: <https://publications.polymtl.ca/8193/>
PolyPublie URL:

Directeurs de recherche: Samuel Pierre
Advisors:

Programme: Génie informatique
Program:

UNIVERSITÉ DE MONTRÉAL

ARCHITECTURE DE DÉCOUVERTE DES SERVICES WEB GÉOLOCALISÉS
DANS LES RÉSEAUX MOBILES DE PROCHAINE GÉNÉRATION

ANDRÉ CLAUDE BAYOMOCK LINWA
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIAE DOCTOR
(GÉNIE INFORMATIQUE)

Août 2005

© André Claude Bayomock Linwa, 2005.



Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 978-0-494-48882-9

Our file Notre référence

ISBN: 978-0-494-48882-9

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

UNIVERSITÉ DE MONTRÉAL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

ARCHITECTURE DE DÉCOUVERTE DES SERVICES WEB GÉOLOCALISÉS
DANS LES RÉSEAUX MOBILES DE PROCHAINE GÉNÉRATION

présentée par : BAYOMOCK LINWA André Claude

en vue de l'obtention du diplôme de : Philosophiae Doctor

a été dûment acceptée par le jury d'examen constitué de :

Mme Hanifa BOUCHENEB, Doctorat, présidente

M. Samuel PIERRE, Ph.D., membre et directeur de recherche

M. Alejandro QUINTERO, Doct., membre

M. Ferhat KHENDEK, Ph.D., membre

Dédicace

Je dédie ce travail à mon épouse *Éliador Georgine Bayomock* née *Bassong* pour son soutien moral, son endurance à supporter mes absences dans l'organisation familiale et son amour indéfectible durant les travaux de cette thèse.

J'ai une pensée pour le don que nous fait le Seigneur d'agrandir notre famille avec l'arrivée de notre bien aimée *Claudia Bayomock*. Par amour, ce travail t'est dédié pour que tu t'engages dans le sens de l'effort durant ta vie.

Vous m'avez inculqué les valeurs d'organisation, de méthode, de rigueur et d'abnégation dans la résolution des problèmes, je vous suis reconnaissant pour la clarté de votre vision et de votre engagement pour mon éducation et ma santé en tant que parents. À titre posthume, *Pauline Linwa* née *Ngo Minyem* et *Daniel Linwa Pagbe*, ce travail est le vôtre et je m'incline devant vous.

J'ai une pensée profonde pour mon beau-père *Louis Jacques Bassong* dont nous sommes sans nouvelles depuis quelques années. Où que tu sois, je partage les effets de ce travail avec toi.

À tous mes frères et sœurs (vivants ou décédés), par amour ma joie est la vôtre.

Remerciements

Je remercie le Professeur *Samuel Pierre* pour son encadrement et ses conseils tout au long de ma recherche. J'apprécie également les efforts consentis pour animer le LARIM (*Laboratoire de Recherche en Réseautique et Informatique Mobile*) dans le but de permettre un échange du savoir dans le groupe.

Je remercie tous mes collègues et le personnel du LARIM pour leur collaboration et surtout pour l'ambiance de travail chaleureuse.

Enfin, je remercie tous ceux que je n'ai pas eu la possibilité de nommer et qui m'ont soutenu, aidé ou encouragé d'une manière ou d'une autre durant ce long processus.

Résumé

Un service Web géolocalisé est un service de Web qui est offert dans une région ou un secteur géographique particulier. Dans sa phase d'exécution, un service Web géolocalisé utilise la position géographique d'un client mobile pour fournir le service. Par exemple, un service Web géolocalisé pourrait être une application de service d'urgence qui informe la police ou l'administration des sapeurs-pompiers d'une région spécifique de l'emplacement où s'est produit un accident de voiture, en utilisant la position géographique du téléphone portable du conducteur. Dans la prochaine génération (troisième génération et plus) des réseaux mobiles, pour obtenir la position d'un mobile, un service Web géolocalisé interagira avec le serveur de position ou de localisation LCS (*LoCation Service*) d'un opérateur de réseau mobile.

En se rapportant aux spécifications récentes sur les réseaux mobiles, les services Web géolocalisés seront déployés sur les réseaux mobiles de la prochaine génération. Ainsi, il sera possible à un serveur d'applications de fournisseur SAS (*Supplier Application Server*) d'obtenir l'emplacement ou la position géographique d'un client mobile. Dans ce contexte, un SAS peut être vu comme un ensemble de services Web qui nécessitent la position d'un client mobile pour fournir le service. De par sa définition, un service Web géolocalisé génère un problème de continuité ou de maintenabilité d'une exécution de service quand un client mobile entre dans une région où le service n'est plus offert par le SAS en cours d'exécution. Plusieurs études montrent que des milliers de services Web géolocalisés seront déployés dans les réseaux mobiles. Le défi principal pour un client mobile sera de découvrir un service spécifique correspondant à ses critères de sélection tels que la proximité, le coût du service, la bande passante ou le taux d'utilisation du serveur d'applications, de maintenir ou de garantir la continuité d'exécution d'un service.

Les protocoles traditionnels pour découvrir des services ne sont pas adaptés aux services Web géolocalisés car la position du client n'est pas un paramètre d'entrée pour trouver le service. Le déploiement de tels services exige donc la définition d'un protocole de découverte de service permettant, entre d'autres, de sélectionner un serveur d'applications sur la base du contexte de localisation d'un client mobile (le contexte de localisation signifie l'emplacement ou la position géographique actuelle d'un client mobile), de coordonner la migration d'un service au SAS le plus proche (c'est le SAS qui offre le service en cours d'exécution au contexte de localisation du client mobile) et de factoriser sur une base thématique (par exemple, *localiser tous les conducteurs de taxi de compagnie ABC qui se trouvent à deux kilomètres de ma position actuelle*, dans ce libellé de localisation, la portion de phrase « les conducteurs de taxi de la compagnie ABC » est le thème ou sujet) les méthodes de localisation communes aux différents types d'applications géolocalisées. Nous appelons *localisation thématique* la factorisation des méthodes de localisation sur une base thématique.

Les travaux effectués dans le domaine de la découverte des services n'abordent pas ces défis de découvrir les services Web géolocalisés, de maintenir l'exécution au serveur d'applications couvrant le contexte de localisation du client mobile et de localiser thématiquement les mobiles. Les architectures courantes sont adaptées pour découvrir les services où les clients fixes sont impliqués et celles qui considèrent la mobilité mettent l'emphasis sur le code et la mobilité d'agent, plutôt que la mobilité des données ou des tâches au serveur d'applications le plus proche du contexte de localisation du client. En effet, les services Web sont orientés services et utilisent principalement un modèle client/serveur. Il est à noter qu'aucune des architectures suggérées dans la littérature n'offre une localisation thématique.

Pour sélectionner un service avec QoS «*Quality of Service*» (coût du service par domaine, bande passante, facteur d'utilisation d'un serveur d'applications), les clients mobiles doivent indiquer au système les valeurs désirées. La quasi totalité des protocoles de découverte des services avec QoS existants intègrent l'accomplissement des opérations de négociation et de renégociation de QoS du réseau dans le processus de

découverte des services. D'un point de vue conceptuel, ces opérations font partie non pas du processus de découverte de services mais plutôt du processus d'acquisition de la QoS du réseau. Les approches qui prennent en compte cette séparation utilisent des données de QoS non consistantes (telle que la bande passante dans un domaine spécifique) dans le critère de sélection du serveur d'applications. L'introduction des données consistantes, telles que la bande passante et le facteur d'utilisation du serveur d'applications dans le critère de sélection du serveur d'applications, permet lors d'une recherche ou d'une migration de service d'effectuer la sélection du serveur en ayant une vue dynamique des données de QoS dans un domaine spécifique.

Cette thèse propose un nouveau système « intergiciel » (*middleware*) appelé architecture de services Web géolocalisés GLWSA (*Geo-Located Web Services Architecture*) qui étend les fonctionnalités des protocoles UDDI (*Universal Description, Discovery, and Integration*) et MLP (*Mobile Location Protocol*) en ajoutant la topologie de GLWSA et les méthodes de localisation thématique factorisées, respectivement. Nous proposons également un nouveau mécanisme de collecte de QoS qui collecte la bande passante et le facteur d'utilisation d'un serveur applications dans un domaine particulier, propage les données collectées aux nœuds de découverte de services de la topologie et les utilise dans le critère de sélection d'un serveur d'applications. Brièvement, le mécanisme de collecte des données dynamiques de la QoS consiste, pour un service spécifique, de collecter périodiquement la bande passante et le facteur d'utilisation d'un SAS d'un domaine particulier en envoyant un message de collecte de trafic à tout SAS qui offre le service en question dans ce domaine. L'objectif principal de GLWSA est d'agir comme un système réparti de découverte de services Web géolocalisés qui permet de maintenir une exécution de service au serveur d'applications le plus proche du contexte de localisation d'un client mobile.

Les résultats d'expérimentation obtenus montrent que le système GLWSA proposé constitue une architecture adéquate pour découvrir les services Web géolocalisés, coordonner la migration de service au serveur d'applications le plus proche et localiser thématiquement les clients mobiles. Ce système permet de rechercher un

service Web géolocalisé avec un temps RTT (*Round Trip Time*) moyen de 28 millisecondes (pour un service Web géolocalisé sans QoS) et 41 millisecondes (pour un service Web géolocalisé avec QoS). Les performances de la coordination de la migration sans ou avec qualité de service sont satisfaisantes car dans les deux cas, plus de 98% des mesures du RTT moyen de la coordination respectent le délai sur la contrainte de coordination de migration (soit inférieur ou égal à 30 millisecondes). Nous démontrons également que le mécanisme de collection et de soumission des données de la QoS du système GLWSA est consistant par rapport à d'autres systèmes effectuant également la collection des données dynamiques de la QoS (par exemple, la bande passante). Les résultats obtenus montrent que la localisation thématique proposée offre un gain substantiel en temps par rapport à la localisation ordinaire d'un groupe de mobiles.

Abstract

A geo-located web service is a web service that is offered in a particular geographical region or area. In the execution phase, a geo-located web service uses the mobile position to deliver a service. For example, a geo-located web service could be an emergency application which informs the police or firefighter administration of a particular region about the location of a car accident, using the driver's mobile phone position. In next generation (third and up generation) of mobile networks, to obtain the mobile position, a geo-located web service will interact with the position or *LoCation Services (LCS)* server of a mobile network operator.

By referring to the mobile network specifications, the geo-located web services will be deployed on the next generation mobile networks. Then, it will be possible to a *Supplier Application Server (SAS)* to get the location or geographical position of a mobile client. In this point of view, an SAS is considered as a set of web services that require the client position to deliver the service.

A geo-located web service generates a maintainability problem of a service execution when a mobile client moves to a location area where the service is no longer offered by the SAS in execution.

The major challenge for a mobile client, as geo-located web services will be increasingly deployed in organizations, will be to discover a specific service corresponding to their requirements such as proximity, service cost, network bandwidth or server utilization rate and to maintain a service in execution.

Traditional protocols for discovering services are not adapted to the geo-located web services as the client position is not an entry parameter to find a service. The deployment of such services thus requires the definition of a service discovery protocol allowing, among others, to select an SAS based on the location context of the mobile client (the location context refers to the current geographical area or position of a mobile client), to migrate services in execution to the nearest or closest SAS (it is an SAS that

offers the service in execution at the current location context of a mobile client) and to factorize common location methods based on thematic aspect (for example, locate all taxi drivers of ABC company within two miles of my current position, where ‘taxi drivers of ABC’ is a theme or subject). The selection of common locations based on thematic aspect is called a thematic location.

Related work in this field do not challenge the problem of discovering geo-located web service and maintain the service execution in the location context of a mobile client. Current architectures are either adapted to discover services where fixed clients are involved and those which consider the mobility put the emphasis on code and agent mobility, rather than data or task mobility to the closest SAS (as the web services are service oriented and use a client/server model) based on the location context of the mobile client and maintainability of service execution. Moreover, none of the architectures suggested in the literature offer a thematic factorization of common location methods to a set of application suppliers.

To select a service with QoS (service cost per domain, the bandwidth, the utilization factor of an application server), mobile clients must specify to the system, the required values to be controlled. Related work in the area of service discovering with QoS emphasizes in the real-time completion of network QoS negotiation and re-negotiation operations to return service address to a client. In design aspect, these operations are not in the discovering service process but in network QoS process. Some approaches which take in account this separation use non consistent QoS data (network bandwidth view in a particular domain) in the application server selection. The input of the consistent network bandwidth in the server selection and service migration criterion induces to select an application server and to migrate a service with knowledge of the network bandwidth traffic in a specific domain.

This thesis proposes a new middleware system called *Geo-Located Web Services Architecture (GLWSA)* which extends the UDDI (*Universal Description, Discovery, and Integration*) registry and the MLP (*Mobile Location Protocol*) by adding the GLWSA topology and the factorized thematic location methods, respectively. We built a new

mechanism that collects the network bandwidth and the utilization factor of a server application, maintains the consistency of collected data and uses them in the selection and the migration criteria of the SAS server. For a specific service, the network bandwidth and SAS utilization factor are collected periodically in a particular GLWSM (*Geo-Located Web Services Manager*) domain by sending a collect traffic message to all SAS that offer the concerned service in this domain. Then, each SAS sends a bandwidth quotation message to the MAP (*Mobile Anchor Point*) of the GLWSM domain in order to get the bandwidth traffic and determines the SAS processor rate. The proposed mechanism aims to select application servers with dynamic QoS parameters (network bandwidth, server utilization factor). The main objective of the GLWSA is to propose a distributed geo-located web services discovery system that allows to maintain a service execution closest to the location context of a mobile client.

Results show that the suggested GLWSA system offers an adequate architecture to discover the geo-located Web services, to coordinate the migration of service to the nearest applications server and to locate thematically the mobile clients. It makes it possible to lookup a geo-located Web service with an average RTT (*Round Time Trip*) of 28 milliseconds (for a geo-located Web service without QoS) and 41 milliseconds (for a geo-located Web service with QoS). The performances of the coordination of the migration without or with quality of service are satisfactory because in both cases 98% of measurements of the average RTT of coordination respects the time on the constraint of coordination of migration (either less than or equal to 30 milliseconds). We also show that the mechanism of collection and propagation of the QoS data in GLWSA system is consistent compared to other systems also carrying out the collection of the dynamic QoS data (for example, the bandwidth). The results obtained show that the thematic location suggested offers a substantial gain in time compared to the ordinary location of a group of mobiles.

Table des matières

Dédicace.....	iv
Remerciements.....	v
Résumé.....	vi
Abstract.....	x
Table des matières.....	xiii
Liste des tableaux.....	xvii
Liste des figures.....	xviii
Liste des sigles et abréviations.....	xxi
Chapitre 1 Introduction	1
1.1 Définitions et concepts de base	4
1.1.1 La localisation des mobiles	4
1.1.2 La localisation des services	10
1.2 Éléments de la problématique	11
1.3 Objectifs de recherche	13
1.4 Esquisse méthodologique	15
1.5 Principales contributions et originalité	16
1.6 Plan de la thèse	19
Chapitre 2 Découverte et distribution des services	20
2.1 Définitions et concepts de base	21
2.1.1 Le protocole WSDL.....	21
2.1.2 Le protocole UDDI.....	23
2.1.3 Le protocole SOAP.....	25
2.1.4 Le protocole MLP.....	26
2.2 Découverte et distribution des services	29
2.2.1 Les protocoles classiques	30
2.2.2 Découverte des services avec les paramètres de QoS	35
2.2.3 Co-localisation service/usager, facturation dépendante de la localisation..	37

2.2.4	Modélisation de la mobilité et distribution des services.....	38
2.2.5	Découverte de services avec contraintes de sécurité.....	41
2.3	Synthèse des requis et défis de recherche.....	42
2.3.1	Distribution des services.....	42
2.3.2	Factorisation des méthodes de localisation des usagers.....	43
2.3.3	La qualité des services.....	43
2.3.4	Sécurité des services.....	43
Chapitre 3	Architecture de découverte de services géolocalisés proposée	45
3.1	Principaux requis de l'architecture proposée.....	45
3.2	Principes de l'architecture proposée.....	47
3.2.1	Facteurs influençant le choix topologique du système GLWSA.....	47
3.2.2	Principes de génie logiciel appliqués.....	50
3.3	Architecture du système GLWSA.....	53
3.3.1	Description des principales composantes.....	53
3.3.2	Recherche d'un service Web géolocalisé.....	56
3.3.3	Publication d'un service Web géolocalisé.....	59
3.3.4	Coordination de la migration d'un service Web géolocalisé.....	61
3.3.5	Collection et consistance des données de QoS.....	63
3.3.6	Impact de la mobilité.....	68
3.3.7	Extension des protocoles UDDI et MLP.....	70
3.3.8	Typologie des applications géolocalisées.....	73
3.4	Formalisme mathématique du système GLWSA.....	78
3.4.1	Ensembles des serveurs de découverte des services et de zones de localisation.....	79
3.4.2	Ensembles des services déployés et des serveurs d'applications de la topologie.....	80
3.4.3	Publication d'un service Web dans le système <i>GLWSA</i>	82
3.4.4	Recherche d'un service Web dans le système <i>GLWSA</i>	83

3.4.5	Coordination de la migration d'un service Web dans le système GLWSA	84
Chapitre 4	Implémentation du système GLWSA	86
4.1	Architecture fonctionnelle du système GLWSA	86
4.2	La couche des services publiés synchrones et d'écoute des messages asynchrones	87
4.2.1	Les services publiés ou interfaces externes synchrones	87
4.2.2	Écoute des messages asynchrones	90
4.3	La couche de gestion des services	91
4.3.1	Authentification et autorisation	92
4.3.2	Gestion de la topologie	92
4.3.3	Gestion des agréments de service	93
4.3.4	Gestion de la QoS dans le réseau	94
4.3.5	Gestion de la découverte des services	98
4.3.6	Gestion de localisation des mobiles	100
4.3.7	Gestion de la coordination d'une migration de services	101
4.3.8	Gestion d'abonnement	103
4.4	La couche des classes de base	103
4.5	La persistance dans le système GLWSA	105
4.6	Extension du protocole MLP : MLPe	106
4.6.1	Structures de données proposées	106
4.6.2	Gestion des structures Theme et ThemeMsid	107
4.6.3	Méthodes de localisation thématique	107
4.6.4	Modèle de sécurité	108
Chapitre 5	Évaluation de l'architecture et résultats	111
5.1	Évaluation des requis fonctionnels	111
5.1.1	Paramètres de configuration de l'environnement de test	111
5.1.2	Résultats et analyse des performances	116
5.2	Évaluation de la collecte des données dynamiques de QoS	129

Chapitre 6 Conclusion.....	136
6.1 Synthèse des travaux	137
6.2 Limitations des travaux	139
6.3 Travaux futurs	140
Références bibliographiques.....	142
Annexe	152

Liste des tableaux

Tableau 2.1 Éléments d'un document WSDL	22
Tableau 2.2 Description des services du protocole MLP	27
Tableau 4.1 Interfaces externes synchrones du système GLWSA	90
Tableau 4.2 Messages asynchrones échangés dans le système GLWSA	91
Tableau 4.3 Classes de base de l'architecture fonctionnelle de GLWSA	103
Tableau 5.1 Rôle des machines dans l'environnement de test	112
Tableau 5.2 Extrait du fichier de la topologie des cellules	114
Tableau 5.3 Extrait du fichier du trajet emprunté par les mobiles	115
Tableau 5.4 Valeurs des liens	131

Liste des figures

Figure 1.1 Architecture LCS issue de UMTS	4
Figure 1.2 Réseau LCS d'un point de vue applicatif	7
Figure 1.3 Méthode OTDOA	9
Figure 1.4 Méthode A-GPS	10
Figure 1.5 Interactions dans une découverte des services	11
Figure 2.1 Diagramme de classes UML de la structure de données UDDI	23
Figure 2.2 Requête SOAP	25
Figure 2.3 Réponse SOAP	25
Figure 2.4 Requête standard de localisation immédiate	28
Figure 2.5 Réponse standard de localisation immédiate	29
Figure 3.1 Architecture du système GLWSA	53
Figure 3.2 Topologie du système GLWSA	55
Figure 3.3 Cas d'utilisation d'une recherche de service Web géolocalisé	57
Figure 3.4 Diagramme de déploiement d'une recherche de service	58
Figure 3.5 Cas d'utilisation d'une publication de service Web géolocalisé	59
Figure 3.6 Diagramme de déploiement d'une publication de service Web géolocalisé	60
Figure 3.7 Cas d'utilisation de la coordination d'une migration de service Web géolocalisé	61
Figure 3.8 Diagramme de déploiement de la coordination d'une migration de service Web géolocalisé	62
Figure 3.9 Choix d'un serveur SAS en utilisant le critère de la bande passante	64
Figure 3.10 Collection et consistance de la QoS dans un domaine GLWSMh	65
Figure 3.11 Appréciation de la bande passante	65
Figure 3.12 Modèle de propagation de QoS	67
Figure 3.13 Impact de la mobilité dans GLWSA avec HMIPv6	69
Figure 3.14 Structures de données de UDDIM	71
Figure 3.15 Structure de données de MLPe	78

Figure 4.1 Architecture fonctionnelle du système GLWSA	86
Figure 4.2 Exemple de contenu d'un fichier de description et de déploiement des services Web	88
Figure 4.3 Extrait de l'implémentation de QuotationBW	95
Figure 4.4 Extrait de l'implémentation de NetworkQoSManager	97
Figure 4.5 Extrait de l'implémentation de LocationManager	101
Figure 4.6 Diagramme de classes des classes de base de l'architecture fonctionnelle	104
Figure 4.7 Persistance des données dans le système GLWSA	105
Figure 5.1 Topologie générée et trajectoire d'un client cible	113
Figure 5.2 Recherche d'un service Web géolocalisé sans QoS	117
Figure 5.3 Variation linéaire du nombre de clients lors d'une recherche d'un service Web géolocalisé sans QoS	118
Figure 5.4 Comparaison GLWSM, SLP et Jini lors d'une recherche d'un service Web géolocalisé sans QoS	119
Figure 5.5 Recherche d'un service Web géolocalisé avec QoS	120
Figure 5.6 Variation linéaire du nombre de clients lors d'une recherche d'un service Web géolocalisé avec QoS	121
Figure 5.7 Publication d'un service Web géolocalisé	122
Figure 5.8 Publication d'un service Web géolocalisé à 5 nœuds « feuille»	123
Figure 5.9 Publication d'un service Web géolocalisé à 10 nœuds « feuille»	123
Figure 5.10 Publication d'un service Web géolocalisé à 20 nœuds « feuille»	124
Figure 5.11 Coordination de migration d'un service sans QoS	125
Figure 5.12 Coordination de migration d'un service avec QoS	127
Figure 5.13 Localisation thématique versus localisation par groupe de mobiles	129
Figure 5.14 Topologie de la simulation	130
Figure 5.15 Soumission de la bande passante	131
Figure 5.16 Mise à jour et propagation de la soumission des données de QoS dans les systèmes QoS-Aware et GLWSA	133
Figure 5.17 Réponse du rafraîchissement moyen	134
Figure A.1 Extrait de l'implémentation de TopologyManager	152

Figure A.2 Extrait de l'implémentation de AgreementManager	154
Figure A.3 Extrait de l'implémentation de DvtServiceManager	156
Figure A.4 Extrait de l'implémentation de MigrationManager	158
Figure A.5 Extrait des méthodes de localisation thématique proposées	162

Liste des sigles et abréviations

3G-MSC	Contrôleur des stations de base 3G.
A-GPS	Assisted-Global Position System
AMRB	Application Module Request Broker
AOA	Angle Of Arrival
API	Application Program Interface
AR	Access Router
CDMA-2000	Code Division Multiple Access 2000
CELL_ID	Méthode de détermination de la position par identification des cellules
CM	Client mobile
DAC	Discretionary Access Control
DTD	Data Type Definition
DHCP	Dynamic Host Configuration Protocol
DiffServ	Differentiated Services
EM	L'équipement mobile.
E-OTD	Enhanced Observed Time Difference
FCC	Federal Communication Commission
GGSN	Gateway GPRS Support Node
GIS	Geographical Information System
GLWSA	Geo-Located Web Services Architecture
GLWSM	Geo-Located Web Services Manager
GMLC	Gateway Mobile Location Center
GPRS	General Packets Radio System
GPS	Global position System
G-QoS	Grid Quality of Service Manager
GR	Gateway Router
GSM	Global System for Mobiles
HLR	Home Location Register
HMIP	Hierarchical Mobile IP

HTML	HyperText Markup Language
HTTP	HyperText Transport Protocol
IMT-2000	International Mobile Telecommunications-2000
IP	Internet Protocol
JDBC	Java DataBase Connectivity
JMS	Java Message Service
JRE	Java Runtime Environment
JVM	Java Virtual Machine
LCoA	Local Care of Address
LDQ	Location Dependent Query
LIF	Location Interoperability Forum
LMU	Location Measurement Unit
MAC	Mandatory Access Control
MAP	Mobile Anchor Point
MIP	Mobile IP
MLP	Mobile Location Protocol
MLPe	Mobile Location Protocol Extension
MOD	Movement Of Objects
NODE-B	Appellation d'une station de base dans le réseau 3G UMTS
ODBC	Open DataBase Connectivity
OTD	Observed Time Difference
OTDOA	Observed Time Difference Of Arrival
PDA	Portable Data Terminal
QML	QoS Modeling Language
QoS	Quality Of Service
QRPC	Queued Remote Procedure Call
RCoA	Regional Care of Address
RCSPM	Reconfiguration Control and Service Provisioning Manager
RDO	Relocable Dynamic Object
RNC	Radio Network Controller
RNF	Requis Non Fonctionnel

RPC	Remote Procedure Call
RSVP	ReSource reserVation Protocol
RTT	Round Trip Time
SAS	Supplier Application Server
SDP	Service Discovery Protocol
SDS	Serveur de Découverte de Services
SGSN	Serving GPRS Support Node
SIG	Système d'Informations Géographiques
SIM	Subscriber Identification Module
SLP	Service Location Protocol
SMLC	Serving Mobile Location Center
SOAP	Simple Object Access Protocol
SRNC	Serving Radio Network Controller
SSDP	Simple Service Discovery Protocol
TCL	Tool Command Language
TDOA	Time Difference Of Arrival
UDDI	Universal Description, Discovery and Integration
UDDIM	UDDI for Mobiles
UF	Utilization Factor
UM	Unité Mobile
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System
UPnP	Universal Plug and Play
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTM	Universal Traverse Mercator
UTRAN	UMTS Terrestrial Radio Access Network
UWC-136	Universal Wireless Communication 136
VLR	Visitor Location Register
WML	Wireless Markup Language
WSDD	Web Service Deployment Descriptor

WSDL	Web Service Description Language
XML	eXtended Markup Language

Liste des annexes

Annexe 152

CHAPITRE I

INTRODUCTION

Les réseaux mobiles ont pour principal rôle de mettre en communication les unités mobiles telles que les ordinateurs portables ou les PDA avec des entités physiques fixes (serveurs d'applications, téléphone fixe, télécopieur, etc.) ou d'autres unités mobiles par le biais des ressources d'un opérateur réseau mobile. Dans ces réseaux, le trafic des messages échangés entre les entités émettrice et réceptrice impliquées dans une communication passe par une interface radio, un réseau radio et un réseau cœur [1], [2], éléments qui constituent le réseau mobile, et par des réseaux indépendants tels que Internet, pour acheminer des paquets de données au destinataire.

Dans la spécification des réseaux mobiles de prochaine génération [1], [2], il y est indiqué qu'il sera possible grâce aux techniques de localisation [32], [69], et par le biais du serveur de localisation ou de position LCS (*LoCation Services*) d'un opérateur réseau, de restituer aux clients intéressés, la position géographique des unités mobiles se déplaçant le long de la topologie des réseaux mobiles. L'implémentation de cette spécification ouvre la voie au développement des applications utilisant la position géographique des unités mobiles afin d'octroyer un service. Un tel type de service, qui est offert dans une région géographique donnée et qui nécessite pour son exécution la position géographique d'un client, est communément appelé service géolocalisé [60]. Certains exemples d'applications des services géolocalisés ont été étudiés dans la littérature pour démontrer l'apport de la géolocalisation dans le développement des applications mobiles [54], [70], [78], [82], [88], [92], [99]. Pour développer les services géolocalisés, la technologie des services Web est généralement suggérée [26] car ces services sont faiblement couplés et doivent être omniprésents (notion d'ubiquité), interopérables et disponibles pour d'autres applications. Nous appelons un tel service un *service Web géolocalisé*.

Contrairement aux applications client/serveur ordinaires, où le client mobile est rattaché à un serveur d'applications au cours de sa mobilité d'un réseau d'accès aux paquets à un autre, la nouvelle capacité (restitution de la position géographique) offerte par les réseaux mobiles, permet de développer des applications où l'exécution du service s'effectuera au serveur d'applications le plus proche du lieu de localisation actuel d'un client mobile requérant. En choisissant continuellement le serveur d'applications le plus proche, le temps de propagation client/serveur décroît si la distance entre le client et le serveur diminue.

D'après certaines études [26], [32], [45], les services Web géolocalisés seront de plus en déployés dans le futur. Ainsi, pour un client donné, le défi principal dans une telle configuration sera de découvrir un service spécifique correspondant à certains requis sur la proximité, le coût du service, la bande passante du réseau, le facteur d'utilisation du serveur ou le délai de propagation.

Les protocoles traditionnels de découverte des services (SLP, Jini, DNS, UDDI, Globe) ne sont pas adaptés aux services Web géolocalisés car la position du client n'est pas un paramètre requis pour sélectionner un service. Le déploiement de tels services exige ainsi la définition d'un protocole de découverte de services permettant, entre autres, de choisir un serveur du fournisseur d'application SAS (*Supplier Application Server*) sur la base du contexte de localisation du client mobile, d'informer le serveur d'applications de migrer le service d'un client mobile en cours d'exécution vers un SAS géographiquement le plus proche et de factoriser thématiquement les méthodes de localisation d'un groupe de mobiles. Le terme utilisé pour dénommer ce type de localisation est *la localisation thématique*. Un exemple de localisation thématique d'un groupe de mobiles serait «*localiser tous les conducteurs de taxi de la compagnie ABC situés à deux kilomètres de ma position actuelle* ». Dans cet exemple, la portion de texte «*les conducteurs de taxi de la compagnie ABC* » matérialise le thème ou le sujet. L'analyse des travaux de recherche relatifs à la découverte des services ne nous a pas permis de relever l'existence d'une proposition couvrant ces différents requis de conception. Les architectures courantes sont adaptées pour découvrir des services où les

clients fixes sont impliqués [39], [73], [85], et celles qui considèrent la mobilité mettent l'emphasis sur la mobilité de code (agent mobile) [24], [41], [59] plutôt que l'exécution de services dans un serveur d'applications proche du contexte de localisation du client mobile. D'ailleurs, aucune des architectures proposées dans la littérature n'offre la possibilité d'effectuer une localisation thématique d'un groupe de mobiles liés par un thème.

De plus, la plupart des travaux sur la découverte des services avec QoS [4], [5], [30], [44], [61], mettent l'accent sur la complétude en temps réel des opérations de négociation et de renégociation de la QoS des réseaux avant de retourner l'adresse du service au client requérant. En réalité, pour que les opérations de négociation et de renégociation prennent effet, le serveur de découverte de services doit avoir une connaissance à priori de l'adresse du serveur d'applications à lier à la requête du client. Cela signifie en clair que le choix du serveur d'applications s'effectue aléatoirement, avant d'entreprendre la négociation et la renégociation de la QoS des réseaux traversés. D'un point de vue conceptuel, ces opérations ne sont pas des opérations du processus de découverte des services mais plutôt du processus d'acquisition des paramètres de QoS du trafic réseau. Afin d'introduire la connaissance du trafic réseau dans la prise de décision du processus de sélection des serveurs d'applications, il faut donc séparer les opérations de négociation et de renégociation du processus de découverte des services, puis d'injecter et maintenir dans le serveur de découverte de services d'un domaine spécifique, une vue consistante des paramètres du trafic réseau dudit domaine. Les travaux actuels qui tiennent compte de cette séparation [30] maintiennent des paramètres de QoS (bande passante ou facteur d'utilisation d'un SAS) non consistantes dans le processus de sélection du serveur d'applications.

Dans l'optique de résoudre les défis énoncés ci-dessus, il importe de définir une architecture cohérente et robuste de découverte des services Web géolocalisés pour les réseaux mobiles de prochaine génération, ce qui fait l'objet de cette thèse. Nous appelons cette architecture de découverte de services **GLWSA** (*Geo-Located Web Services Architecture*).

Dans ce chapitre d'introduction, nous présenterons les définitions et concepts de base, exposerons les éléments de la problématique, préciserons les objectifs de la recherche et esquisserons la méthodologie utilisée ainsi que le plan de la thèse.

1.1 Définitions et concepts de base

La découverte des services géolocalisés dans un réseau mobile fait ressortir deux principales notions : la localisation des mobiles et la localisation des services ou découverte des services proprement dite.

1.1.1 La localisation des mobiles

Grâce à l'implémentation des spécifications du standard IMT-2000 (*International Mobile Telecommunications-2000*) [77], [90], les réseaux mobiles de prochaine génération ont le potentiel de permettre le développement des services basés sur le contexte de localisation géographique des unités mobiles. Différentes architectures ayant la fonctionnalité de déterminer la localisation géographique des clients mobiles sont proposées par les opérateurs des réseaux mobiles telles que UMTS (*Universal Mobile Telecommunications System*) [32] ou CDMA-2000 (*Code Division Multiple Access 2000*) [2]. Dans la littérature, les réseaux mobiles qui ont le potentiel de restituer la localisation géographique des unités mobiles aux applications sont appelés réseaux LCS (*LoCation Services*) [18], [32]. Un exemple de réseau LCS est le réseau LCS de UMTS illustré à la Figure 1.1.

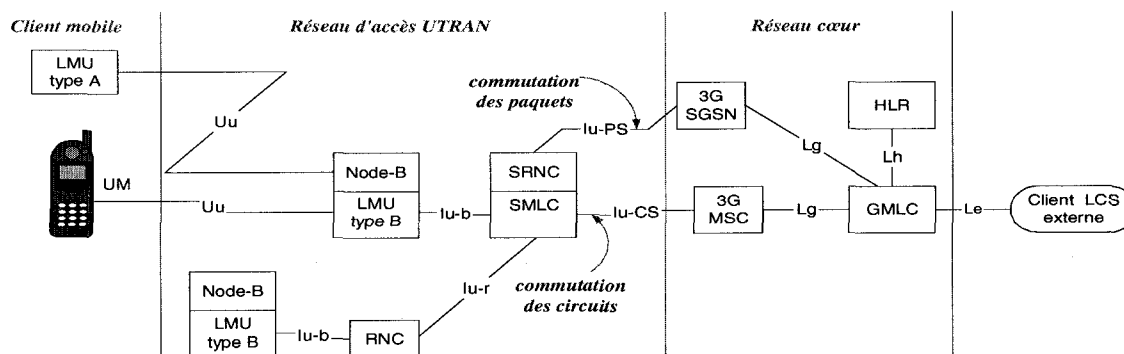


Figure 1.1 Architecture LCS issue de UMTS

Un réseau UMTS est composé de trois domaines : le domaine du client mobile, le domaine du réseau d'accès UTRAN (*UMTS Terrestrial Radio Access Network*) et le domaine du réseau cœur. Le domaine du client mobile regroupe différents types de terminaux que l'utilisateur peut utiliser (PDAs, ordinateurs, téléphones, etc.) pour accéder à l'infrastructure et aux services. Le réseau d'accès fournit à un usager du domaine client mobile les ressources nécessaires pour accéder au réseau cœur. Le réseau d'accès UTRAN est composé des équipements tels que la station de base nœud B (*node-B*), le contrôleur du réseau radio RNC (*Radio Network Controller*) et le contrôleur des services du réseau radio SRNC (*Serving Radio Network Controller*). Le rôle principal du nœud B est d'assurer la transmission et la réception radio entre l'UTRAN et un ou plusieurs équipements usagers. Un RNC gère l'admission et la relève (*handover*) des requêtes destinées aux nœuds B qui sont sous son contrôle. Le SRNC permet d'interconnecter le réseau d'accès au réseau cœur. Le terme sous-système du réseau radio RNS (*Radio Network System*) est généralement utilisé pour désigner un RNC et l'ensemble des nœuds B qui lui sont associés. Le réseau cœur est composé de l'ensemble des équipements qui assurent des fonctions telles que le contrôle des appels, le contrôle de la sécurité, la gestion de l'interface avec les réseaux externes. Il permet aux usagers de communiquer à l'intérieur d'un même réseau de téléphonie mobile, assure l'interconnexion avec les réseaux externes et permet de maintenir des communications sécuritaires même lorsque l'utilisateur est en déplacement. Le réseau cœur est subdivisé en domaines à commutation de circuits CS (*Circuit Switch*) et à commutation de paquets PS (*Packet Switch*). Le domaine de commutation de circuits est composé des équipements tels que 3G-MSC et VLR. Le domaine de commutation des paquets est composé des équipements tels que le nœud de service d'un paquet généralisé de service radio SGSN (*Serving General packet radio service Support Node*), le nœud passerelle d'un paquet généralisé de service radio GGSN (*Gateway General packet radio service Support Node*) et HLR (*Home Location Register*).

Les principaux composants matériels de l'architecture LCS sont: l'unité mobile (*UM*), l'unité des mesures de localisation (*LMU*), le centre de localisation des

services du mobile (*SMLC*), la passerelle au serveur de localisation des services des mobiles (*GMLC*), le *3G-MSC/VLR*, la station de base (*Node-B*), le contrôleur du sous-système radio (*RNC*), le contrôleur du sous-système radio courant (*SRNC*) et le registre de localisation de résidence (*HLR*).

L'unité mobile utilise la communication radio pour interagir avec les stations de base *Node-B*. Certaines unités mobiles sont équipées d'un équipement GPS «Global Position System» qui détecte la position géographique d'une unité mobile.

La LMU capte les signaux émis par les stations de base qu'elle contrôle (3 ou 4 *Node-B*), puis effectue des calculs de différence de temps d'arrivée des paires de signaux captés. Ces calculs sont nécessaires aux algorithmes de détermination de la position géographique d'une unité mobile tels que *E-OTD (Enhanced Observed Time Difference)* et *OTDOA (Observed Time Difference Of Arrival)*. Les LMUs sont intégrées dans les *Node-B* (LMU type B) ou des entités distinctes et distribuées dans le réseau (LMU type A).

Le *SMLC* est le gestionnaire principal des activités de détermination de la position géographique des mobiles. Il reçoit de la *GMLC* les requêtes de détermination de la position des unités mobiles et les transmet aux LMUs appropriées. Les calculs intermédiaires effectués par les LMUs ou les unités mobiles sont transmis par la suite au *SMLC* pour déterminer la position géographique. Le *SMLC* est un élément réseau distinct ou intégré au *SRNC*.

La *GMLC* reçoit d'un abonné au service les requêtes de détermination de la position géographique d'un client mobile. Elle contrôle le mécanisme d'authentification, de facturation du requérant et vérifie ses droits d'accès aux ressources. Elle transmet ensuite la requête au *SMLC* et retourne la position géographique calculée au client concerné. Dans la littérature de la localisation des services de réseaux mobiles, l'équipement qui reçoit les requêtes de positionnement et calcule la position des unités mobiles est connu sous le nom de *serveur LCS*.

Le *3G-MSC* est un centre de commutation des mobiles et le *VLR* est le registre de localisation des visiteurs. Le *Node-B* est l'appellation donnée à une station de base dans

le système UMTS. Le RNC est le contrôleur du sous-système radio. Il interconnecte plusieurs Node-B. Le SRNC est le contrôleur du sous-système radio courant d'une unité mobile donnée. Le HLR contient les informations de souscription au LCS, d'authentification et d'identification des clients mobiles. Le client LCS externe est généralement un serveur d'applications d'un fournisseur qui utilise le réseau LCS afin de déterminer la position des clients mobiles.

Les interfaces (Uu , $Iu-b$, $Iu-r$, $Iu-CS$, $Iu-PS$, Lg , Lh et Le) permettent d'établir un échange d'informations entre deux entités distinctes. L'interface Uu permet d'établir une communication entre l'unité mobile et les ressources radio (Node-B, SRNC). L'interface $Iu-b$ permet d'échanger entre un RNC et un Node-B les informations de signalisation et de contrôle telles que la gestion des canaux communs, des ressources communes ou de configuration des cellules. L'interface $Iu-r$ permet d'échanger entre deux RNCs les informations de signalisation et de contrôle telles que la gestion des liens radio et physiques, et la gestion de relocalisation des SRNCs. Les interfaces $Iu-CS$ et $Iu-PS$ permettent d'échanger les informations entre un RNC et le domaine de commutation des circuits du réseau cœur et entre un RNC et le domaine de commutation des paquets du réseau cœur, respectivement. L'interface Lg permet que les domaines de la commutation des circuits et des paquets puissent communiquer avec la GMLC. L'interface Lh permet un échange d'informations entre le GMLC et le HLR. L'interface Le permet à un client externe d'interagir avec les ressources de localisation des mobiles.

D'un point de vue applicatif, la représentation du réseau LCS (Figure 1.2) revient à un serveur localisation LCS qui regroupe les fonctionnalités de détermination de la position des mobiles et interagit avec les ressources du réseau radio et du réseau cœur considérées comme boîte noire afin de calculer la position des clients mobiles.

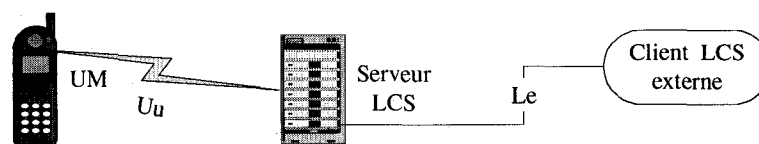


Figure 1.2 Réseau LCS d'un point de vue applicatif

Pour déterminer la position géographique d'une unité mobile, le réseau LCS utilise des méthodes pour calculer la position. Ces méthodes peuvent être classifiées en 2 catégories [6], [50], [83], [87] : les méthodes orientées terminal (calcul de la position dans le terminal de l'unité mobile) et les méthodes orientées réseau (calcul de la position dans un réseau). Nous présentons ci dessous les méthodes requises dans les réseaux mobiles de prochaine génération.

➤ **Méthode CELL_ID**

La méthode d'identification de la cellule CELL_ID consiste à déterminer la cellule où se trouve l'équipement mobile. C'est une méthode issue des procédures de mise à jour et de recherche d'une unité mobile dans le réseau étudié. Généralement, la méthode CELL_ID est utilisée de concert avec la méthode RTT pour améliorer la précision des résultats [32]. Sa précision varie entre 300 mètres et 10 kilomètres et dépend du diamètre de la cellule.

➤ **Méthode RTT**

La méthode RTT permet de mesurer le temps RTT (*Round Trip Time*) et d'améliorer la précision de mesure de CELL_ID, en calculant le temps d'un trajet aller-retour d'un signal émis par la station de base courante vers l'unité mobile [32]. La mesure de ce temps est effectuée à la LMU associée à la station de base courante, puis la distance relative entre la station de base courante et l'unité est alors déduite. Sa précision est comprise entre 100 et 500 mètres.

➤ **Méthode OTDOA**

Dans la méthode OTDOA (*Observed Time Difference Of Arrival*), la position géographique de toutes les stations de base d'un réseau est supposée connue et une LMU est en charge de contrôler trois stations de base [69]. Cette méthode est illustrée à la Figure 1.3.

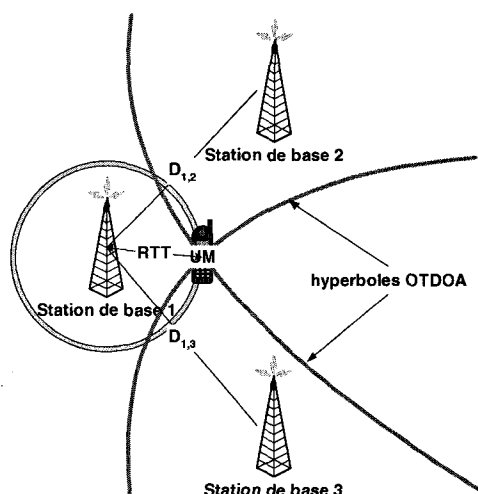


Figure 1.3 Méthode OTDOA

Pour déterminer la position, la LMU envoie un signal de synchronisation aux trois stations de base les plus proches de l'unité mobile. À la réception du signal de synchronisation, chacune de ces stations émet un signal vers l'unité mobile. L'unité mobile mesure la différence de temps d'inter-arrivée des signaux reçus. Les différences de temps mesurées par l'unité mobile sont transmises par la suite au SMLC afin d'estimer la position géographique du mobile. L'emplacement géométrique de l'unité mobile est le point d'intersection des deux courbes hyperboliques tel qu'illustré à la Figure 1.3. Sa précision est de l'ordre de 300 mètres.

➤ **Méthode A-GPS**

A-GPS (*Assisted Global Position System*) illustrée à la Figure 1.4, est une méthode hybride dans le sens qu'elle utilise les ressources satellites GPS (*Global Position System*) et les ressources du réseau mobile pour déterminer la position géographique d'une unité mobile [29], [94], [102]. Fonctionnellement, l'unité mobile est localisée préalablement par le réseau mobile en utilisant la méthode CELL_ID. Puis, afin d'améliorer la précision des résultats, l'unité mobile et la LMU sont équipées chacune d'un capteur GPS et reçoivent des signaux en provenance d'au moins 4 satellites. La LMU connaît en permanence l'adresse des quatre satellites avec lesquelles

elle est en communication car elle reçoit périodiquement de ceux-ci leur signature et leur position géostationnaire. Elle transmet les adresses des satellites à l'unité mobile afin d'éliminer le temps de recherche desdits satellites. L'unité mobile envoie par la suite un signal à ces quatre satellites afin de déterminer le RTT avec chacun d'eux. Après avoir calculé le délai de réception de chaque signal satellite, l'unité mobile les transmet à la LMU qui détermine la position géographique de l'unité mobile. Sa précision est de l'ordre de 5 mètres.

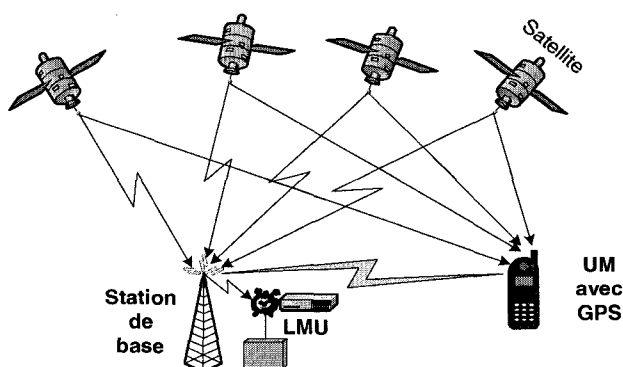


Figure 1.4 Méthode A-GPS

1.1.2 La localisation des services

La localisation des services, plus connue sous la dénomination découverte des services, illustrée à la Figure 1.5, a trois fonctionnalités principales : la publication d'un service, la recherche d'un service et la mise en relation du client avec un serveur d'applications [23], [103]. Nous entendons par service un ensemble de méthodes publiques spécifiées dans une interface ou un document WSDL (*Web Service Document Language*) et mise à la disposition des clients dans le but d'interagir avec un composant logiciel d'une application spécifique. Un service a un nom et plusieurs autres attributs tels que l'adresse de localisation ou sa description [40], [103].

La publication des services est un composant logiciel implémentant une méthode ou un ensemble de méthodes qui permettent d'enregistrer la description d'un service dans un registre de découverte des services [22], [42], [43], [71], [103]. Elle consiste à transmettre les informations qui décrivent un service au serveur de découverte de

services et à les enregistrer dans un registre dédié. Si les serveurs de découverte de services sont distribués, la publication des services fait appel à des fonctionnalités de réplication de services pour maintenir une consistance des données réparties.

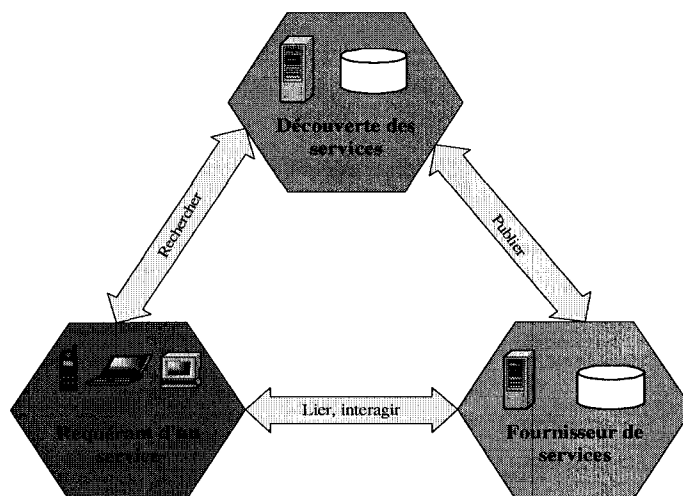


Figure 1.5 Interactions dans une découverte des services

La recherche des services est un composant logiciel implémentant une méthode ou un ensemble de méthodes permettant de sélectionner un service sur la base des paramètres requis par un requérant [22], [42], [43], [71], [103]. Elle consiste principalement à sélectionner un service désiré, puis de trouver le serveur d'applications le mieux approprié aux requis de la requête envoyée (par exemple la position géographique du client mobile). Elle retourne au requérant l'adresse du serveur d'applications et le met en relation avec le serveur d'applications sélectionné grâce à une redirection de la requête vers l'adresse dudit serveur [103].

1.2 Éléments de la problématique

Avec l'étendue des utilisateurs interconnectés au réseau Internet, les fournisseurs d'applications distribuent leurs services afin de réguler la charge des serveurs d'applications et de tenir compte de l'expansion des clients. Pour découvrir ces services, des serveurs de découverte des services sont déployés et implémentent des protocoles de

découverte des services qui maintiennent les informations sur la description détaillée des services des fournisseurs d'applications.

Dans une communication de bout-en-bout entre un client mobile et un serveur d'applications, les messages des clients mobiles interconnectés au réseau Internet transitent au préalable par un réseau mobile, ce qui augmente le temps global de transit. Ce temps a une incidence sur la facturation des communications des clients mobiles. En effet, la tarification des clients mobiles est élevée et établie au volume de données échangées ou au temps d'utilisation des ressources [49], alors que les clients fixes ont généralement un taux forfaitaire d'utilisation à prix modique. Certains opérateurs réseaux mobiles tels que SFR [75] ont choisi d'offrir leur tarification sur la durée d'utilisation des ressources. Dans le but de réduire le temps global de transit, les serveurs d'applications doivent être distribués avec comme critère principal le rapprochement des services aux clients mobiles. Ceci n'est possible que si l'on connaît exactement la position géographique des clients mobiles. Grâce aux réseaux mobiles de prochaine génération, cette condition est remplie par un serveur LCS qui retourne aux applications la position géographique d'un client mobile. Cependant, les protocoles de découverte de services classiques [39], [73], [85], [91] ne permettent pas de découvrir les services sur la base du contexte de localisation des clients mobiles et encore moins de maintenir l'exécution d'un service au serveur d'applications le plus proche du client mobile le long de sa trajectoire de déplacement dans le réseau.

Afin de tenir compte de ce critère de rapprochement des services du contexte de localisation des clients mobiles, il est nécessaire de définir une topologie adéquate de répartition des serveurs de découverte de services sur un réseau LCS d'un opérateur donné. Ce choix topologique vise par ailleurs à réduire le flux de messages échangés dans une communication de bout-en-bout et à maintenir des données consistantes de QoS dans le cas d'une découverte de services avec QoS. Les opérations de gestion de cette topologie ont été adjointes au protocole UDDI (*Universal Description, Discovery and Integration*) utilisé comme standard de découverte des services Web. Dans l'optique de découpler le serveur LCS et les serveurs d'applications SAS, il est convenable de

factoriser les fonctionnalités de localisation des mobiles au serveur «intergiciel» de découverte des services. Le découplage est nécessaire si on souhaite couvrir un grand éventail de fournisseurs d'applications. Dans la littérature, nous n'avons pas relevé l'existence d'une factorisation sur une base thématique des méthodes de localisation des mobiles. Nous démontrons dans les résultats obtenus que la localisation thématique permettrait, dans une interrogation d'un groupe d'utilisateurs mobiles, de diminuer la taille des requêtes sur la position des clients mobiles formulée par les applications. Car, les messages envoyés contiendront le nom spécifique du sujet interrogé au lieu d'une liste d'identifiants de mobiles que l'on veut interroger tel que cela se fait présentement.

Dans le cas d'une recherche d'un service avec QoS, les protocoles existants sélectionnent un serveur d'applications sur une base aléatoire avant d'effectuer la négociation et la re-négociation de la QoS. Pour résoudre ce défaut architectural, nous proposons un nouveau mécanisme de collecte de QoS (bande passante et facteur d'utilisation d'un SAS) qui permettra d'utiliser les paramètres de QoS collectés dans le processus de sélection du serveur d'applications.

La mise en place d'une architecture adéquate de découverte des services Web déployée le long d'un réseau mobile et satisfaisant un certain nombre de propriétés constitue donc une préoccupation pertinente et un défi de recherche intéressant..

1.3 Objectifs de recherche

L'objectif principal de cette thèse est de proposer une architecture répartie de découverte des services Web géolocalisés permettant de découvrir les services Web géographiquement les plus proches du contexte de localisation d'un client mobile. De manière plus spécifique, les objectifs visés sont:

1. de proposer un modèle architectural de découverte des services Web géolocalisés qui permet de:
 - publier les services Web géolocalisés ;

- rechercher un service Web géolocalisé (avec ou sans QoS) le plus proche du contexte de localisation d'un client mobile ;
 - coordonner la migration d'un service au serveur d'applications géographiquement le plus proche du contexte de localisation d'un client mobile ;
 - proposer un mécanisme de collecte de données de QoS (bande passante et facteur d'utilisation d'un serveur d'applications) pour un service donné et dans un domaine spécifique contrôlé par un serveur de découverte de services ;
 - sélectionner un serveur d'applications sur la base des paramètres de QoS préalablement collectés (bande passante et facteur d'utilisation d'un serveur d'applications) ;
 - définir un modèle de localisation thématique des services géolocalisés.
2. proposer et évaluer l'architecture fonctionnelle proposée.

L'architecture proposée sera assujettie aux contraintes non fonctionnelles d'expansion et de coordination de la migration de services. L'expansion réfère à la possibilité d'accroître le nombre de nœuds dans la topologie, advenant une augmentation du nombre de clients. La contrainte de coordination d'une migration de service est une contrainte de temps limite que nous imposons au RTT (*Round Trip Time*) moyen de l'opération de coordination d'une migration de service afin de limiter le surcoût de temps induit par le contrôle de la migration de service et d'évaluer la fonctionnalité de coordination. Nous fixons cette valeur limite du RTT moyen de la coordination de migration de service à 30 millisecondes.

Nous formulons comme hypothèses l'existence d'une architecture LCS de base offerte par l'opérateur réseau et utilisons le protocole MLP (*Mobile Location Protocol*), recommandé par le groupe LIF (*Location Inter-operability Forum*) [50], comme protocole d'interopérabilité avec un serveur de position LCS.

1.4 Esquisse méthodologique

L'approche utilisée pour atteindre les objectifs projetés repose essentiellement sur l'analyse de la découverte des services Web dans un contexte de mobilité du client. Dans cette optique, toutes les entités qui interagissent avec le système sont découvertes, les protocoles de communication sont résolus, et une organisation appropriée des composants est effectuée. Nous supposons que UDDI est le protocole de base de découverte des services Web et nous adjoignons à ce protocole les fonctionnalités liées à la mobilité du client. Nous considérons MLP comme protocole d'interopérabilité de localisation des mobiles, auquel nous adjoignons des méthodes de localisation des groupes de mobiles en factorisant les méthodes de localisation sur une base thématique. Pour factoriser thématiquement les méthodes de localisation, nous avons établi une typologie des services géolocalisés connus dans la littérature. Nous distinguons principalement 4 catégories de services: informationnels, suivi des traces d'un objet mobile, gestion de ressources et navigationnels [83]. Cette approche nous conduira à définir les méthodes qui décrivent le comportement du système (rechercher un service Web avec ou sans QoS, publier un service Web, coordonner la migration d'un service Web).

Pour collecter les données de QoS, nous allons distinguer deux catégories de données de QoS: les données statiques et les données dynamiques. Les paramètres statiques de QoS seront obtenus lors de la publication des services par un fournisseur autorisé. On distingue comme paramètres statiques : le coût du service par domaine, la bande passante minimum (BW_min), la bande passante maximum (BW_max), la taille maximum des données de service (SDU_smax), le délai de propagation minimum (PPD_min) et le délai de propagation maximum (PPD_max). Les paramètres dynamiques de QoS (bande passante du réseau (BW), le facteur d'utilisation d'un SAS (UF)) sont obtenus dans un domaine donné et pour un service Web spécifique, par le serveur découverte des services GLWSM dudit domaine, qui formule une requête de soumission de QoS aux SAS de son domaine implémentant le service en question.

Les méthodes décrites conduiront à la définition d'une architecture fonctionnelle découpant les fonctionnalités de l'architecture en modules ou composants logiciels. Pour analyser l'architecture fonctionnelle, la méthodologie « top-down », consistant à étudier les cas d'utilisation du système, est utilisée pour découvrir les composants logiciels et les organiser. Cette approche utilise le paradigme orienté objets et permet de découvrir les classes, de représenter les séquences d'appels de méthodes conduisant à la réalisation d'un cas d'utilisation donné. Elle permet aussi de montrer le choix des technologies logicielles utilisées pour résoudre le problème étudié.

1.5 Principales contributions et originalité

Cette thèse résout trois problèmes inhérents à la découverte des services Web géolocalisés dans les réseaux mobiles: découvrir les services Web géolocalisés avec ou sans QoS quand un client mobile est dans une région donnée, continuer l'exécution d'un service au serveur d'applications le plus proche quand un client sort de la région géographique couverte par le serveur d'applications courant, et interroger un groupe de mobiles lié par un thème.

Afin de découvrir les services Web géolocalisés, nous proposons dans nos travaux un nouveau modèle architectural qui tient compte de la mobilité du client et détermine le serveur d'applications le plus proche couvrant la position géographique d'un client mobile. Ce modèle architectural repose sur une topologie hiérarchique à deux niveaux: le niveau racine et le niveau des feuilles. Le niveau racine possède un nœud unique appelé nœud racine. Le niveau des feuilles est composé de plusieurs nœuds feuille repartis dans le système. Chaque nœud feuille couvre certaines zones de localisation du réseau mobile d'un opérateur et est lié à un ensemble de serveurs d'applications. Cette solution architecturale permet de contrôler la cohésion des informations des services Web publiés dans le système en coordonnant la publication des services par le biais du nœud racine. Le service publié au nœud racine est par la suite propagé aux nœuds feuille où le service sera offert en utilisant une queue de publication

écoutée par ces nœuds feuille. Une conséquence logique de cette solution est qu'elle limite le flux d'échange de messages entre différents nœuds lors de la publication d'un service, comparativement aux modèles existants qui utilisent une topologie hiérarchique à trois niveaux minimum. L'autre apport de la solution architecturale proposée est la consistance des données de QoS (bande passante et facteur d'utilisation d'un serveur d'applications) collectées et propagées dans le système. En effet, pour permettre à un nœud de découverte des services de sélectionner un serveur d'applications lié à un autre nœud de découverte de services, nous définissons et utilisons une relation de visibilité permettant à partir d'un nœud donné offrant un service S de sélectionner tous les nœuds qui offrent le même service S . Cette relation de visibilité est utilisée par un nœud donné pour propager les données de QoS collectées dans son domaine et pour un service S à d'autres nœuds qui offrent le même service. Nous utilisons une queue de propagation des informations de QoS pour rafraîchir régulièrement les données de QoS collectées à un nœud spécifique. La solution proposée limite également le flux d'échange de messages entre différents nœuds lors d'une propagation des données de QoS, comparativement aux modèles existants qui utilisent une topologie hiérarchique à trois niveaux au minimum.

Pour collecter les données de QoS, nous avons défini un mécanisme de collecte de données dynamiques de QoS (bande passante, facteur d'utilisation d'un serveur d'applications) qui permet de prélever dans un domaine contrôlé par un serveur de découverte de services, la bande passante et le facteur d'utilisation d'un serveur d'applications appartenant à un fournisseur de services. Le principe consiste globalement à envoyer une requête de collecte de données de QoS à un serveur d'applications qui la redirige au MAP (Mobile Anchor Point) associé au domaine contrôlé par le serveur de découverte de services, afin de mesurer la bande passante disponible dans le domaine. Quant au facteur d'utilisation, nous le déterminons en utilisant les fonctions natives du système d'exploitation. Le mécanisme de collecte de données de QoS proposé permet d'entretenir dans le système les données consistantes de

QoS utiles à la sélection du serveur d'applications en cas d'une recherche de service avec QoS ou d'une coordination de migration de service avec QoS.

Les services découverts ayant un caractère géolocalisé, leur exécution est effectuée à un serveur d'applications préalablement sélectionné et dont la portée d'exécution est associée à une région géographique spécifique. Il en résulte que, si un client mobile qui a un service en exécution sort de la région géographique du serveur d'applications courant exécutant le service, l'exécution du service subira une fin de session. Pour permettre une continuité d'exécution du service lors de la mobilité du client, nous proposons un nouveau protocole qui coordonne la migration de service du serveur d'applications courant au prochain serveur d'applications le plus proche couvrant la position actuelle du client mobile. À cette fin, le mécanisme de coordination de migration de service proposé traque continuellement la position géographique du client mobile au nœud de découverte de services couvrant la position actuelle dudit client et informe le serveur d'applications en cours d'exécution de migrer le service au prochain serveur d'applications, dans le cas où le client mobile sort de la région géographique couverte par le serveur d'applications courant.

En effectuant l'analyse des types d'applications géolocalisées qui peuvent être offerts, nous avons constaté que la localisation thématique des groupes de mobiles n'est pas prise en compte dans les méthodes de localisation courantes. Nous avons donc proposé une extension du protocole MLP (*Mobile Location Protocol*) afin d'y ajouter le caractère thématique de la localisation d'un groupe de mobiles liés par un thème. Nous appelons le protocole ainsi étendu MLPe (*Mobile Location Protocol Extension*). La localisation thématique nous a conduit à enrichir le protocole MLP d'une nouvelle structure de données (*Theme*, *ThemeMsid*) et des API (Application Protocol Interface) qui permettent d'interagir avec le serveur de position LCS. Nous démontrons également, par les résultats obtenus, que la localisation thématique a un temps de recherche de mobiles meilleur que la localisation ordinaire d'un groupe de mobiles qui consiste à envoyer une liste d'identifiants de mobiles (dont la position géographique est recherchée) au serveur de position LCS.

1.6 Plan de la thèse

La suite de ce document est organisée de la manière suivante. Le chapitre 2 présente la revue de littérature sur la découverte ou localisation des services. Le chapitre 3 décrit l'architecture de découverte des services proposée dans cette thèse, détaille les principes et requis de la conception de l'architecture proposée ainsi que le modèle de collecte et de propagation de données consistantes de QoS, enfin expose le fonctionnement du système y compris la localisation thématique (structure de données et APIs). Le chapitre 4 explique la démarche utilisée pour l'implémentation de cette architecture. Le chapitre 5 présente les résultats obtenus. Le chapitre 6, en guise de conclusion, fait une synthèse des travaux, en relate les limites et esquisse quelques avenues de recherches futures.

CHAPITRE II

DÉCOUVERTE ET DISTRIBUTION DES SERVICES

Traditionnellement, la découverte des services est un processus qui permet de découvrir la localisation d'une entité logicielle ou matérielle, en utilisant des ressources réseaux [37]. Ce processus favorise l'annonce de la présence des services dans le réseau, la découverte des ressources locales et distantes, l'adaptation à la mobilité dans le cas des clients mobiles, la description des capacités d'un service et la migration des services si nécessaire.

Du point de vue du client, la découverte des services se résume à résoudre un nom symbolique de service contre une valeur représentant son adresse. Cette adresse permet par la suite de lier le requérant au fournisseur dudit service. Le désir de localiser un service ou un objet se pose dès lors où le requérant ne possède pas suffisamment d'informations pour déterminer l'adresse de l'entité avec laquelle il souhaite communiquer. Dans le domaine de l'informatisation des données, le rôle de découvrir les services est confié à des serveurs qui enregistrent les services et permettent la recherche de ceux-ci. Ces serveurs sont généralement distribués à travers une topologie de déploiement pour tenir compte de l'expansion du nombre de clients et effectuer une couverture régionale de l'exécution du service [8], [43], [51], [54], [55], [97]. Les informations enregistrées sont généralement le nom d'un service, la description du service, le nom du propriétaire ou fournisseur du service, et les informations sur le chemin d'accès pour interagir avec le service [43], [81], [102]. Les protocoles de découverte des services dans les réseaux mobiles doivent tenir compte de la diversité des appareils portatifs, de l'expansion du nombre d'utilisateurs, de la mise à jour des services, de la sélection des services sur des critères de qualité et de la sécurité des données de localisation d'un client. Dans ce chapitre nous définirons les termes et concepts

nécessaires à la découverte des services, puis ferons une revue de littérature pertinente du domaine étudié.

2.1 Définitions et concepts de base

La découverte d'un service met en relation trois acteurs principaux : le client, le fournisseur et le courtier ou distributeur des services. Le client utilise les fonctionnalités du système dans le but de rechercher un service auprès d'un serveur de découverte de services et de communiquer avec le serveur d'applications d'un fournisseur désiré. Le distributeur ou courtier de services offre les ressources matérielles et logicielles permettant de stocker, d'organiser, de rechercher, de publier, de modifier ou de supprimer un service. Le fournisseur de services implémente la logique des services recherchés par un client dans l'un de ces serveurs d'applications. Il publie ses services à un ou plusieurs serveurs de découverte de services et les met à jour. Il est responsable de la maintenance du code exécutable des services publiés.

Dans le cadre de cette thèse, les services publiés au serveur de découverte de services sont des services Web. On définit un service Web [59] comme un service qui utilise le réseau Internet en faisant appel à des méthodes publiées ou tâches d'affaires d'une application décrite dans un format de spécification technique. Le format de spécification généralement utilisé pour les services Web est le WSDL (*Web Service Description Language*).

2.1.1 Le protocole WSDL

Le protocole WSDL est un langage issu de XML qui permet une communication entre applications développées sous différents langages (Java, C++, etc.) et sous différentes plates-formes (Windows, Unix, Linux, etc.). Fonctionnellement, il décrit un service encapsulé dans une interface publiée et spécifie comment l'invoquer. Il possède des balises nécessaires à la description d'une interface publiée via le Web et est composé des éléments décrits au Tableau 2.1.

Tableau 2.1 Éléments d'un document WSDL

Élément	Informations contenues
<portType>	Il est l'équivalent d'une interface. Il contient une ou plusieurs opérations (méthodes) publiées du fournisseur des services. Une interface publiée est un élément du service web. Chaque opération contient les noms des messages d'entrée et de sortie de l'opération.
<message>	Il définit les messages ou paramètres d'une méthode publiée. Ces messages sont échangés lors d'une communication Web. On distingue des messages d'entrée et de sortie.
<types>	Les types de données complexes utilisés dans les méthodes publiées. Cet élément utilise la structure XML pour définir les types complexes.
<binding>	Il permet de donner les détails de l'association d'une interface décrite dans l'élément portType. Il décrit le protocole de communication utilisé par le service Web et le format des messages d'un port.
<import>	Permet de séparer les différents éléments d'une définition de service en plusieurs documents indépendants, qui peuvent être importés si besoin est.
<serviceType>	Il regroupe l'ensemble des interfaces publiées de l'application.
<service >	Il définit les adresses permettant d'invoquer un service donné. Il se comporte comme une collection de ports. Un port est un représentant de l'élément portType et possède une adresse de localisation.
<definitions>	L'élément racine du document, il donne le nom du service, déclare les espaces de noms utilisés, et contient les éléments du service.

Une interface publiée est une collection de méthodes publiques regroupées logiquement dans un document ou objet n'ayant aucune implémentation de leurs comportements. La classe qui implémente une interface publiée développe toutes les méthodes qu'elle encapsule. Les méthodes d'une interface publiée sont accessibles à un usager autorisé utilisant une machine distante ou locale. Le langage WSDL spécifie le format des messages échangés lors d'une requête, ainsi que l'adresse de chacune de ces méthodes et le protocole de transport utilisé tel que http (*HyperText Transport Protocol*) ou JMS (*Java Message Service*).

Pour rendre les services Web découvrables, les fournisseurs de services enregistrent et publient leurs services Web ainsi que les documents WSDL associés dans un registre dédié appelé UDDI.

2.1.2 Le protocole UDDI

Le protocole UDDI est un protocole permettant de découvrir les services Web. Il est associé à un registre portant le même nom UDDI et dont le rôle est de conserver les données persistantes des services publiés. Une représentation UML de la structure de données UDDI [25] est illustrée à la Figure 2.1.

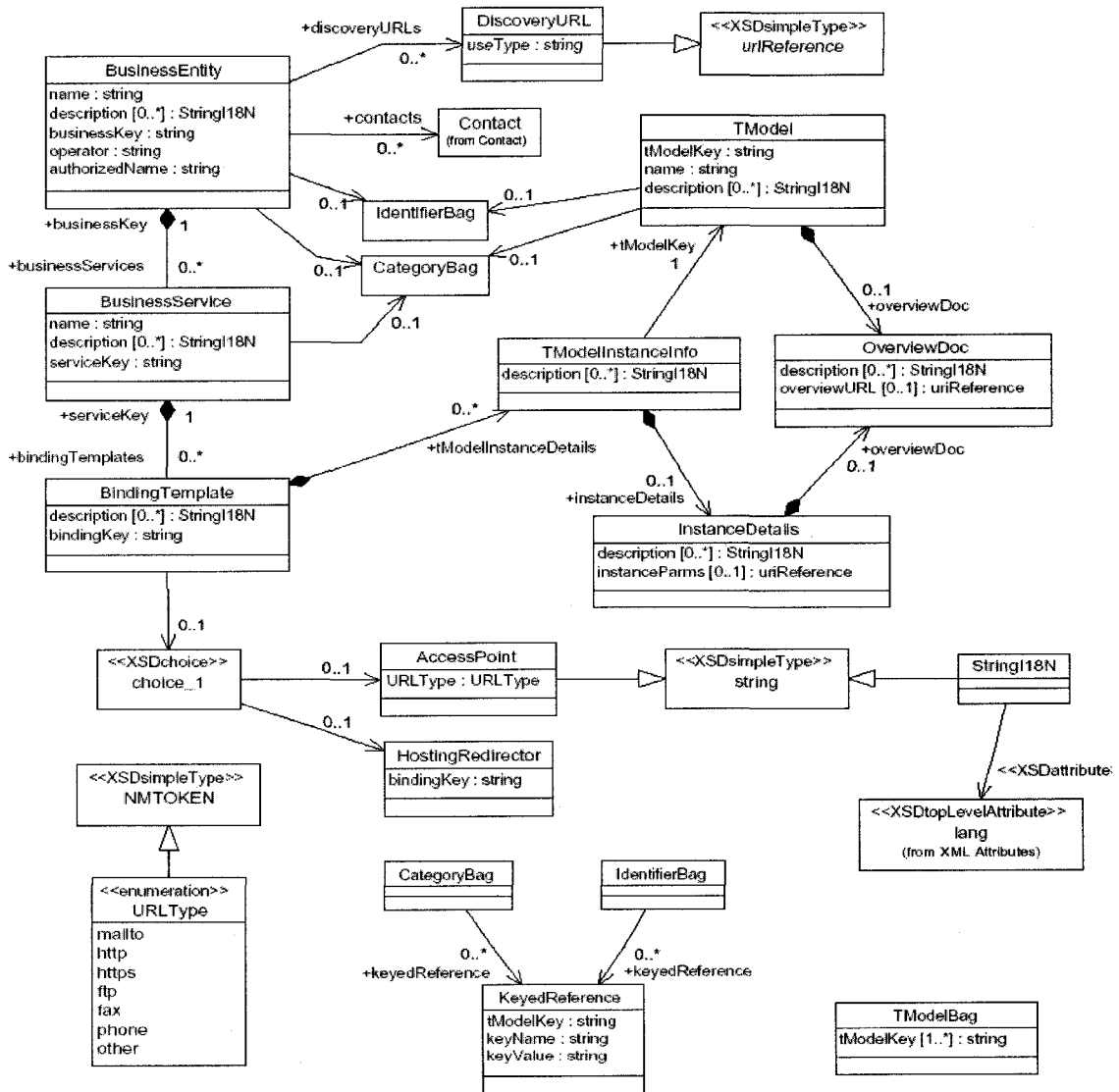


Figure 2.1 Diagramme de classes UML de la structure de données UDDI

Le registre UDDI est un annuaire dont le but est de publier et d'enregistrer les services Web, de décrire chaque service Web ainsi que la manière d'y accéder dans un document WSDL associé. Un registre UDDI a un format spécification qui définit les éléments de la structure de données conservée dans le registre ainsi que les API pour interagir avec le registre. Ce format de spécification utilise des balises standards XML. Le registre UDDI remplit les fonctions de pages blanches (*recherche par le nom*), jaunes (*recherche par catégorie*) et vertes (*recherche par type de produit*).

Le registre UDDI est composé principalement de quatre entités : *businessEntity*, *businessServices*, *bindingTemplates* et *tModels*. L'entité *businessEntity* décrit une entreprise ou une organisation. Elle contient les informations sur l'organisation, les services offerts par celle-ci et les nœuds de publication des services. Dans un schéma XML, l'entité « *businessEntity* » enveloppe toutes les autres entités. Chaque *businessServices* décrit l'ensemble des services offerts par une organisation. L'entité *bindingTemplates* d'un « *businessService* » spécifique décrit les liens URL des nœuds d'accès d'un service WEB et fournit un mécanisme d'association à l'élément « *tModels* » d'un service. L'entité *tModels* contient les informations sur le modèle du document WSDL et décrit une interface des services WEB publiés.

Le protocole UDDI offre deux grands groupes de API: les API de recherche et les API de publication. Les API de recherche permettent de rechercher dans le registre une entité (*businessEntity*, *businessServices*, *bindingTemplates*, *tModels*) ou un élément de l'une de ces entités. Par contre, les API de publication permettent de publier et d'enregistrer un élément d'une entité ou une entité (*businessEntity*, *businessServices*, *bindingTemplates*, *tModels*) dans le registre. Pour faire appel aux services Web publiés dans le registre UDDI, le protocole d'interopérabilité SOAP (*Simple Object Access Protocol*) est utilisé pour formater les messages envoyés par un requérant au serveur UDDI.

2.1.3 Le protocole SOAP

Le protocole SOAP (*Simple Object Accces Protocol*) est un protocole issu de XML et permet d'appeler à distance les services Web implémentés dans un serveur Web. Deux documents sont générés pour un appel de service Web distant: un message SOAP requérant et un message SOAP de réponse. Les documents générés sont illustrés aux Figures 2.2 et 2.3. Les messages SOAP d'envoi et de réponse sont structurés et font appel aux procédures à distance RPC (*Remote Procedure Call*) pour interagir avec les services Web [36], [67], [79]. Le protocole SOAP utilise principalement le protocole HTTP pour transporter les messages.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.stock.org/stock">
    <m:GetStockPrice>
      m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Figure 2.2 Requête SOAP

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.stock.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>
</soap:Envelope>
```

Figure 2.3 Réponse SOAP

Un message SOAP peut être vu comme un document XML constitué de trois parties : l'enveloppe (*soap-Envelope*), les règles d'encodage des types de données (*soap:encodingStyle*) et l'invocation des méthodes (*SOAP-RPC* par défaut). L'enveloppe est composée de deux sous-éléments: l'entête et le corps du message (*soap:Body*). L'entête SOAP est un élément transactionnel et contient un ensemble de directives sur les attributs, tels que les noms des espaces ou répertoires utilisés, l'autorisation, l'encryption ou la persistance des messages [79]. Les données envoyées par une

application sont transformées et encodées dans des types de base prédéfinis ou des types complexes XML avant d'être transmis à un receveur [67]. La description des méthodes à invoquer par le biais du protocole RPC se fait dans le corps du message en utilisant deux éléments XML : un élément pour la requête et l'autre élément pour la réponse. Cette description contient les informations telles que le nom de la méthode, la signature de la méthode, l'URI (*Uniform Resource Identifier*) de la classe implémentant la méthode et le type d'objet retourné par la méthode.

Bien que très riche en fonctionnalités, le protocole SOAP ne possède pas des balises XML nécessaires à l'interopérabilité entre les serveurs d'applications et les serveurs LCS. Afin de permettre cette interopérabilité, le groupe LIF a défini un protocole de localisation des mobiles appelé MLP (*Mobile Location Protocol*) issu du langage SOAP.

2.1.4 Le protocole MLP

Le protocole MLP [52] est un protocole qui décrit les messages de requête sur la position des stations mobiles (téléphones mobiles, PDA, etc.) indépendamment de la technologie utilisée pour déployer le réseau mobile (UMTS, CDMA-2000, ou WGS-136). Il intègre trois couches principales : services, éléments et transport. La *couche de services* définit les services offerts. Les services sont classés en 3 catégories: base, avancés et autres. Les services de base sont issus des services de localisation définis et spécifiés par le groupe de recherche 3GPP (Tableau 2.2). Les services avancés et les autres services sont respectivement des services supplémentaires qui seront spécifiés dans des versions de spécification ultérieures et des services spécifiés par un autre protocole, conformément à la structure de MLP.

La *couche des éléments* contient des composants de services communs à la couche de services. Ces composants sont décrits de manière descriptive en utilisant les documents DTD (*Document Type Definition*). Cette couche contient sept éléments DTD:

Tableau 2.2 Description des services du protocole MLP

Services	Description
Service standard de localisation immédiat	C'est une requête de service standard variant selon les paramètres d'entrée. Ce service est utilisé quand une réponse de positionnement est immédiatement requise ou quand le traitement de la requête est asynchrone. Le service consiste aux requêtes et réponses suivantes : • Requête standard de localisation immédiat, elle débute par la balise <slir>; • Réponse standard de localisation immédiate, elle débute par la balise <slia>; • Rapport standard immédiat; elle débute par la balise <slirep>.
Service d'urgence de localisation immédiat	Ce service est utilisé spécialement afin d'avoir la position d'un abonné en cas d'un appel d'urgence. La réponse de positionnement est immédiatement requise. Le service consiste aux messages suivants : • Requête d'urgence de localisation immédiate, elle débute par la balise <eme_lir>; • Réponse d'urgence de localisation immédiate, elle débute par la balise <eme_lia>.
Service standard du rapport de localisation	C'est un service standard destiné aux abonnés ayant souscrit à la réception de leur propre localisation. • Le rapport standard de localisation débute par la balise <slrep>.
Service d'urgence du rapport de localisation	C'est un service d'urgence utilisé par l'opérateur réseau afin de localiser un abonné en détresse suite à un appel d'urgence. • Le rapport d'urgence de localisation débute par la balise <emerep>.
Service événementiel du rapport de localisation	Ce service événementiel est utilisé quand la position d'un abonné mobile doit être rapportée à intervalle régulier ou à l'occurrence d'un événement spécifique. Le service consiste aux messages suivants : • Requête événementielle rapportant le début de localisation, elle débute par la balise <tlrr>; • Réponse événementielle rapportant le début de localisation, elle débute par la balise <tlra>; • Rapport événementiel de localisation, elle débute par la balise <tlrep>; • Requête événementielle rapportant la fin de localisation, elle débute par la balise <tlrsr>; • Réponse événementielle rapportant la fin de localisation, elle débute par la balise <tlrsa>.
Autres services	Si un service n'est pas implémenté, le LCS retourne un message d'erreur générale. Les messages d'erreurs sont précédés par la balise <gem>.

- MLP_ID.DTD: définit l'identification des éléments. On distingue différents éléments tels que *msid* qui définit l'identité d'un usager mobile, *codeword* qui est une clé ou un mot de passe protégeant l'information sur la position du mobile ;
- MLP_FUNC.DTD: définit les fonctions des éléments ;
- MLP_LOCATION.DTD: définit la localisation des éléments ;
- MLP_SHAPE.DTD: définit la forme des éléments géométriques ;
- MLP_QOP.DTD: définit la qualité de la position obtenue des éléments ;

- MLP_GSM_NET.DTD: définit les paramètres du réseau mobile des éléments ;
- MLP_CTXT.DTD : définit le contexte des éléments.

La *couche de transport* définit comment les messages mis sous format XML sont transportés. Les protocoles de transport présentement supportés et utilisés par le protocole MLP sont : HTTP et SOAP.

Les Figures 2.4 et 2.5 illustrent respectivement une requête et une réponse sur la position des mobiles identifiés par *msids* [52]. Dans ces figures, *start-msid* et *stop-msid* indiquent le début à la fin des identifiants mobiles lus dans un intervalle donné (*msid_range*). La balise *<eqop>* permet de spécifier la précision sur la position et la balise *<geo-info>* indique le système de coordonnées à utiliser. La balise *<pos>* retourne la position d'un objet trouvé.

```

<slir ver="3.0.0" res_type="SYNC">
  <msids>
    <msid type="IPV4">93.10.0.250</msid>
    <msid_range>
      <start_msid>
        <msid>461018765710</msid>
      </start_msid>
      <stop_msid>
        <msid>461018765712</msid>
      </stop_msid>
    </msid_range>
  </msids>
  <eqop>
    <resp_req type="LOW_DELAY" />
    <hor_acc>1000</hor_acc>
  </eqop>
  <geo_info>
    <CoordinateReferenceSystem>
      <Identifier>
        <code>4004</code>
        <codeSpace>EPSG</codeSpace>
        <edition>6.1</edition>
      </Identifier>
    </CoordinateReferenceSystem>
  </geo_info>
  <loc_type type="CURRENT_OR_LAST" />
  <prio type="HIGH" />
</slir>

```

Figure 2.4 Requête standard de localisation immédiate

```

slia ver="3.0.0" >
  <pos>
    <msid>461018765710</msid>
    <pd>
      <time utc_off="+0300">20020623134454</time>
      <shape>
        <CircularArea srsName="www.epsg.org#4004">
          <coord>
            <X>301228.302</X>
            <Y>865633.863</Y>
          </coord>
          <radius>570</radius>
        </CircularArea>
      </shape>
    </pd>
  </pos>
  <pos>
    <msid>461018765711</msid>
    <pd>
      <time utc_off="+0300">20020623110205</time>
      <shape>
        <CircularArea srsName="www.epsg.org#4004">
          <coord>
            <X>781234.322</X>
            <Y>762162.823</Y>
          </coord>
          <radius>15</radius>
        </CircularArea>
      </shape>
    </pd>
  </pos>
  <pos>
    <msid>461018765712</msid>
    <poserr>
      <result resid="10">QOP NOT ATTAINABLE</result>
      <time>20020623134454</time>
    </poserr>
  </pos>
</slia>

```

Figure 2.5 Réponse standard de localisation immédiate

2.2 Découverte et distribution des services

Déterminer les ressources et les services répartis dans les réseaux informatiques nécessite des protocoles de découverte de services permettant de rechercher, publier, modifier, sélectionner et utiliser un service. Ces protocoles sont exécutés dans des serveurs de découverte de services distribués à travers le réseau. Les travaux pertinents dans ce domaine concernent les protocoles classiques, la découverte de services avec contrainte de QoS, la co-localisation, la facturation basée sur la localisation, la

modélisation de la mobilité et distribution des services, et la découverte des services avec contraintes de sécurité.

2.2.1 Les protocoles classiques

De nombreux protocoles de découverte des services ont été proposés et les plus cités sont : SLP, Jini, Salutation, UPnP, SDP, Bluetooth SDP, Globe, Rover et AMRB.

➤ *Le protocole SLP*

Le protocole SLP (*Service Location Protocol*) a été développé par le groupe IETF [15] et une version 2 est actuellement disponible [39]. SLP est adapté aux réseaux IP. Fonctionnellement, le protocole SLP utilise trois agents: un agent d'utilisateur *UA* (*User Agent*), un agent de service *SA* (*Service Agent*) et un agent de répertoire *DA* (*Directory Agent*) [39]. Quand un nouveau service doit être ajouté au réseau, le *SA* contacte le *DA* pour enregistrer ledit service. Un usager autorisé qui souhaite localiser un service, utilise un *UA* approprié pour interagir avec le *DA*. Après traitement par le serveur de découverte des services, le *UA* retourne à l'utilisateur l'adresse et d'autres caractéristiques du service désiré. Si le *UA* ou le *SA* n'a pas connaissance de l'adresse du *DA*, il utilise une méthode statique, active ou passive, pour découvrir le *DA*. La méthode statique consiste à utiliser le protocole DHCP (*Dynamic Host Configuration Protocol*) pour découvrir le *DA* [31], [65], [66]. Les serveurs qui offrent les fonctionnalités de DHCP retournent l'adresse des *DA* aux hôtes ayant formulés une requête de localisation des *DA*. Quand la méthode active est utilisée, les requêtes de localisation des *DA* sont formulées à un groupe «multicast» écouté par les *DA* du système de découverte des services. Un *DA* ayant reçu et traité la requête retourne au *SA* ou *UA* par communication «unicast» son adresse. Dans le cas d'une méthode passive, un *DA* envoie de manière périodique son adresse aux *UA* et *SA* de son sous-réseau. Les *UA* et *SA* qui reçoivent ces annonces les sauvegardent dans un espace mémoire alloué qu'ils consultent en cas d'établissement de connexion avec un *DA*. Il est important de souligner que le protocole

SLP peut fonctionner sans *DA*, auquel cas le *UA* utilise une communication «multicast» avec les *SA*, et les *SA* retournent le service désiré au *UA* par communication «unicast».

➤ ***Le protocole Jini***

Jini est un protocole élaboré par *Sun Microsystems* sur la découverte des services. Le développement de Jini a été réalisé en utilisant le langage JAVA. Chaque machine qui a Jini pour protocole de découverte des services, est dotée d'une machine virtuelle JAVA *JVM (Java Virtual Machine)*. Le fonctionnement de Jini est similaire à celui de SLP [85]. Tout comme SLP, il utilise trois acteurs: le client ou le réquérant du service, le serveur de recherche de services et le serveur du fournisseur de services. Dans Jini, le protocole de découverte de services est utilisé par le fournisseur et le client pour trouver le serveur de découverte de services. Le fournisseur de services [9] doit s'enregistrer au serveur de recherche en utilisant l'API *Join*. Le client formule sa requête de service au serveur de découverte de services en utilisant l'API *Lookup*. Le client et le fournisseur utilisent un adressage «multicast» vers les serveurs de découverte de services de leur sous réseau. En retour, les serveurs de découverte de services répondent par un adressage «unicast» et retournent un objet *Proxy* à l'initiateur de la requête. Le réquérant peut par la suite adresser les services du serveur via le Proxy. Le protocole RMI (*Remote Method Interface*) est le protocole d'interopérabilité utilisé entre le client (via l'objet *Proxy*) et le fournisseur de service.

➤ ***Le protocole Salutation***

Salutation est une autre approche classique de découverte des services. L'architecture de Salutation [73] a été conçue par un groupe appelé Salutation Consortium. Fonctionnellement, le protocole Salutation utilise un composant courtier de services *SLM (SaLutation Manager)* et un composant de transport *TM (Transport Manager)*. Un *SLM* offre des fonctionnalités d'enregistrement, de disponibilité et de recherche de services. Un fournisseur de services enregistre les caractéristiques d'un service par le biais d'un *SLM*. Les clients envoient leurs requêtes au *SLM* pour

rechercher un service. Le *TM* établit des canaux de communication fiables pour acheminer les données dans le réseau.

➤ ***Le protocole UPnP***

Le protocole UPnP (*Universal Plug and Play*) a été développé par *Microsoft Systems* pour découvrir les ressources à travers le réseau IP [91]. Il permet d'auto-configurer les machines, de découvrir et de contrôler les services. En effet, un serveur de découverte des services qui sollicite l'utilisation de UPnP doit préalablement avoir obtenu une adresse IP (en utilisant par exemple le protocole DHCP), diffusé les caractéristiques des services qu'elle offre afin de découvrir les services offerts par d'autres serveurs de découverte de services. Le protocole UPnP interagit avec le protocole SSDP (*Simple Service Discovery Protocol*) [100] pour découvrir les services. Le protocole SSDP utilise les protocoles HTTP et UDP pour transmettre les messages, annoncer la présence des services et les capacités d'un serveur à d'autres serveurs de découverte des services. Un serveur de découverte de services qui se joint au réseau envoie un message «multicast» destiné à tous les autres serveurs de découverte de services déjà connectés dans lequel il annonce les services qu'il offre et reçoit en retour une réponse «unicast» des services offerts par les autres serveurs. La description des services se fait par le biais d'un document XML.

➤ ***Le protocole Bluetooth***

Bluetooth est une technologie sans fil qui permet de créer des réseaux personnels sans fil ayant une bande de fréquence de 2.4 GHz. Ces réseaux ne nécessitent pas d'autorisation d'utilisation et ont une portée de 10 mètres [53]. Ils sont composés de périphériques nomades telles que les téléphones portables, les assistants personnels et les ordinateurs portables. Bluetooth utilise le protocole SDP (*Service Discovery Protocol*) pour localiser les ressources d'un réseau constitué. Le protocole SDP offre aux applications clientes les moyens de découvrir l'existence des services fournis par les applications serveurs ainsi que les attributs de ces services (par exemple le type ou la

classe du service offert et l'information sur le protocole nécessaire pour utiliser le service) [16], [58]. Le protocole SDP établit un canal de communication entre un serveur SDP et un client SDP. Le serveur maintient une liste de services publiés. Chaque service publié est lié à un seul serveur. Le client peut récupérer l'information d'un service publié et maintenu au serveur SDP en émettant une requête SDP de recherche de service. Afin de communiquer avec un fournisseur de services, le client établit au préalable une connexion avec le serveur SDP. Généralement, un client SDP recherche les services sur la base de certaines caractéristiques de services désirées. Cependant, il est parfois désirable d'avoir plus de détails sur un service particulier, auquel cas on utilise la technique de navigation (*browsing*).

➤ ***Le protocole Globe***

Les chercheurs du projet Globe ont développé une architecture appelée «Globe Location Service» [7], [43]. Cette architecture permet de localiser les usagers et les services mobiles. Globe définit et implémente des objets distribués dans une topologie hiérarchique en arbre. Lorsqu'un objet est recherché dans un nœud de la feuille de l'arbre d'une zone de localisation donnée, l'objet est soit présent ou absent. S'il est absent, l'objet Globe a une adresse de contact enregistré au nœud interrogé. Cette adresse est en réalité un pointeur de redirection vers le nœud parent de l'arbre. La procédure est répétée jusqu'à résolution de la requête. Globe propose une nomenclature géographique pour reconnaître les nœuds. Elle est semblable à l'architecture de gestion de mobilité des réseaux GSM (par le biais des bases de données distribuées), mais n'est pas adaptée au contexte de la connaissance préalable de l'utilisateur pour octroyer continuellement le service le plus proche.

➤ ***Le protocole Rover***

Rover [56] est une architecture alternative, fournissant aux applications mobiles la possibilité de transférer les objets relogeables dynamiquement RDO (*Relocatable Dynamic Objects*) du serveur au client (vice versa). Rover suppose que le client mobile a

suffisamment de ressources mémoire pour continuer l'exécution d'un service en cas de perte de connexion avec le réseau. Rover utilise une queue RPC appelée QRPC (*Queued Remote Procedure Call*), qui est un protocole de communication asynchrone. Cette architecture ne permet qu'une communication entre un client et un serveur tout en préservant le contexte d'exécution; l'échange entre le client et plusieurs serveurs est proscrit. Rover n'est pas adaptée au contexte de la connaissance préalable de la position du mobile et ne garantit pas l'exécution du service auprès d'un serveur le plus proche le long de la trajectoire d'un client mobile.

➤ ***Le protocole AMRB***

Un groupe de chercheurs de l'Université de KEIO [41] a proposé une architecture de localisation des objets ou des codes mobiles appelée AMRB (*Application Module Request Broker*). Cette architecture est composée de plusieurs serveurs de découverte des services distribués dans une topologie en arbre. Un nœud de découverte de services AMRB permet aux clients de découvrir le module ou code d'une application mobile *AM* (*Application Module*). Un nœud AMRB est composé principalement de deux composants logiciels: le gestionnaire de localisation et le gestionnaire de migration. Les gestionnaires de localisation et de migration permettent respectivement de localiser un module *AM* et de faire migrer un module *AM*. Si un *AM* est déplacé dans une zone de localisation couverte par un nœud AMRB, le système enregistre le module *AM* dans le registre associé à ce nœud, puis un adressage «multicast» est effectué sur l'ensemble des nœuds de la topologie afin de mettre à jour les informations de localisation de *AM* dans leurs registres respectifs. Lors d'une recherche d'un module *AM* si un nœud ne possède pas l'information sur la localisation du module, il re-route la requête vers l'adresse du nœud de redirection du module recherché dont la valeur indique l'adresse du nœud où se trouverait le module *AM*. Si *AM* n'est pas présent au nœud où il a été redirigé, ce dernier recherche dans son registre l'adresse de redirection qu'il a enregistré avant son déplacement au prochain nœud de redirection. Cette recherche est faite nœud après nœud jusqu'à découverte du module *AM*. Ce travail, bien

que très intéressant, n'utilise pas le contexte de localisation de l'utilisateur pour offrir le service mais résout plutôt la détermination de l'emplacement du code mobile.

2.2.2 Découverte des services avec les paramètres de QoS

Il a été proposé dans [30] une architecture de découverte de services avec contraintes de QoS. Fonctionnellement, l'architecture proposée répartit les serveurs de découverte des services dans une topologie hiérarchique en arbre où les serveurs situés aux feuilles de l'arbre sont les plus proches du client. Si un service n'est pas présent à un nœud donné, la recherche est redirigée vers le nœud parent immédiat. Afin de réduire le temps de recherche, une cache d'informations est créée au niveau de chaque nœud dans le but de conserver les informations sur les services recherchés et non présents à un nœud spécifique. Dans cette architecture, chaque serveur de fournisseur de services est associé à un serveur de découverte de services situé à la feuille de l'arbre. Pour rechercher les services avec contrainte de QoS, les auteurs proposent une extension de cette cache afin de conserver les informations sur la qualité de service obtenues par un client pour un serveur d'applications spécifique. Ainsi, lorsqu'un client négocie et obtient une QoS pour un service S localisé dans un domaine D , il transmet par la suite les informations de QoS obtenues à son serveur de résidence de découverte de services qui vérifie la présence effective du service dans la cache et met à jour les informations reçues, puis retransmet le message à son nœud parent de découverte de services. La procédure de mise à jour est répétée jusqu'au sommet de l'arbre. Pour tenir compte de la fiabilité des informations de QoS dans la cache, on adjoint à ces paramètres une durée de vie. Ainsi, plus les informations de QoS sur un service donné sont fréquemment mises à jour par les clients rattachés au domaine de résidence d'un serveur découverte de services, plus le service est fiable. Le problème dans cette proposition est la non fiabilité et l'inconsistance des données de QoS dans la cache. En effet, si la fréquence de mise à jour n'est pas régulière, la recherche avec QoS d'un service ne se fera non pas sur un serveur mais sur plusieurs serveurs de découverte de services avec le risque que les informations de QoS de la cache soient inexploitable car non fiables ou inconsistants.

D'autre part, compte tenu du modèle hiérarchique (à plusieurs niveaux) de la répartition des serveurs de découverte des services, toute défaillance qui survient lors de la mise à jour des informations de QoS sur un serveur découverte de services *SDS* est répercutée à tous les nœuds parents de *SDS*.

Dans [61], une architecture de découverte de services avec QoS est décrite. L'architecture possède deux composants principaux: gestion de ressources avec QoS et gestion des services avec QoS. Dans la gestion des services avec QoS, pour découvrir le service avec QoS, les clients transmettent à leur serveur de résidence de découverte des services (qui utilisent des protocoles tels que SLP ou Jini), la QoS obtenue avec un serveur d'application. Le procédé détaillé de la rétroaction de QoS obtenue au serveur de découverte de services n'y est pas décrit. Mais, les auteurs proposent que l'on utilise un processus semblable à celui proposée dans [30], pour stocker les données de QoS obtenues dans une base de données. Ainsi, à la réception d'une requête de recherche de services avec QoS, le serveur de découverte de services vérifie si les paramètres de QoS requis sont rencontrés. Si la comparaison est validée, le système renvoie l'adresse de serveur d'applications choisie. Puis, le client appelle le service approprié du serveur d'applications sélectionné. Quand la connexion est établie entre le client et le serveur d'applications, ce dernier charge le courtier de ressources (élément de la gestion des ressources avec QoS). Le courtier des ressources négocie les ressources réseau en utilisant comme paramètres initiaux, les paramètres de QoS requis par le client. Le problème avec cette approche est que la sélection du serveur d'application est effectuée sur la base des informations de la QoS observée par les clients et dépend de la fréquence de rétroaction ou de mise à jour de la QoS au serveur de découverte des services. Par conséquent, si la fréquence de mise à jour de la QoS n'est pas régulière, les informations de la QoS deviennent non fiables et inconsistantes.

Une architecture alternative de découverte des services avec contraintes de QoS a été proposée dans [4], [5]. Dans leurs travaux, les auteurs proposent une architecture appelée G-QoSM (*Grid Quality of Service Manager*). L'objet QoS est scindé en trois éléments: la QoS du serveur d'application d'un fournisseur, la QoS du serveur intergiciel

et la QoS des ressources réseau. La qualité de service du serveur d'applications permet de spécifier les paramètres de performance tels que la disponibilité, l'accessibilité, la fiabilité ou le niveau de sécurité. La qualité de service du serveur intergiciel permet de spécifier les paramètres tels que l'adresse du serveur intergiciel, le nombre de serveurs à allouer (si traitement parallèle) ou l'espace mémoire à réserver. La qualité de service des ressources réseau permet de spécifier les paramètres tels que l'adresse de l'expéditeur, l'adresse du receveur, la bande passante ou le pourcentage toléré de perte de paquets. Fonctionnellement, G-QoSM a trois modules essentiels: AQoS (*Application QoS*), RM (*Ressource Manager*) et NRM (*Network Ressource Manager*). Le composant logiciel AQoS interagit avec le client et les deux autres composants (RM et NRM) afin de découvrir les services ayant les contraintes de QoS requises dans la requête d'un client, coordonne l'allocation des ressources nécessaires avec les deux gestionnaires de ressources RM et NRM, puis gère l'exécution du service. Le composant NRM a pour charge de gérer, réserver les ressources réseau nécessaires à l'exécution d'un service. Il agit comme un *proxy* et fait appel aux protocoles RSVP et DiffServ pour garantir la QoS au niveau Réseau. Le composant RM interagit avec une base de données UDDI qui conserve les services publiés et leurs qualités de service associées. Il permet donc de découvrir, de publier ou de modifier les services. Il réserve aussi les ressources requises par un client au serveur de découverte des services afin d'exécuter un service spécifique. Cette architecture de découverte des services n'est toutefois pas adaptée au contexte de la mobilité et des services géolocalisés. Dans [44], une approche similaire à [4], [5], basé sur l'utilisation d'un *proxy* pour négocier la QoS est proposée. Cette approche sélectionne le serveur d'applications sur la base de la QoS statique (par exemple, coût d'un service) et fait entièrement abstraction de la QoS dynamique (par exemple, bande passante) dans le processus de sélection du serveur d'applications.

2.2.3 Co-localisation service/usager, facturation dépendante de la localisation

La co-localisation des services et de l'utilisateur est une nouvelle avenue de recherche qui consiste principalement en un flux poussé des services quand le client se

trouve dans une zone de localisation d'un service donné [24], [57]. Le serveur de découverte des services a une base de connaissances des habitudes du client et génère des envois de notification aux serveurs d'application si la co-localisation est établie.

Certains travaux [64] sont effectués présentement sur les modèles de facturation basés sur le contexte de localisation de la position du mobile. Les auteurs proposent une architecture appelée RCSPM (*Reconfiguration Control and Service Provisioning Manager*). Cette architecture permet de découvrir les services dans un contexte de la mobilité du client. En effet, le serveur de découverte des services traque la position géographique des clients et permet au requérant de service de sélectionner l'adresse du serveur d'applications sur la base du coût du service. Les fournisseurs publient leurs services dans un document XML en mentionnant le coût horaire du service dans un domaine administratif donné. Cependant, l'exécution rapprochée et la maintenance du service ne sont pas intégrées dans leurs objectifs.

2.2.4 Modélisation de la mobilité et distribution des services

La gestion des objets mobiles dans les bases données est nécessaire pour les applications de suivi ou de trace des mobiles. Ces applications utilisent la position d'un objet ou d'un groupe d'objets mobiles calculée par le serveur de localisation de l'opérateur réseau sollicité (par exemple, Bell mobilité Canada). Dépendamment des désirs du client, ces données peuvent être fournies périodiquement au serveur d'applications qui les sauvegarde dans une base de données et les restitue par la suite au requérant de service. Ces actions génèrent un flux de données entrant et sortant de la base de données. La gestion efficace de ces données requiert une modélisation des objets en mobilité en tenant compte de leurs aspects spatial et temporel [84], [98].

Des modèles de représentation des objets mobiles et l'optimisation des techniques d'indexage des requêtes spatio-temporelles ont été développés par différents chercheurs [19], [33], [34], [47], [68], [72], [74], [89], [95], [101]. Certains d'entre eux [74] ont proposé de formaliser les requêtes d'interrogation en utilisant le langage LDQ (*Location Dependent Query*) qui tient compte des notions telles que «le plus proche» ou

« dans un rayon » pour interagir avec les bases de données. Des propositions ont été faites sur le choix des attributs à mettre à jour, à insérer ou à supprimer de la base de données. Toutefois, les modèles proposés ne tiennent pas compte de la composante thématique de l'entité géographique qui réduirait le flux d'informations mis sur le réseau par les applications.

Afin d'optimiser le volume d'informations mis sur le réseau pour ou par les clients mobiles (réduire les délais de transmission et d'obtention de données entre les différents entités), des travaux ont été effectués sur les petits déplacements, la capacité et la puissance limitées des appareils mobiles. Il a été proposé un ajout des paramètres décrivant les nombres minimum et maximum de mises à jour par appareil, la période minimum de transmission entre deux mises à jour de la localisation et la variation de distance minimale à considérer pour transmettre une mise à jour [19]. D'autres chercheurs [101] ont proposé que le calcul de la position des usagers mobiles dans les réseaux de prochaine génération soit effectué par le terminal pour éviter les délais liés à la synchronisation des signaux de la triangulation des stations de base ou des équipements de mesure de localisation du réseau LMU (*Location Measurement Unit*). Dans leur approche, chaque unité mobile a un agent de message *MA* qui se déplace vers une station de base. L'agent *MA* contient les informations sur le type d'algorithme et la précision de mesure désirés. La station de base génère un agent de géolocalisation *GA* qui permet de déterminer la position d'un client et le retourne à l'unité mobile.

D'autres chercheurs [72], [76], [98] ont proposé de créer une cache des informations obtenues du serveur de position réseau dans le serveur d'applications. Les requêtes du client qui peuvent être résolues dans la cache ne seront pas transmises au serveur de position. La cache utilise des fonctions d'estimation de la position tenant compte de la dernière position et de la vitesse du mobile obtenues du serveur de positions. Après une période de temps, le serveur de positions transmet les informations réelles de la position au client pour une mise à jour [72]. Des travaux récents montrent que la cache est de plus en plus utilisée pour le partage de l'information de la position avec d'autres applications [56], [96]. En effet, ils ont proposé un *middleware* avec un

agent courtier de localisation. L'agent courtier négocie avec le propriétaire de l'unité mobile le partage de sa position avec d'autres applications intéressées.

Le partage de l'information est aussi abordé dans des projets très connus [56] : Odyssey, Bayou et Xmiddle. Odyssey offre un modèle d'adaptation collaboratif où le moniteur de référence d'accès aux données est le système opérationnel; Bayou permet une gestion de conflits grâce à une collaboration des serveurs d'applications répliquées; Xmiddle hiérarchise les serveurs d'applications partageant les données en utilisant XML.

Les travaux sur la cache d'informations sont intimement liés aux travaux de la gestion de distribution des services qui permettent de contrôler de manière cohérente la répartition des services de l'application. La distribution des services rapproche géographiquement le service à l'utilisateur afin d'interagir avec le serveur d'applications pour un abonnement, de rechercher un service, de publier, d'enregistrer ou de notifier un service. Elle pose la question où, quand et comment distribuer les services aux serveurs d'applications. Elle coordonne les activités échangées avec d'autres types de serveurs impliqués (par exemple, le serveur de positions LCS).

La distribution des services pose un problème d'interopérabilité. Ce problème est résolu dans le cadre de la localisation des services dans les réseaux mobiles avec le choix du protocole de description des requêtes de position MLP du groupe LIF par les plus grands opérateurs 3G (Alcatel, NOKIA, Ericsson, etc.). MLP est un protocole issu de XML et est implémenté en utilisant du SOAP-RPC sur HTTP.

Sur le marché, certains produits intergiciels fournissant la position des usagers mobiles ont été développés tels que *LocatioNet*, *OpenWave@LocationStudio*, *Nexus* ou *Cellpoint* [56], [84], [88]. Bien que très riches en fonctionnalités (sécurité, détermination de la position, facturation, authentification, autorisation), ces produits n'offrent pas des mécanismes de découverte de services et de migration d'exécution selon le contexte de localisation de l'utilisateur.

2.2.5 Découverte de services avec contraintes de sécurité

Lors d'une découverte des services géolocalisés, des données sensibles telles que la position et l'identifiant du client sont transportées dans le réseau. Parmi les travaux effectués dans ce domaine [27], [48], [104], l'architecture de sécurité de découverte des services SDS (*Secure Discovery Service*) est la plus citée [27]. Chaque serveur SDS reçoit périodiquement des annonces décrivant les services publiés par les serveurs d'applications des fournisseurs appartenant aux domaines qu'il couvre. Les services sont encryptés en utilisant une méthode d'encryption symétrique hybride. Elle consiste à encrypter le service publié par une clé secrète dont la valeur est elle-même encryptée par une clé symétrique partagée par un serveur SDS et un serveur d'applications de fournisseur de services. La communication entre un client et un serveur SDS, ou entre serveurs SDS, est conditionnée à une authentification mutuelle par échange de certificats. Pour tout client autorisé, le SDS associe un objet qui contient sa liste des capacités dans le système. Dans le cadre de la découverte des services géolocalisés, Zhu et al. [104] ont proposé un protocole de découverte avec contraintes de sécurité appelé Splendor. Tout comme le système SDS, Splendor est orienté intergiciel et utilise une authentification mutuelle entre entités en communication. Après l'authentification mutuelle, une clé de session est générée et est valable durant toute la session de la communication établie. Les clients et les services mobiles reçoivent périodiquement des certificats d'authentification des serveurs de découverte des services et des étiquettes d'informations les renseignant de leur localisation chaque fois qu'il change de zone de localisation. Les deux architectures SDS et Splendor utilisent un modèle d'autorisation d'accès basé le modèle DAC. En effet, le modèle DAC est de la forme:

< sujet > < liste des opérations permises > < objet cible >.

Ce modèle de contrôle d'accès n'est pas bien adapté à la localisation thématique car l'utilisation d'une méthode de l'application peut engendrer l'interrogation de la position géographique d'une tierce personne.

2.3 Synthèse des requis et défis de recherche

La définition et la conception d'une architecture de découverte des services géolocalisés sont importantes et essentielles pour les raisons de performance des applications distribuées. Certaines études ont été proposées et d'autres sont en cours pour la définition architecturale des services géolocalisés. Des notions architecturales telles que la qualité de service, la personnalisation des services, la distribution des services, la sécurité des services, l'interopérabilité des services sont invoquées dans les solutions proposées. Toutefois, ces architectures ne mettent pas l'accent sur la recherche et la maintenance d'exécution de services au serveur d'applications le plus proche du client mobile. Une architecture offrant ces fonctionnalités exige une définition d'un protocole de recherche, de publication ou de migration de services dans un contexte de la mobilité du client. La réalisation d'une telle architecture soulève des problèmes de distribution des services, de l'autorisation d'accès aux informations sensibles d'un client par un autre client et de la découverte des services avec contraintes de qualité de service.

2.3.1 Distribution des services

La distribution des services consiste généralement en deux opérations : la réplication et la partition [34]. Dans le cadre du développement des services géolocalisés, la distribution des services dans les serveurs de découverte des services est très importante pour rapprocher l'exécution d'un service de la position géographique réelle d'un client mobile. La distribution tient aussi compte de l'expansion du service. L'expansion du service reflète le fait que le système applicatif s'adapte à une croissance des usagers ou des objets [43]. Pour distribuer les services, une topologie adéquate de la répartition géographique des serveurs de découverte des services doit donc être définie, de même qu'une politique de publication des services. Distribuer les services revient aussi à interconnecter des entités (serveurs/clients) en communication. Réaliser ces notions exprimées simplement est complexe. Il est conseillé, pour résoudre le problème, de proposer un modèle topologique intégrant et solutionnant les problèmes soulevés.

L'avantage qu'on en tire est de factoriser du code, de diminuer le temps de recherche de service et du transport des données de traitement du client au serveur et vice-versa.

2.3.2 Factorisation des méthodes de localisation des usagers

La factorisation des méthodes de localisation des usagers est importante dans le sens que le serveur de découverte des services est destiné aux applications qui utilisent la position de l'utilisateur pour octroyer les services. Ainsi, ces applications implémentent des services permettant de localiser les usagers qui peuvent être factorisés par le serveur de découverte de services. Une étude doit donc être faite pour sélectionner les méthodes pertinentes.

2.3.3 La qualité des services

La proposition d'une architecture de découverte des services géolocalisés devrait prendre l'utilisation des paramètres de la qualité de services au niveau intergiciel pour sélectionner le serveur d'applications approprié. Les paramètres de QoS (bande passante, facteur d'utilisation d'un serveur d'applications) doivent être conservés au niveau de chaque nœud de découverte des services afin d'effectuer une sélection locale du serveur d'applications approprié. Certains chercheurs [30], [61] abordent le problème mais utilisent des données non consistantes dans la sélection du serveur d'applications du fait que la mise à jour des données de QoS dépend du nombre de clients actifs dans un réseau d'accès donné.

2.3.4 Sécurité des services

L'accès aux informations sensibles dans une découverte des services géolocalisés ne suit ni le modèle MAC (*Mandatory Access Control*) car les sujets (les clients mobiles) et les objets (les clients mobiles) ne peuvent être réduits à des compartiments classifiés et hiérarchisés en niveaux de sécurité, ni le modèle DAC (*Discretionary Access Control*) tel que expliqué précédemment. En effet, un client d'une application peut interroger la localisation d'un autre client. Pour éviter que les informations

sensibles telles que la position ou l'identité d'un client soient transmises sans contrôle lors d'une localisation thématique, le modèle d'autorisation d'accès aux méthodes de localisation des usagers doit être conditionné au fait que le client dont la position est requise appartienne à la liste des clients à laquelle le requérant a un droit d'accès.

CHAPITRE III

ARCHITECTURE DE DÉCOUVERTE DE SERVICES GÉOLOCALISÉS PROPOSÉE

De manière générale, la description d'une architecture permet de définir un système et son environnement (entités externes), de spécifier, d'organiser et d'échanger les informations nécessaires à l'atteinte des objectifs visés. Nous présentons dans ce chapitre la description du système GLWSA proposé dans cette thèse comme architecture de découverte de services Web géolocalisés. Nous définissons d'abord les principaux requis de l'architecture, puis les principes qui ont influencé le choix topologique effectué sur la répartition des serveurs de découverte des services. Ensuite, nous décrivons l'architecture et le formalisme mathématique du système GLWSA.

3.1 Principaux requis de l'architecture proposée

Afin de réaliser la conception architecturale, les objectifs visés et exprimés en langage naturel, ont été transformés en un ensemble de requis de conception qui détermine les fonctionnalités du système, et les contraintes de l'architecture sont soit associées aux performances désirées du système ou transformées en requis fonctionnels. On distingue deux familles de requis: requis fonctionnels et requis non fonctionnels. Les requis fonctionnels sont associés aux objectifs fonctionnels du système:

- proposer un modèle de découverte des services Web géolocalisés: ce modèle permet de publier les services Web géolocalisés, de rechercher un service Web géolocalisé sur la base du contexte de localisation de l'utilisateur et de lier le client mobile au serveur d'applications géographiquement le plus proche ;
- définir d'une topologie de répartition des nœuds ;

- proposer un modèle de coordination de la migration de services ;
- proposer un mécanisme de collecte de données de QoS (bande passante et facteur d'utilisation des serveurs d'application) pour un service donné et dans un domaine spécifique ;
- proposer un modèle de localisation thématique d'un groupe de mobiles.

Par définition, les objectifs fonctionnels décrivent le comportement du système. Pour les réaliser, nous les transformons en requis fonctionnels en utilisant un langage de modélisation. En ce qui concerne le système proposé, nous utiliserons le langage UML (*Unified Modeling Language*) pour analyser et représenter, par le biais des cas d'utilisation les requis fonctionnels du système. Ces derniers sont vus dans le système comme une couche des services, des interfaces externes ou des API (*Application Program Interface*). Les requis fonctionnels sont présentés en détails au prochain chapitre.

Quant aux requis non fonctionnels (*RNF*), nous en avons principalement cinq:

- l'expansion des nœuds de la topologie (*RNF. 1*) ;
- l'interopérabilité du système proposé avec les entités externes (*RNF. 2*) ;
- la réduction du temps global de transit entre un serveur de découverte des services et son MAP associé (*RNF. 3*) ;
- l'autonomie de prise de décision dans la sélection d'un serveur d'applications (*RNF. 4*) ;
- la consistance des données de QoS (*RNF. 5*).

Le requis de l'expansion des nœuds de la topologie est transformé dans sa réalisation en un requis fonctionnel de gestion de la topologie. Cette fonctionnalité consiste à ajouter, supprimer ou rechercher un nœud de la topologie. Sa description est détaillée au prochain chapitre.

L'interopérabilité permet au système proposé de pouvoir échanger les messages avec les entités externes interagissant avec lui. Elle contribue à la compréhension et à l'interprétation des messages reçus, indépendamment des plates-formes utilisées par les

deux entités en communication de bout-en-bout. La réalisation de ce requis consiste à choisir un protocole d'interopérabilité qui cadre avec le type de message transmis et la technologie des services Web.

La réduction du temps global de transit entre un serveur de découverte des services et son MAP est un critère utilisé pour déterminer l'emplacement idéal du LAN d'un serveur de découverte des services associé à un réseau mobile d'accès aux paquets. Le principe détaillé de ce requis est expliqué dans la section 3.2.1.

L'autonomie de prise de décision dans la sélection d'un serveur d'applications est transformée dans sa réalisation en un requis fonctionnel permettant, lors d'une recherche ou d'une coordination de la migration de service, que le serveur de découverte des services impliqué dans le choix du serveur d'applications géographiquement le plus proche (et offrant le niveau de QoS requis par le client si le service est avec QoS) puisse prendre la décision de sélectionner un serveur d'applications sans communiquer avec une autre entité externe. Pour réaliser ce requis, nous proposons une relation de visibilité expliquée à la section 3.3.4.

Le requis sur la consistance de données de QoS est transformé dans sa réalisation en un requis fonctionnel qui consiste à rafraîchir périodiquement les données de QoS nécessaires à la sélection d'un serveur d'applications dans le cas d'une recherche ou d'une migration de service avec QoS. Le principe de ce requis est expliqué à la section 3.3.5.

3.2 Principes de l'architecture proposée

Les principes que nous établissons ici permettent d'éclairer ou de comprendre les facteurs qui ont influencé le choix topologique effectué et les concepts de génie logiciel utilisés.

3.2.1 Facteurs influençant le choix topologique du système GLWSA

Distribuer un service revient à multiplier des points de sources fournissant le service. La distribution soulève différents types de questions : où, quand et comment

répartir le service. En effet, les questions « où » et « quand » reviennent à définir les critères qui font en sorte qu'une région plus qu'une autre soit éligible à héberger un serveur de découverte de services. Le « comment » revient à expliciter la logique utilisée pour arriver à cette fin. Nous considérons quatre facteurs principaux influençant le choix topologique du système GLWSA:

- la perception de la mobilité par un serveur rattaché au réseau IP ;
- la réduction du temps global de transit entre un serveur de découverte des services et son MAP associé ;
- l'autonomie de prise de décision ;
- la consistance des données de QoS.

Dans une communication IP reliant un client mobile à un serveur quelconque rattaché au réseau Internet, le réseau mobile de prochaine génération peut être vu comme un ensemble de réseaux d'accès aux paquets ayant chacun un MAP ou une passerelle vers le réseau Internet. Le réseau d'accès aux paquets permet d'interconnecter une ou plusieurs zones de localisation d'un réseau mobile et d'agréger ainsi le trafic de données vers le réseau Internet. Pour rediriger les données vers le réseau Internet ou vers les clients mobiles, on associe à chaque zone de localisation un routeur d'accès.

Dans ce contexte, le serveur en question perçoit et résume la mobilité du client à une macromobilité (un changement de MAP), car, pour maintenir la communication en cours, ledit serveur n'a pas besoin de connaître la mobilité du client dans les zones de localisation couvertes par le MAP de la position courante du client. Cela fait du MAP le point d'agrégation de la mobilité le plus élevé. Ce point d'agrégation est utilisé dans la mobilité IP pour reconnaître le réseau d'accès aux paquets de résidence (*home access network*) et les réseaux d'accès aux paquets visités (*visited access networks*) par un client mobile.

Ainsi, dans le cas où le client effectue une micromobilité (mobilité dans les zones de localisation couvertes par un réseau d'accès aux paquets donné), le trajet de tout paquet IP envoyé par le client mobile en direction dudit serveur passe par le MAP associé à ce réseau d'accès aux paquets. Du fait que, dans un réseau d'accès aux

paquets, tout paquet qui y est émis passe par le MAP, si l'on veut réduire le temps de transit global entre un client et le serveur de découverte de services de son réseau d'accès aux paquets, il faudrait choisir convenablement où disposer un serveur de découverte de services par rapport au MAP.

Le temps global de transit d'un paquet est la somme de deux expressions: le temps de propagation et le temps de transit de tous les routeurs intermédiaires (requis non fonctionnel *RNF*. 3). Le temps de propagation est lié à la distance séparant le MAP et le serveur de découverte des services, tandis que le temps de transit de tous les routeurs intermédiaires est la somme des temps d'attente de tous les routeurs intermédiaires traversés (en considérant le temps de traitement de chaque routeur intermédiaire comme négligeable). Par conséquent, pour réduire le temps global de transit des messages (temps de propagation + temps d'attente) du client mobile au serveur, il suffit de collocaliser le serveur de découverte des services au MAP. Nous entendons par *collocalisation* du serveur de découverte des services avec le MAP le fait que le LAN du serveur de découverte des services soit directement rattaché au MAP. En effet, la collocalisation du serveur de découverte des services et du MAP permet de réduire le nombre de routeurs intermédiaires traversés par un paquet et de rapprocher géographiquement la distance entre les deux entités; il en résulte ainsi une réduction du temps global de transit d'un paquet entre le MAP et le serveur de découverte des services.

Lors d'une coordination de migration de service ou d'une requête de recherche de service émise par un client auprès d'un serveur de découverte de services, la prise de décision de sélectionner un serveur d'applications sur un ensemble de serveurs d'applications éligibles doit être effectuée localement au serveur de découverte de services interrogé. Cette propriété d'autonomie du serveur de découverte de services permet d'éliminer un flux supplémentaire d'échange de messages entre serveurs au moment du traitement de la requête d'un client. En effet, les informations nécessaires à un serveur de découverte de services pour sa prise de décision de sélectionner un service *S* sont acquises en temps masqué et sont valides pour une période de temps déterminée

en ce qui concerne l'acquisition des paramètres de QoS (bande passante dans un domaine spécifique, facteur d'utilisation d'un serveur d'applications). Cette période de temps sert de signal pour déclencher le rafraîchissement des données de QoS. Ce rafraîchissement permet alors d'avoir ou de conserver des données consistantes de QoS.

La spécification des réseaux mobiles de prochaine génération fait ressortir principalement quatre catégories de QoS pour les applications de trafic: conversationnel (voix, vidéo, téléphonie, etc.), flots (multimédia, etc.), interactives (interactions avec le Web, client/serveur, accès aux bases de données, etc.) et d'arrière plan (courriel, téléchargement, etc.). Les applications géolocalisées (applications qui implémentent des services Web géolocalisés) font partie de la catégorie des applications interactives. Nous avons choisi les paramètres de QoS en conformité aux recommandations formulées par le groupe 3GPP [35].

3.2.2 Principes de génie logiciel appliqués

L'architecture proposé GLWSA est une architecture *orientée services*. Nous entendons par *architecture orientée services*, une architecture qui consiste à développer les applications comme des composants logiciels indépendants, ayant des interfaces externes de services avec lesquels les usagers interagissent. Elle permet aussi de sélectionner des services désirés dans une collection donnée et de les assembler afin de réaliser une caractéristique fonctionnelle spécifique [20]. Les services utilisés dans le système GLWSA utilise la technologie Web (on parle alors de service Web) et la technologie des messages asynchrones.

Un *service Web* est une interface externe (ou publiée) qui est mise à la disposition des clients dans le but d'interagir avec une application logicielle. Il utilise la technologie Web (HTTP) pour transporter les messages échangés. Un service Web a un formalisme qui est décrit dans un document WSDL (la description de ce protocole est effectué à la section 2.1.1). Dans le cas d'un appel service Web, les messages échangés entre un émetteur et un récepteur sont mis sous le format d'interopérabilité SOAP (expliqué au dans la section 2.1.3, *requis non fonctionnel RNF.2*) issu de XML. En effet,

le formalisme d'interopérabilité SOAP est utilisé comme standard d'interopérabilité pour effectuer des appels distants de services Web et échangés des structures de données mises sous format XML. L'implémentation d'un service Web est effectuée par une classe qui lui est associée. Le service Web dans le système GLWSA est distribué à différents nœuds représentant les points d'accès aux services.

La technologie des messages asynchrones permet à un client d'envoyer un message destiné à un receveur (on parle d'une communication point à point) ou d'envoyer un message destiné à plusieurs receveurs y ayant souscrit (on parle d'un modèle publication/souscription).

L'architecture orientée services utilise à la base le modèle objets pour représenter les fonctionnalités et les structures de données du système. La modélisation objet consiste à créer une représentation abstraite des entités ayant une existence matérielle (arbre, personne, cellulaire, etc.) ou bien virtuelle (sécurité sociale, compte bancaire, etc.) sous forme d'objets. Un objet est caractérisé par plusieurs notions:

- *Les attributs* (on parle parfois de propriétés): Il s'agit des données caractérisant l'objet. Ce sont des variables stockant des informations d'état de l'objet.
- *Les méthodes* (appelées parfois fonctions membres): Les méthodes d'un objet caractérisent son comportement, c'est-à-dire l'ensemble des actions (appelées opérations) que l'objet est à même de réaliser. Ces opérations permettent de faire réagir l'objet aux sollicitations extérieures (ou d'agir sur les autres objets). De plus, les opérations sont étroitement liées aux attributs, car leurs actions peuvent dépendre des valeurs des attributs, ou bien les modifier.
- *L'identité*: L'objet possède une identité, qui permet de le distinguer des autres objets, indépendamment de son état. On construit généralement cette identité grâce à un identifiant découlant naturellement du problème (par exemple un produit pourra être repéré par un code, une voiture par un numéro de série, etc.).

On appelle classe la structure d'un objet, c'est-à-dire la déclaration de l'ensemble des entités qui composeront un objet. Un objet est donc "issu" d'une classe. En réalité, on dit qu'un objet est une **instanciation** d'une classe, c'est la raison pour laquelle on

pourra parler indifféremment d'objet ou d'instance (éventuellement d'occurrence). Une classe est composée des *attributs* (il s'agit des données, dont les valeurs représentent l'état de l'objet) et des *méthodes* (il s'agit des opérations applicables aux objets).

L'un des principaux concepts du paradigme objet est *l'encapsulation*. L'encapsulation est un mécanisme consistant à rassembler les données et les méthodes au sein d'une structure en cachant l'implémentation de l'objet, c'est-à-dire en empêchant l'accès aux données par un moyen autre que celui proposé par les services publiés.

En effet, la programmation orientée objets permet de cacher l'implémentation des méthodes et la structure des données des attributs d'un objet. Pour qu'un client puisse utiliser les objets d'un système, la programmation orientée objets préconise qu'il interagisse avec des méthodes publiques encapsulées dans une interface externe. On distingue principalement trois niveaux d'encapsulation ou de visibilité:

- *publique*: Il permet un accès sans condition à une méthode ou un attribut d'une classe. Il s'agit du plus bas niveau de protection des données ;
- *protégée*: Seules les classes qui héritent de la classe de base offrant cet accès à une ressource (un attribut ou une méthode) et d'autres méthodes de cette classe de base peuvent interagir avec la dite ressource ;
- *privée*: L'accès à un attribut ou une méthode est limité aux méthodes de la classe elle-même. Il s'agit du niveau de protection des données le plus élevé.

La notion d'*héritage* permet à une classe dite *dérivée* de réutiliser les attributs et les méthodes implémentés par une classe de base. La classe dérivée se différencie de la classe de base soit par le biais des attributs et des méthodes qui lui sont propres ou par une re-écriture des méthodes implémentées dans la classe de base.

Si une classe de base est héritée par plusieurs classes dérivées, l'instanciation de classe de base lors d'une exécution peut prendre la forme de la structure de données de l'une des classes dérivées. On parle alors de *polymorphisme* d'un objet.

3.3 Architecture du système GLWSA

Pour réaliser cette architecture, notre approche méthodologique consiste à délimiter le système étudié afin de déterminer les entités externes agissant sur lui-même, de proposer une topologie de répartition des nœuds, et d'énoncer les concepts de la politique de recherche, de publication et de migration de services [11], [12], [13], [14]. Cette section présente la description des principales composantes de l'architecture, l'énoncé du principe de la consistance de données de QoS, l'impact de la mobilité sur le système, l'extension du registre UDDI et du protocole MLP, et la typologie des applications géolocalisées.

3.3.1 Description des principales composantes

Comme l'illustre la Figure 3.1, le système GLWSA est composé principalement de trois entités :

- le serveur de découverte des services Web géolocalisés GLWSM (*Geo-Located Web Services Manager*) ;
- la base de données UDDIM (*UDDI for Mobiles*) ;
- et la base de données des informations des usagers *ClientInfosDB*.

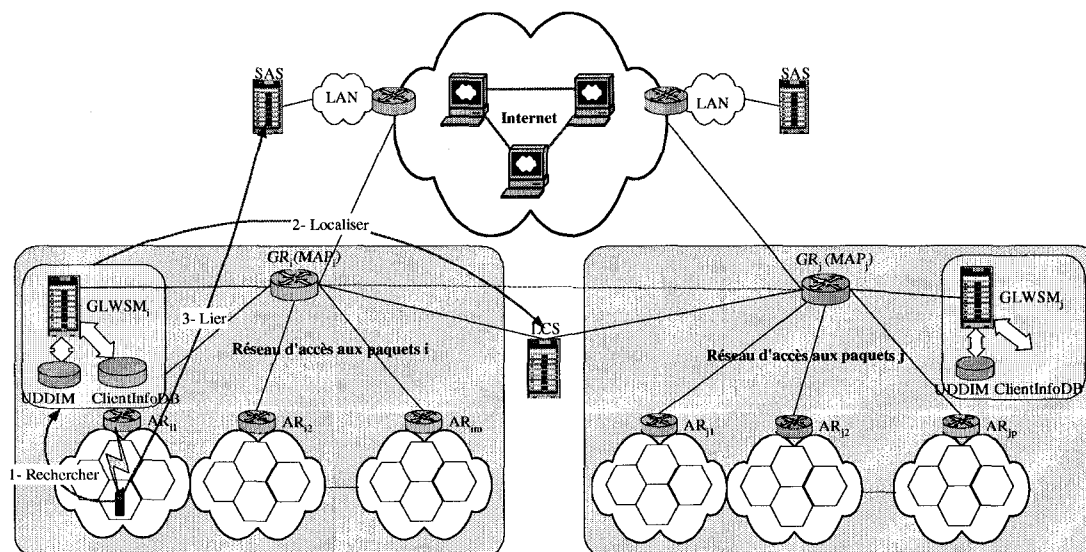


Figure 3.1 Architecture du système GLWSA

Le GLWSM est l'organe qui contrôle la logique du traitement des informations du système. La base de données UDDIM est une extension du registre UDDI pour les mobiles dans le but de tenir compte de la topologie de répartition des nœuds GLWSM. Cette extension ajoute au UDDI les informations sur la position des nœuds GLWSM, l'agrément d'un service et ses paramètres de QoS. La base de données *ClientInfosDB* enregistre les informations personnelles d'un client, les données sur l'équipement utilisé par un client mobile, et les paramètres sur l'abonnement d'un client mobile.

Le serveur GLWSM interagit avec des entités externes telles que les clients mobiles, les serveurs des fournisseurs d'application SAS et le serveur LCS. Le langage SOAP sert à formater les messages échangés entre un client mobile et un GLWSM, ainsi qu'entre un GLWSM et un SAS. Les messages asynchrones (en utilisant par exemple JMS « *Java Message Service* ») sont échangés entre deux GLWSM afin d'effectuer le suivi de la position courante d'un client mobile. Le dialogue entre un GLWSM et un LCS se fait par le biais du protocole MLPe (*MLP extension*). Le protocole MLPe est une extension du protocole MLP afin de pouvoir effectuer une localisation des mobiles sur une base thématique. L'interaction entre un serveur GLWSM et ses bases de données associées UDDIM et ClientInfosDB se fait par le biais d'un connecteur des données ODBC (*Open DataBase Connectivity*) ou JDBC (*Java DataBase Connectivity*). Dans le système GLWSA, le transport des messages échangés entre entités de bout-en-bout se fait en mettant à contribution les ressources du réseau mobile et du réseau Internet.

➤ **Topologie du système GLWSA**

Tel qu'illustré à la Figure 3.2, le modèle de répartition des nœuds GLWSM proposé est un modèle hiérarchique à deux niveaux [11], [12], [13], [14]. Ce modèle a été choisi pour tenir compte de la contrainte d'extensibilité (gestion des nœuds), de l'autonomie de la prise de décision dans la sélection des SAS et de la consistance des données de QoS en limitant le flux d'échange de messages entre nœuds GLWSM. Le niveau le plus bas est le niveau des feuilles de l'arbre et le niveau supérieur représente la

racine de l'arbre. La racine de l'arbre a un nœud racine associé. Le niveau des feuilles comprend plusieurs nœuds.

Un nœud feuille contient et expose un ensemble de services publiés par des fournisseurs autorisés. La publication d'un service est coordonnée par le nœud de racine qui a une base de connaissances de tous les nœuds GLWSM déployés dans la topologie de GLWSA. Chaque réseau d'accès aux paquets est associé à un nœud feuille GLWSM et couvre un nombre déterminé de zones de localisation. Toute paire de nœuds GLWSM distincts couvre des zones de localisation distinctes. Un SAS peut être associée à plusieurs GLWSMs. Un service Web d'un fournisseur peut être réparti sur plusieurs SAS.

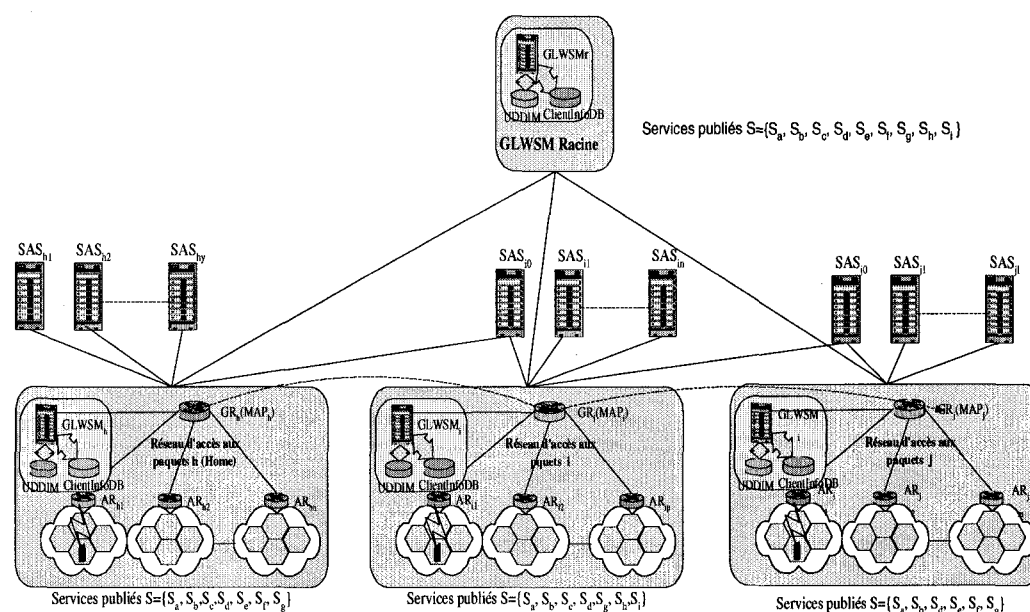


Figure 3.2 Topologie du système GLWSA

La caractéristique principale des nœuds feuille est qu'ils dépendent des zones de localisation du réseau mobile pour assurer le suivi de la mobilité du client, pour maintenir l'exécution de service dans un SAS, et pour contrôler la migration de service lorsqu'un client mobile entre dans une zone de localisation gérée par un autre nœud de GLWSM où le service dans l'exécution existe. Autrement, l'exécution du service cessera

d'être fournie par le SAS en cours d'exécution. Tous les nœuds feuille offrent des fonctionnalités pour l'authentification et l'autorisation, l'abonnement, la localisation des mobiles, la migration, la recherche et la publication des services.

➤ ***Relations et propriétés***

Les relations et propriétés du système sont décrites ci dessous.

Relation de visibilité : Tout nœud GLWSM qui a un service S_i connaît tous les autres nœuds GLWSMs qui offrent le même service.

La relation de visibilité permet à tout nœud d'être autonome dans la prise de décision, de sélectionner un SAS le plus proche de la position courante du client mobile (requis non fonctionnel *RNF*. 4).

Nœud de résidence (Home Node) : C'est le nœud feuille GLWSM_h où un client mobile est enregistré, s'authentifie ou s'identifie, et formule sa requête de recherche d'un service donné.

Nœud visité (Visited Node) : C'est un nœud feuille GLWSM distinct du nœud de résidence d'un client mobile spécifique.

Nœud le plus proche : C'est le nœud feuille GLWSM qui couvre la position géographique courante d'un client mobile spécifique.

SAS le plus proche : C'est un serveur du fournisseur d'applications associé au nœud GLWSM le plus proche et qui rencontre les requis de QoS d'un client mobile.

3.3.2 Recherche d'un service Web géolocalisé

La procédure de recherche d'un service Web géolocalisé implique cinq acteurs (Figure 3.3) : le client mobile, le serveur GLWSM de résidence, le serveur GLWSM le plus proche, le serveur d'applications SAS le plus proche et le serveur de localisation LCS.

La description fonctionnelle de la procédure est la suivante (Figure 3.4):

1. le client mobile envoie une requête de recherche de service Web géolocalisé à son serveur GLWSM_r de résidence. La requête peut être avec ou sans QoS.

Les paramètres transmis dans cette communication sont l'identifiant du service et l'identifiant du mobile auxquels on peut rajouter les paramètres de QoS requis (facteur d'utilisation du serveur d'applications, la bande passante et le coût du service) ;

2. le serveur GLWSM_r envoie au LCS une requête de localisation en spécifiant l'identifiant du mobile ;

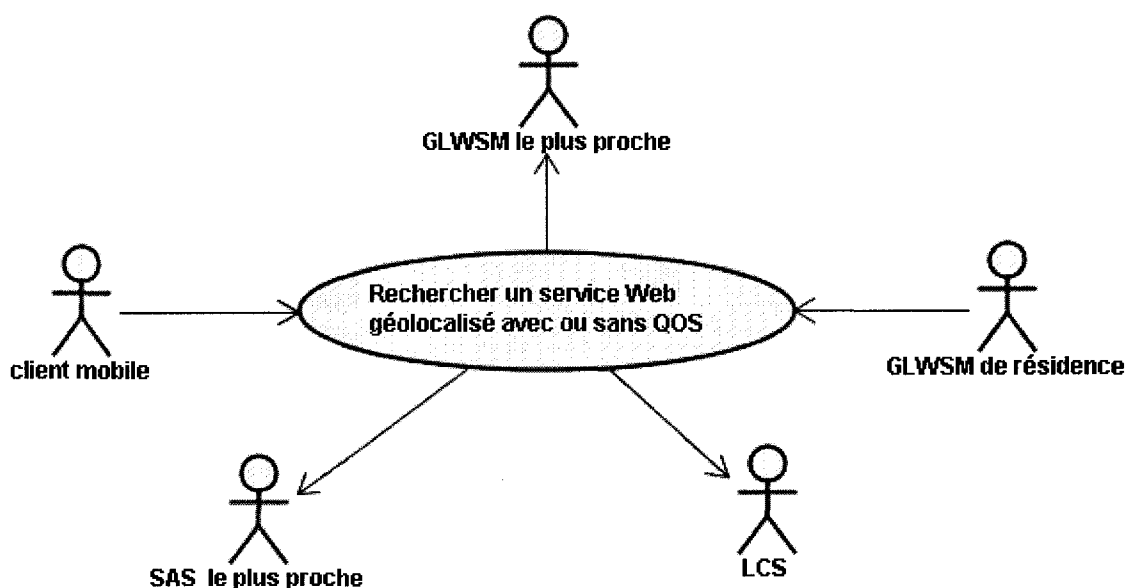


Figure 3.3 Cas d'utilisation d'une recherche de service Web géolocalisé

3. le serveur LCS retourne la position du client mobile au GLWSM_r, celui-ci sélectionne alors dans la base de données UDDIM le nœud GLWSM qui couvre la position courante du client mobile. Le nœud GLWSM sélectionné est considéré comme le serveur GLWSM le plus proche ;
4. le serveur GLWSM vérifie ensuite si le service demandé est octroyé par le nœud GLWSM sélectionné à l'étape 3 (le requis *RNF. 4* est utilisé ici), puis sélectionne l'URL du serveur d'applications. Si la QoS est requise, le critère statique sur le coût ainsi que le critère dynamique sur la bande passante et le facteur d'utilisation est validé avant de sélectionner le serveur d'applications

- SAS. Le serveur SAS trouvé à cette étape est considéré comme le serveur SAS le plus proche ;
5. le serveur GLWSM_r envoie un message asynchrone au serveur GLWSM trouvé à l'étape 3 lui demandant de traquer ou de faire un suivi de la position du mobile se trouvant dans sa région de couverture ;
 6. le serveur GLWSM_r retourne au client mobile l'URL du serveur d'applications ainsi que les paramètres nécessaires à la négociation de la QoS (bande passante minimum et maximum, le délai de propagation minimum et maximum, et la taille maximale d'une unité de service) ;
 7. le client mobile utilise l'URL pour se connecter au serveur SAS le plus proche.

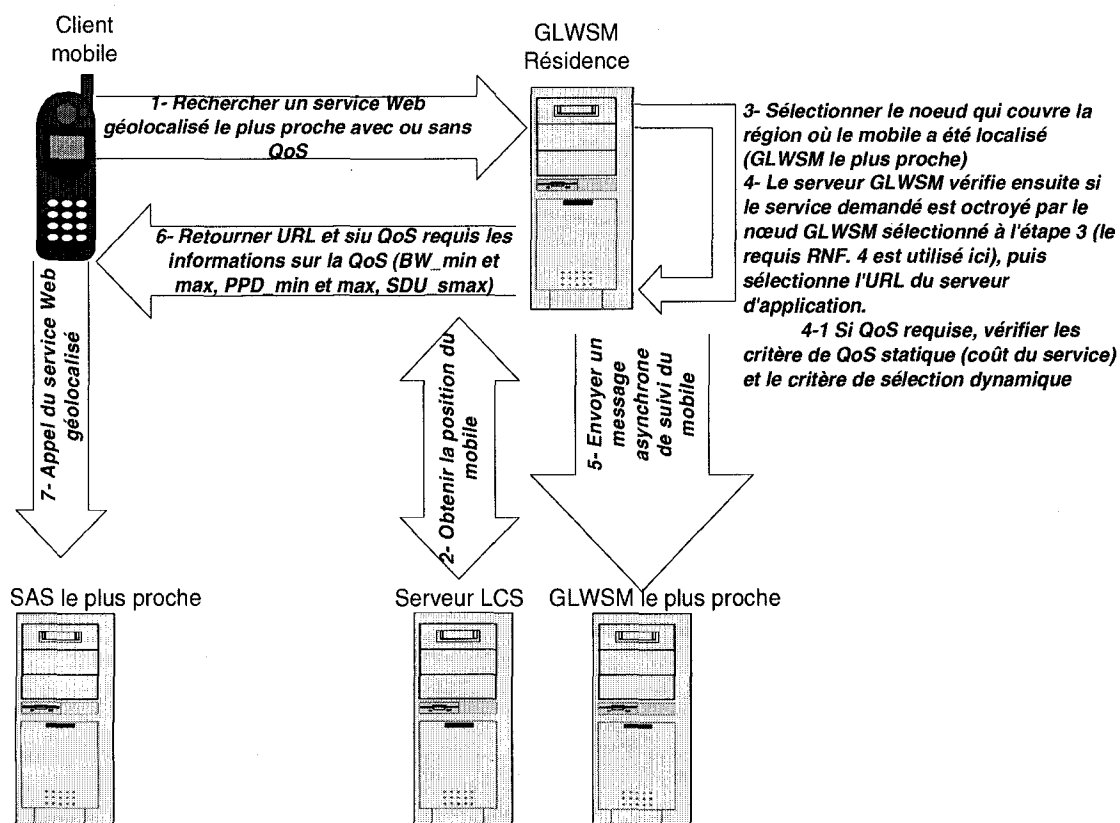


Figure 3.4 Diagramme de déploiement d'une recherche de service

3.3.3 Publication d'un service Web géolocalisé

La procédure de publication d'un service Web géolocalisé implique trois acteurs (Figure 3.5): le fournisseur autorisé, le serveur GLWSM racine, les serveurs GLWSM offrant le service à publier.

Conditions préalables : On suppose que le fournisseur autorisé a obtenu au préalable un agrément de service aux nœuds.

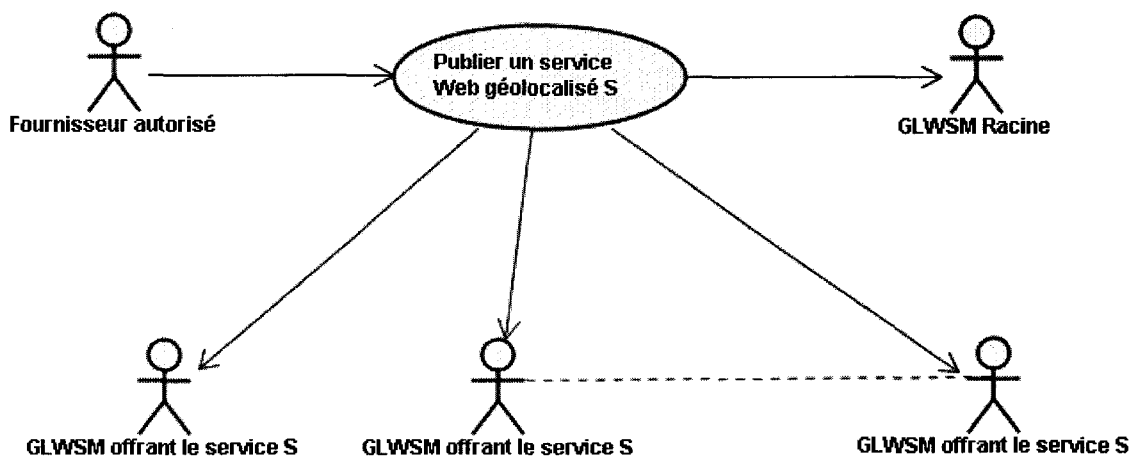


Figure 3.5 Cas d'utilisation d'une publication de service Web géolocalisé

Procédure :

La description de la procédure de déploiement est la suivante (Figure 3.6) :

1. le fournisseur autorisé envoie une requête de publication de service Web géolocalisé au serveur GLWSM_h racine. La requête contient les informations sur le service et la liste des nœuds feuille GLWSM sélectionnés qui offriront un accès au service ;
2. le serveur GLWSM_h enregistre le service à son registre UDDIM associé ;
3. le serveur GLWSM_h envoie un message asynchrone de réplication du service à tous les nœuds feuille GLWSM sélectionnés. Toutes les informations transmises à l'étape (1) sont redirigées à chacun des nœuds. Ainsi, chaque nœud feuille connaît les autres nœuds feuille offrant le service publié (requis

non fonctionnel *RNF4*). Après trois tentatives d'envoi, un message d'erreur contenant la date, les informations sur le service, les identifiants des nœuds feuille où le service n'a pas été répliqué est ajouté au fichier d'erreur de publication pour traitement futur ;

4. les nœuds feuille GLWSM écoutent la queue de publication de service ;
5. chaque nœud feuille GLWSM enregistre le service ainsi que les informations sur les autres nœuds feuille offrant le service ;

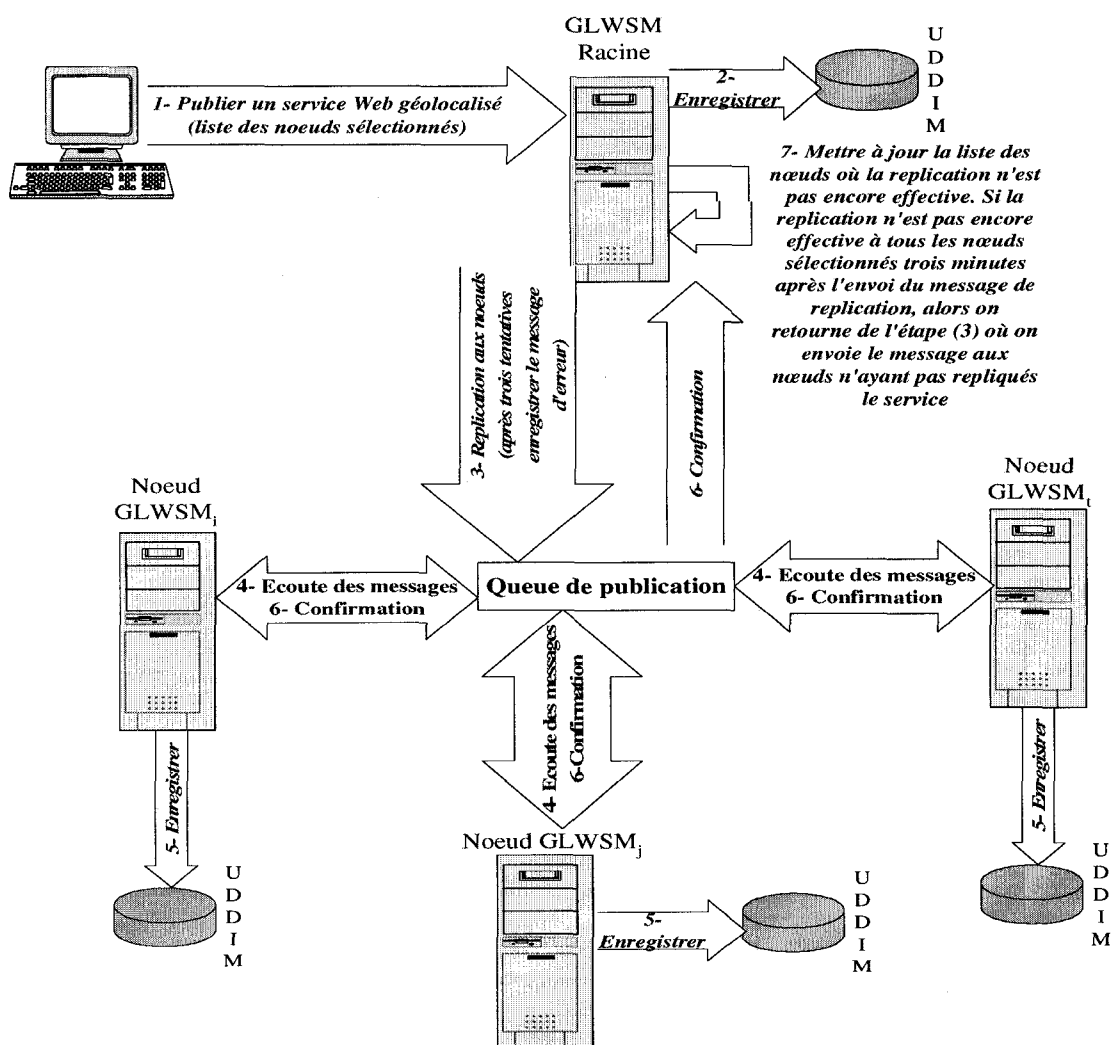


Figure 3.6 Diagramme de déploiement d'une publication de service Web géolocalisé

6. chaque nœud feuille GLWSM confirme l'enregistrement du service au nœud racine $GLWSM_h$ via une queue de publication ;
7. cette confirmation est traitée par le GLWSM afin de mettre à jour la liste des nœuds où la réplication n'est pas encore effective. Si la réplication n'est pas encore effective à tous les nœuds sélectionnés trois minutes après l'envoi du message à répliquer, alors on retourne de l'étape (3) où on envoie à nouveau le message aux nœuds n'ayant pas enregistré le service.

3.3.4 Coordination de la migration d'un service Web géolocalisé

La procédure de coordination de la migration d'un service Web géolocalisé implique cinq acteurs (Figure 3.7) : le GLWSM courant, le GLWSM suivant, le serveur LCS, le serveur SAS courant et le serveur SAS suivant.

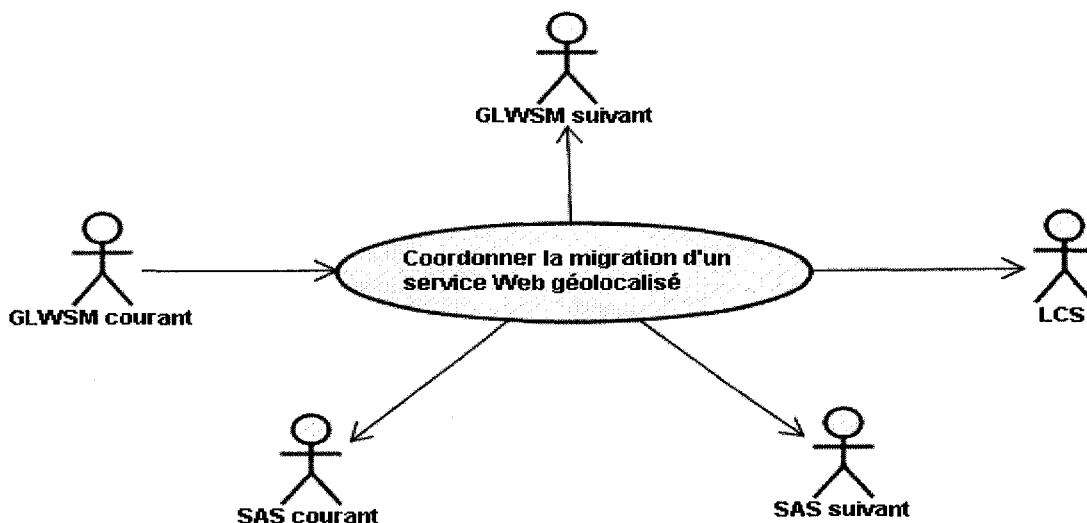


Figure 3.7 Cas d'utilisation de la coordination d'une migration de service Web géolocalisé

Procédure :

La description de la procédure de déploiement est la suivante (Figure 3.8) :

1. en communiquant avec le serveur LCS, le serveur GLWSM courant traque la position d'un mobile en cours d'exécution et se trouvant dans sa zone de couverture géographique ;
2. si le mobile sort de la zone de couverture du GLWSM courant, ce dernier sélectionne dans son registre UDDIM le nœud GLWSM où le mobile a été localisé (ce nœud est considéré comme le nœud GLWSM suivant le plus proche) ;

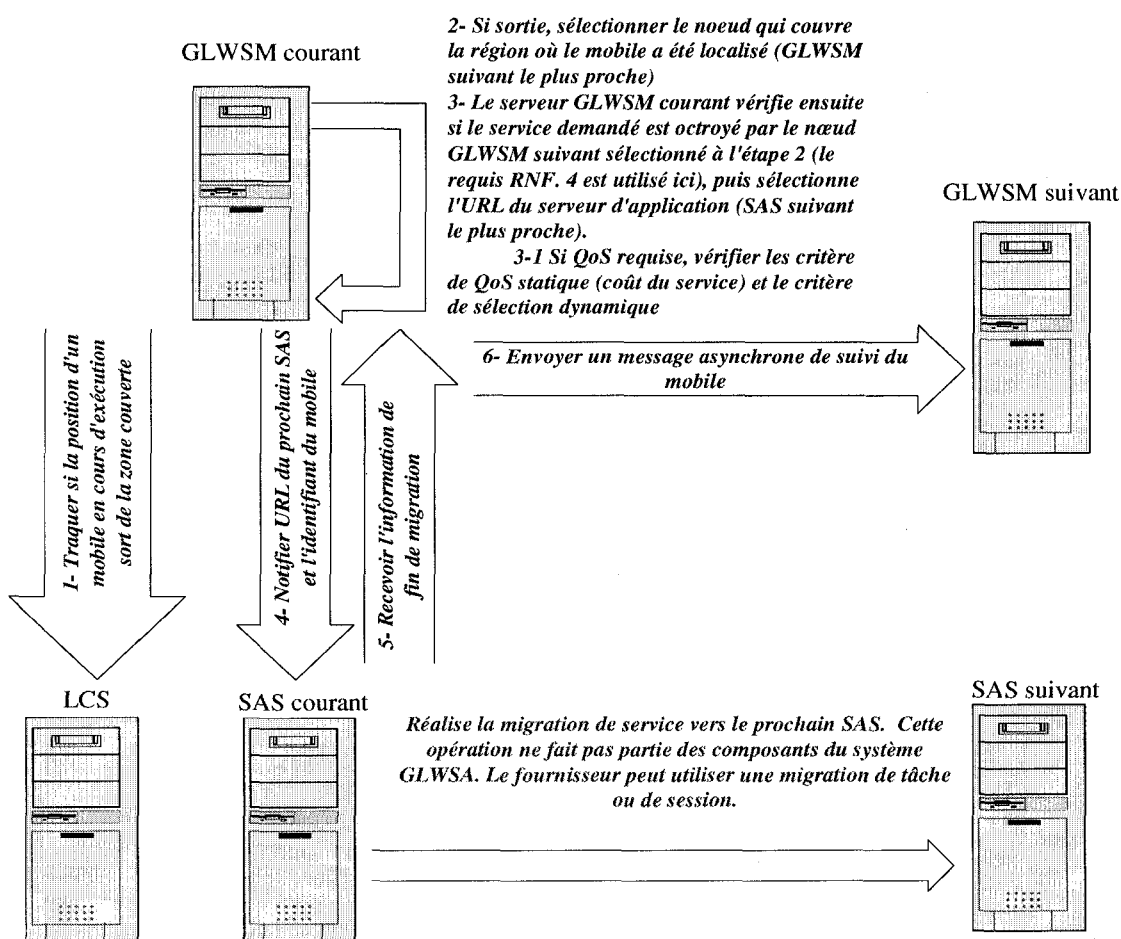


Figure 3.8 Diagramme de déploiement de la coordination d'une migration de service Web géolocalisé

3. le serveur GLWSM courant vérifie si le service en cours d'exécution est octroyé au nœud GLWSM suivant (requis non fonctionnel *RNF*. 4). Si c'est le cas, il sélectionne, par le biais d'une cache d'informations de QoS, l'URL de serveur SAS suivant qui respecte les critères de QoS du client sur le coût, la bande passante et le facteur d'utilisation ;
4. le serveur GLWSM courant notifie au SAS courant l'URL du prochain SAS ainsi que l'identifiant du mobile concerné. À la réception du message, le SAS courant effectue la migration de service au SAS suivant (cette opération de migration de service de SAS à SAS n'est pas de la portée du système GLWSA, le fournisseur peut utiliser une technique de migration de session ou de tâche existante) ;
5. le serveur SAS envoie un accusé de fin de migration au serveur GLWSM courant ;
6. le serveur GLWSM courant envoie un message asynchrone de suivi de mobile au nœud GLWSM suivant.

3.3.5 Collection et consistance des données de QoS

Le système GLWSA possède deux catégories de données de QoS : les données statiques et les données dynamiques [12], [14]. Les données statiques de QoS sont transmises et enregistrées dans le système lors d'une publication de service. Il s'agit du coût d'un service à un nœud, la bande passante minimale et maximale d'un service à un nœud, le délai de propagation minimal et maximal d'un service à un nœud, et la taille maximale d'une unité de service.

➤ *Collection des données dynamiques de QoS*

Les données dynamiques sont collectées périodiquement pour un service donné à un nœud GLWSM spécifique. La logique utilisée est la suivante: pour un client mobile se trouvant dans une zone de localisation donnée (Figure 3.9), le choix du meilleur serveur SAS si l'on utilise comme critère la bande passante disponible n'est pas

déterminant dans le domaine délimité entre le client mobile et le MAP, mais plutôt entre le MAP et les différents serveurs SAS [12], [14]. En effet, dans le sous domaine incluant le client mobile et le MAP, s'il n'y a pas suffisamment de bande passante disponible et requise par un client mobile, aucun des serveurs SAS liés au nœud cible GLWSM n'offrira une bande passante suffisante pour répondre au niveau requis par le client mobile. Si par contre la bande passante requise peut être octroyée dans le sous domaine incluant le client mobile et le MAP, la différence dans la bande passante disponible se fera entre le MAP et les différents SAS offrant le service.

Le principe détaillé de la collection de la QoS dans un domaine particulier $GLWSM_h$ consiste à formuler une requête de soumission de QoS d'un service donné au serveur d'applications SAS l'implémentant (Figure 3.10). À la réception de cette requête, le SAS interroge le réseau sur la bande passante disponible et détermine son facteur d'utilisation. Il retourne par la suite les valeurs obtenues au $GLWSM_h$ qui les retransmet à tous les serveurs GLWSM stockant le même service en écrivant dans une queue qu'ils écoutent.

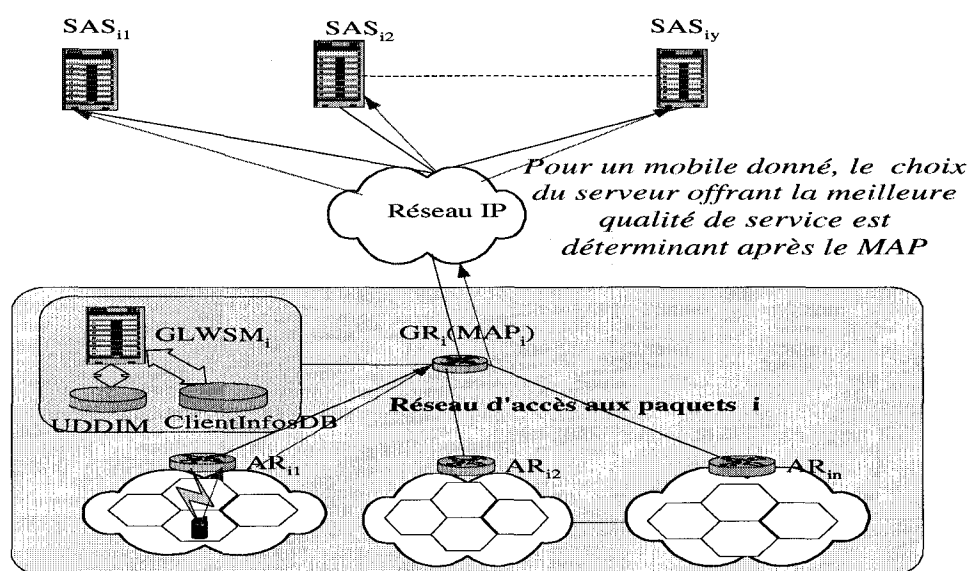


Figure 3.9 Choix d'un serveur SAS en utilisant le critère de la bande passante

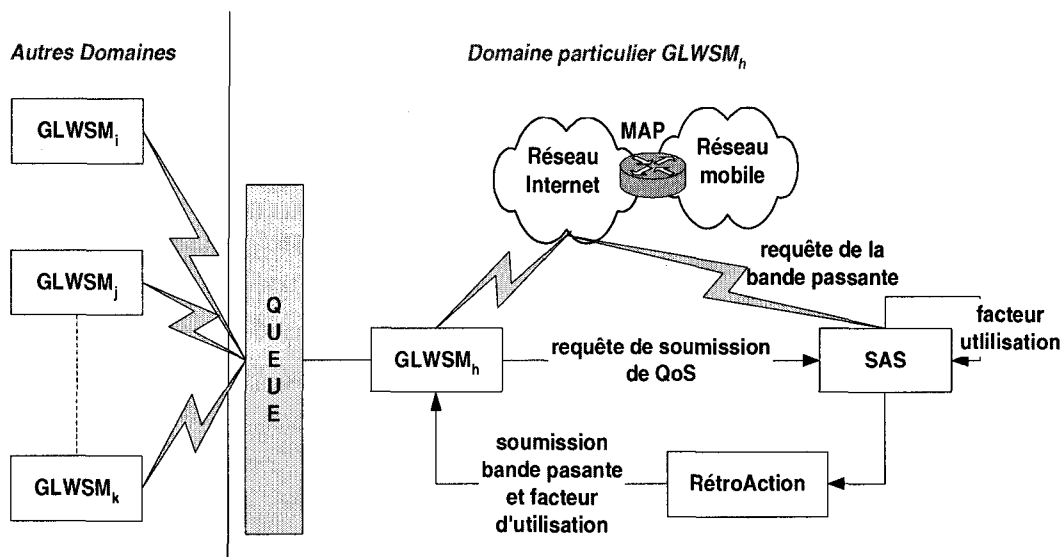


Figure 3.10 Collection et consistance de la QoS dans un domaine $GLWSM_h$

➤ **Rafraîchissement des données de QoS et appréciation de la bande passante**

Les données de QoS collectées sont rafraîchies toutes les 30 secondes du fait que les applications qui utilisent la réservation des ressources (généralement les applications multimédia) sont généralement impliquées dans de longs échanges de données. Cette période peut être modifiée par un administrateur autorisé.

Pour apprécier la bande passante collectée (Figure 3.11), nous divisons l'ensemble des valeurs admissibles de la bande passante (de BW_{min} à BW_{max}) en dix segments équidistants BWS_i (i variant de 1 à 10). Chaque segment BWS_i a un poids associé de i . Le poids de 10 est affecté au segment ayant pour borne supérieure BW_{max} .

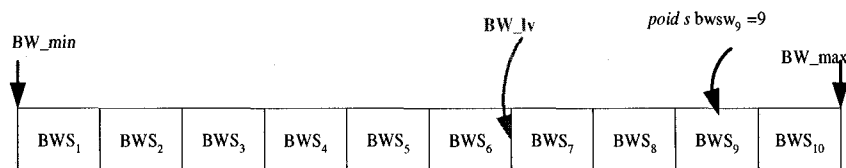


Figure 3.11 Appréciation de la bande passante

➤ **Critère de sélection dynamique**

Le critère de sélection dynamique DSC (*Dynamic Selection Criteria*) d'un serveur d'applications est donné par les équations suivantes [12], [14]:

$$DSC = (0.1 * b_{wsw_i}) * \alpha + (1 - u_{f_i}) * \beta \quad (3.1)$$

$$\alpha + \beta = 1 \quad (3.2)$$

Où b_{wsw_i} représente le poids de la bande passante et u_{f_i} le facteur d'utilisation du serveur SAS_i . Les coefficients α ($\alpha \geq 0$) et β ($\beta \geq 0$) sont des paramètres réels et positifs qui atténuent les mesures effectuées.

➤ **Consistance des données dynamiques de QoS**

La consistance des données de soumission de la QoS (requis non fonctionnel *RNF*. 5) matérialise la capacité d'un système réparti de découverte de services de transmettre, avant un délai déterminé DD, la soumission des données de QoS obtenue par un serveur de découverte de services (pour un service spécifique S et dans un domaine particulier) à tous les serveurs de découverte de services qui publient le service S [12], [14]. Le choix topologique de répartition des nœuds (nombre de niveaux, échange de messages entre serveurs de découverte) a une influence sur les performances ou l'appréciation de la consistance.

Le message de soumission de QoS est composé de deux paramètres de type réel, à savoir : la bande passante disponible dans un domaine particulier et le facteur d'utilisation du serveur d'applications dans ce domaine. Le message de soumission de QoS a donc une taille de 8 octets (équivalent à deux nombres réels). Le délai de communication subi dans la transmission de la soumission d'un serveur à un autre est induit davantage par le temps d'attente car le temps de propagation et le temps de transmission sont considérés comme négligeables.

Nous démontrons que la topologie proposée du système GLWSA a une meilleure performance que la topologie hiérarchique à trois niveaux minimum utilisée dans le système de découverte de services QoS-Aware [30]. Ce dernier a une approche

semblable à la nôtre en retournant par une boucle de rétroaction la QoS observée par les clients mobiles à leur serveur de résidence de découverte de services.

Dans le système de découverte QoS-Aware, deux concepts sont employés pour propager la soumission de QoS à tous les serveurs de découverte des services (*SDS*) de la feuille de l'arbre topologique, concernés par la mise à jour de QoS. D'abord, la propagation est faite verticalement. Un serveur de résidence de découverte de services envoie un message de mise à jour de la soumission de QoS à son nœud parent. Si le nœud parent courant fournit un accès au service, une mise à jour des données de QoS se produira. Puis, le nœud parent courant réoriente le message à ses nœuds fils qui stockent le service et à son nœud parent aussi. Le message est propagé progressivement à tous les serveurs de la topologie.

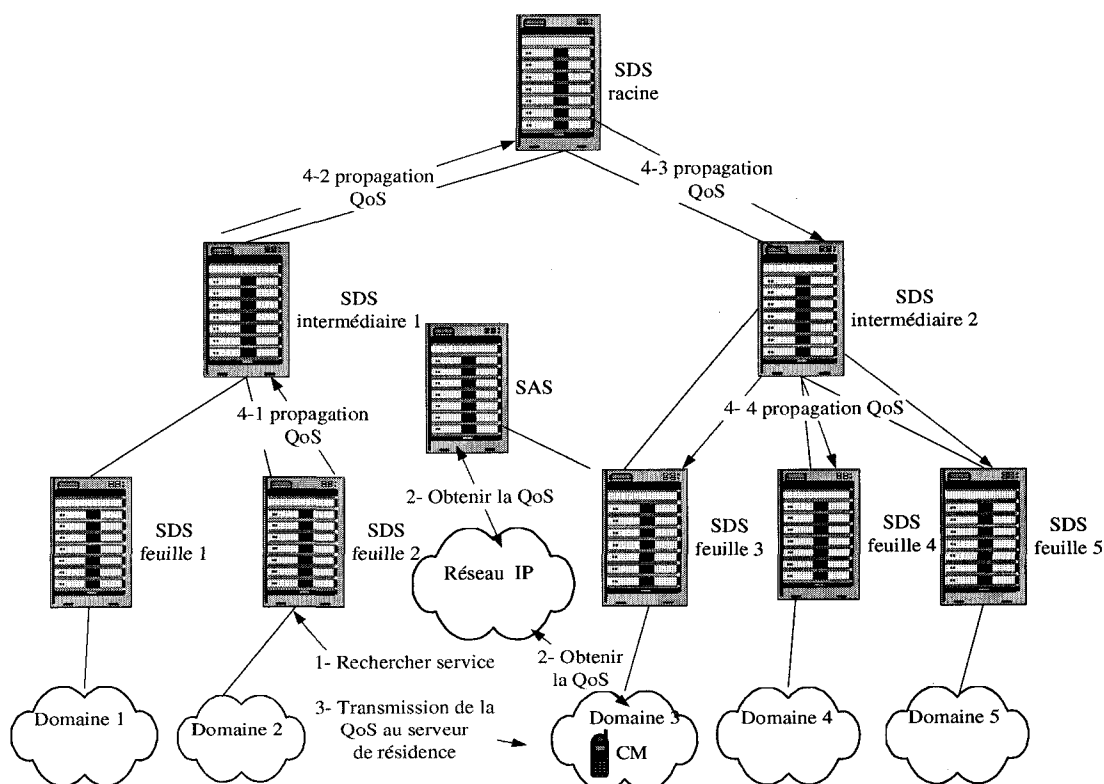


Figure 3.12 Modèle de propagation de QoS

Dans le deuxième concept, la propagation de la soumission de QoS se fait horizontalement. Le serveur de résidence de découverte des services initie la propagation du message de mise à jour de la QoS en la transmettant à son nœud successeur. Si le nœud successeur stocke le service dont la QoS qui a été propagée, une mise à jour de QoS se produira. Le nœud successeur change de rôle pour devenir le nœud courant. Puis, le nœud courant envoie à son successeur (le prochain nœud) un message de mise à jour. La propagation du message se fera progressivement vers tous les serveurs de découverte de services de la feuille de l'arbre concernés par les données de QoS. Une représentation du modèle de propagation des données de QoS du système QoS-Aware est donnée à la Figure 3.12.

Avec ces deux concepts, si un délai d'attente DQ survient pendant une mise à jour de la QoS entre deux serveurs de découverte de services (SDS_i vers SDS_j), ce délai sera induit à tous les autres serveurs de découverte de services qui dépendent de SDS_j . Ainsi, si le délai DQ est égal ou plus grand que le délai fixé DD, alors, le serveur SDS_j et tous serveurs qui dépendent de l'exécution de la mise à jour du message de QoS dans SDS_j auront des données de QoS inconsistantes.

Dans notre approche, un serveur de découverte des services $GLWSM_h$ déclenche la mise à jour de la soumission de QoS en écrivant dans une queue écoutée par tous les serveurs de découverte de services concernés par la mise à jour de la soumission de QoS. Par conséquent, si un délai d'attente DQ survient pendant une mise à jour de la QoS entre $GLWSM_h$ et un serveur $GLWSM_j$, ce délai ne sera pas propagé à tout autre serveur concerné par cette mise à jour. Dès lors, si le délai DQ est égal ou plus grand que le délai fixé DD, les données inconsistantes seront limitées au seul serveur $GLWSM_j$.

3.3.6 Impact de la mobilité

Deux scénarios de mobilité sont possibles quand un client mobile interagit avec un nœud feuille GLWSM : la macromobilité et la micromobilité (Figure 3.13).

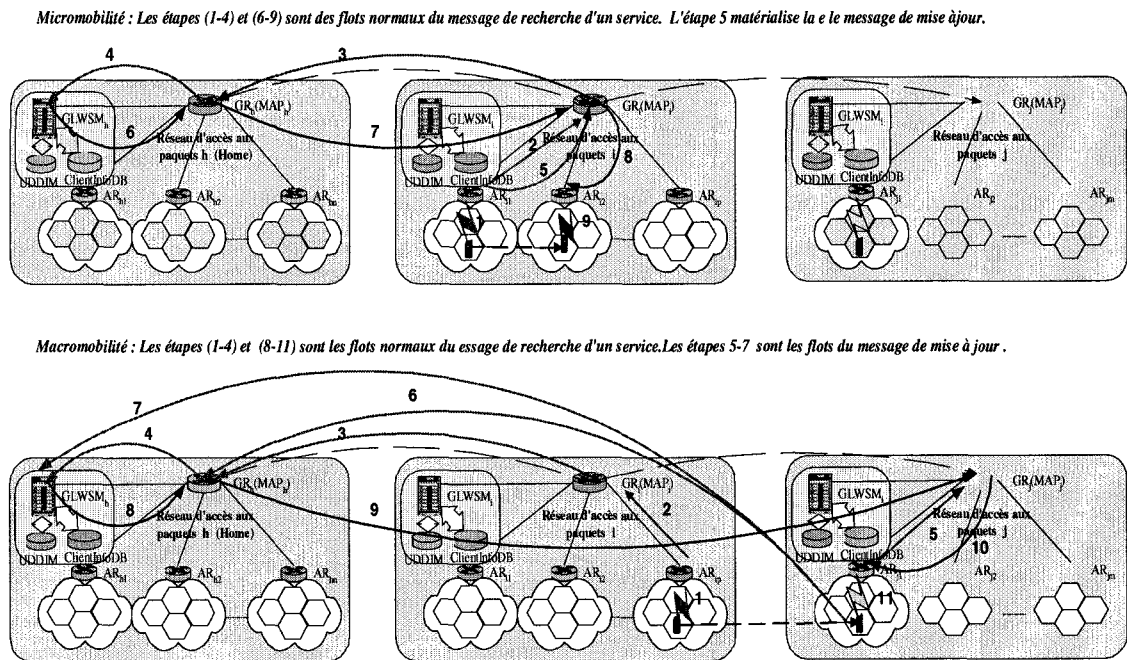


Figure 3.13 Impact de la mobilité dans GLWSA avec HMIPv6

Supposons que HMIPv6 soit utilisé comme protocole de micromobilité, et qu'un client mobile envoie un message de recherche de service au nœud $GLWSM_h$ par le biais du routeur d'accès AR_{i1} , puis change de point d'attache de AR_{i1} à AR_{i2} avant la réception de la réponse. Durant cette relève locale, le client mobile reçoit deux nouvelles adresses temporaires : $LCoA_{i2}$ (*Local Care-of-Address*) et $RCoA_i$ (*Regional Care-of-Address*). Ensuite, il envoie un message de mise à jour des adresses reçues $RCoA_i$ et $LCoA_{i2}$ au routeur ou passerelle GR_i (*Gateway Router*) qui agit comme un MAP. La passerelle GR_i extrait les adresses $LCoA_{i2}$ et $RCoA_i$ du message reçu, puis met à jour sa table de routage. Suite à cette mise à jour, tous les paquets envoyés par le nœud $GLWSM_h$ à destination du client mobile seront redirigés ou routés en utilisant les adresses $RCoA_i$ et $LCoA_{i2}$, ce qui permet au client mobile de recevoir la réponse de sa requête de recherche de service.

Dans le cas où le client mobile n'a pas reçu de réponse à son message de recherche de service envoyé au nœud $GLWSM_h$ et qu'une relève survienne lors d'une

macromobilité, c'est à dire que le client mobile change de réseau d'accès aux paquets de GR_i à GR_j , alors le client mobile reçoit du réseau IP une nouvelle adresse locale de transit $LCoA_{j1}$ et une nouvelle adresse régionale de transit $RCoA_j$. Le client mobile envoie trois mises à jour des adresses de transit reçues respectivement à la passerelle GR_j (contenant $LCoA_{j1}$ et $RCoA_j$), à l'agent de résidence « *Home Agent* » (contenant l'adresse $RCoA_j$) et au nœud de correspondance $GLWSM_h$ (contenant l'adresse $RCoA_j$). Ainsi, tous les paquets adressés au client mobile seront routés à sa nouvelle adresse régionale $RCoA_j$. La passerelle GR_j qui les reçoit, le redirige à l'adresse locale de transit $LCoA_{j1}$.

3.3.7 Extension des protocoles UDDI et MLP

Du fait que le protocole UDDI est utilisé comme standard dans la découverte des services Web, nous l'avons choisi comme protocole de base pour découvrir les services Web géolocalisés. Toutefois, ce protocole n'offre pas de structure de données pour enregistrer les services Web géolocalisés dans le registre qui lui est associé. L'extension de la structure de données du registre UDDI, que nous appelons UDDIM (UDDI pour mobiles), permet d'y ajouter la topologie du système GLWSA.

Par contre, l'extension du protocole MLP permet d'ajouter des fonctionnalités et des structures de données de la localisation thématique d'un groupe de mobiles. L'extension du protocole MLP est appelée MLPe.

➤ *Registre UDDIM*

Le registre UDDIM est une extension du registre UDDI dans le but de gérer la topologie des nœuds du système GLWSA, d'assigner un service à un nœud particulier (matérialisé par la classe *AgreementNode*) et d'ajouter une qualité de service à un service spécifique (matérialisé par la classe *Agreement*). Pratiquement, nous ajoutons quatre structures de données XML à l'UDDI : *Node*, *Agreement*, *AgreementNode* et *PolygonLineSegment*. La représentation UML de ces éléments est donnée à la Figure 3.14. La structure de données de l'élément *Node* décrit les informations d'un nœud

GLWSM. Les paramètres qui identifient un nœud GLWSM sont l'identifiant du nœud *nodeId*, l'URL du nœud *urlNode*, le type de nœud (« racine » ou « feuille »), la position géographique du nœud (*xPos*, *yPos*, *zPos*), la distance maximum tolérée *distanceMaxNodeService* entre un SAS et le nœud *nodeId*, l'adresse du MAP *MAP_Addr* associée au nœud et le nombre de segments de droite *numberOfPolygonLineSegment* qui délimitent la région couverte par le nœud (nous considérons qu'une région couverte par un nœud a la forme d'un polygone).

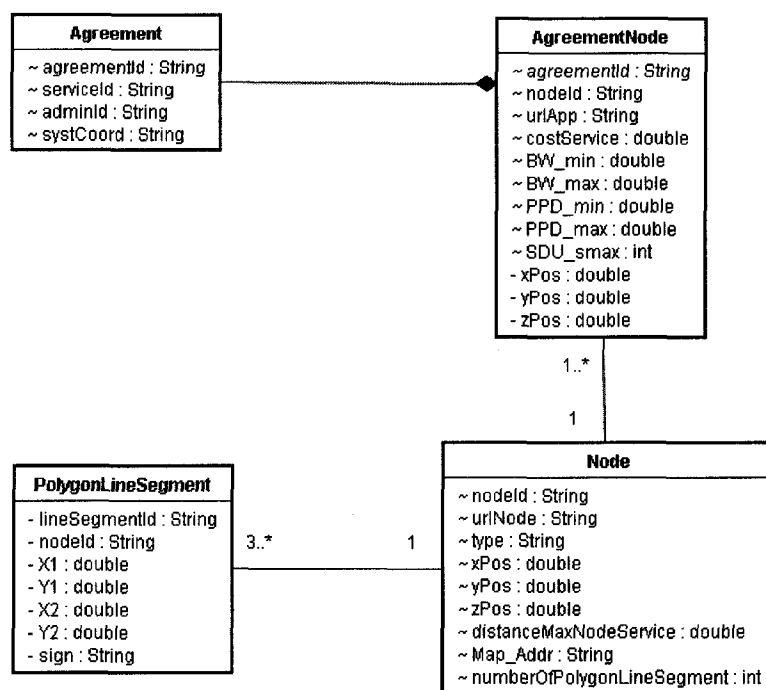


Figure 3.14 Structures de données de UDDIM

L'élément *Agreement* représente l'agrément d'un service spécifique. Il est composé d'un identifiant d'agrément *agreementId*, d'un identifiant de service *serviceId*, d'un identifiant de l'administrateur du service *adminId* et d'une référence de système de coordonnées utilisé *systCoord* (cartésien, longitude/latitude/altitude). L'élément *AgreementNode* matérialise la description détaillée d'un agrément de service à un nœud GLWSM spécifique. Il se compose de paramètres statiques de QoS (*costService*, *BW_min*, *BW_max*, *PPD_min*, *PPD_max*, *SDU_smax*). Les paramètres *BW_min* et

BW_max représentent le minimum et maximum de bande passante qu'un service *serviceId* peut offrir au nœud *nodeId*. Les paramètres *PPD_min* et le *PPD_max* représentent le minimum et maximum du délai sur le chemin de propagation d'un service *serviceId* au nœud *nodeId*. Le paramètre *SDU_smax* est la taille maximum d'une unité de service. Les paramètres (*xPos*, *yPos*, *zPos*) d'un élément *AgreementNode* matérialisent la position géographique d'un serveur SAS qui offre le service *serviceId* au nœud *nodeId*. L'élément *PolygonLineSegment* permet de représenter un segment de droite de la région couverte par le nœud *nodeId*. Les paramètres (*X1*, *Y1*) et (*X2*, *Y2*) représentent les coordonnées de deux points appartenant à une droite délimitant le polygone. Le signe *sign* de l'inéquation de *PolygonLineSegment* peut prendre une des valeurs "*<=*" ou "*>=*".

Les API que nous avons adjoints au registre UDDIM sont les suivants :

- **addNode** (Les paramètres de cette méthode sont tous les attributs de l'élément 'Node') ajoute un nœud GLWSM à la topologie ;
- **remove** (nodeKey: String) supprime un nœud de la topologie ;
- **getRootNode** (type: String) recherche un nœud racine de la topologie;
- **findNode** (nodeKey: String) recherche un nœud spécifique dans le registre UDDIM ;
- **createAgreement** (Les paramètres de cette méthode sont tous les attributs de l'élément 'Agreement') crée un nouvel objet d'accord pour un service spécifique ;
- **removeAgreement** (agreementId: String) supprime un accord spécifique ;
- **findAgreement** (agreementId: String) recherche un accord spécifique;
- **createAgreementNode** (Les paramètres de cette méthode sont tous les attributs de l'élément 'AgreementNode') crée un nouveau objet d'accord pour un service spécifique à un nœud particulier ;
- **removeAgreementNode** (agreementKey: String, nodeKey: String) supprime un accord spécifique à un nœud particulier;

- **findAgreementNode** (agreementKey: String, nodeKey: String) recherche un agrément spécifique à un nœud particulier ;
- **createPolygonLineSegment** (*Les paramètres de cette méthode sont tous les attributs de l'élément 'PolygonLineSegment'*) crée un nouvel objet *PolygonLineSegment* ;
- **removePolygonLineSegment** (lineSegmentId: String) supprime un segment de droite spécifique ;
- **findPolygonLineSegment** (lineSegmentId: String) recherche un segment de droite spécifique.

➤ ***Le protocole MLPe***

La localisation des clients mobiles est une fonctionnalité commune à toutes les applications géolocalisées. Avec la version courante (version 3.0) du protocole MLP, quand une application géolocalisée fait une requête de position sur un groupe de clients mobiles, elle envoie vers le serveur LCS une liste d'identifiants des mobiles. Nous avons analysé les requêtes de localisation sur les groupes de clients mobiles et avons constaté que les clients mobiles interrogés sont liés par un sujet thématique. Ainsi, nous avons factorisé les méthodes de localisation des groupes de clients mobiles sur une base thématique dans le but d'interagir. Cette localisation thématique nous a donc conduit à étendre ou à ajouter des fonctionnalités au protocole MLP courant.

La localisation thématique est importante car elle permet une réutilisation du code et réduit la taille des messages de localisation des groupes de mobiles envoyés sur le réseau. En effet, la factorisation thématique permet d'interroger un groupement de mobiles liés par un sujet et évite ainsi de mettre sur le réseau une liste des mobiles qui augmenterait le temps de transmission des requêtes au serveur LCS.

3.3.8 Typologie des applications géolocalisées

Afin de factoriser les services géolocalisés communs, nous avons établi une typologie des applications géolocalisées connues dans la littérature. Nous distinguons

principalement quatre catégories d'applications: services informationnels, suivi des traces d'un objet mobile, gestion de ressources, et navigation.

➤ ***Les services informationnels***

Les services informationnels permettent de questionner le serveur LCS (par le biais du serveur GLWSM) sur la position d'un objet mobile ou d'un groupement d'objets mobiles et de vérifier les conditions sur la position si nécessaire. Dans les requêtes transmises, les objets mobiles appartenant à un groupement sont liés par un sujet mobile commun. Dépendamment de la formulation de la requête, la position du requérant peut être exigée. Après analyse, nous avons classifié en trois catégories les services informationnels pouvant être factorisés:

- le sujet mobile; exemple : où se trouvent les bus de la STCUM ? Dans cette requête « bus de la STCUM » est le sujet mobile ;
- le sujet mobile et la précision de la position désirée ; exemple : quels sont les taxis de la compagnie « ABC » qui sont les plus proches de moi ?
- le sujet mobile et une zone géographique; exemple : quels sont les taxis de la compagnie *taxi Angrignon* se trouvant dans la *zone de localisation* 'idz' ?

À chaque service informationnel à factoriser, nous avons associé une méthode respective soit donc :

- ***locateSubject*** (*sujet : String*) permet de localiser la position de tous les mobiles liés au sujet interrogé ;
- ***locateSubjectNearOf*** (*sujet : String, position : Spatial, precision : float*) : permet de localiser la position de tous les mobiles du sujet interrogé dont la distance par rapport à la valeur « position » est inférieure ou égale à « precision » ;
- ***locateSubjectInZone*** (*sujet : String, idzone : Zone*) : permet de localiser la position de tous les mobiles du sujet interrogé se trouvant dans la zone «idzone».

➤ *Les services de suivi des traces d'un objet mobile*

Les services de suivi des traces d'un objet mobile consistent à s'intéresser aux événements de la position des objets mobiles (présence, entrée, sortie) dans une zone géographique à un moment donné. Sur les frontières de la zone de localisation observée, l'objet est entrant ou sortant. Les applications de suivi des traces d'un mobile sont des applications événementielles. Elles permettent de traquer le déplacement d'un mobile ou d'un groupe de mobiles. Les objets mobiles se déplacent de manière aléatoire dans un réseau mobile. Si les conditions énoncées dans la requête sont rencontrées, le serveur de position LCS notifie à l'application du fournisseur de service, via un GLWSM, la position des mobiles impliqués dans la requête. Les requêtes issues de ces applications événementielles sont réalisées en deux parties : la première partie consiste à envoyer les requêtes au LCS; la deuxième partie revient à écouter et à notifier les clients intéressés des résultats de la requête lorsque l'événement écouté est vérifié. Nous distinguons trois possibilités de factorisation thématique des services de suivi de traces :

- *événements d'entrée d'un membre du sujet dans une zone géographique; exemple : m'informer quand un membre des *médecins ambulants* entre dans une nouvelle zone de localisation ?*
- *événement de sortie d'un membre du sujet d'une zone géographique; exemple : m'informer quand un membre des *médecins ambulants* sort d'une zone de localisation ?*
- *événement de coïncidence ou de collocalisation des membres d'un sujet mobile avec un objet mobile spécifique; exemple : quels sont les taxis de la compagnie « ABC » se trouvant dans la même zone de localisation que moi ?*

À chaque service de suivi de traces à factoriser, nous avons associé une méthode respective soit donc :

- *ifEntry* (*sujet: String, idzone: Zone*) : permet de trouver les mobiles du sujet interrogé entrant dans la zone de localisation « idzone » et de notifier l'entrée du mobile à l'application du fournisseur de services.
 - *notifyEntry* (*msid: String*)
- *ifExit* (*sujet: String, idzone: Zone*) : permet de trouver les mobiles du sujet interrogé sortant de la zone de localisation « idzone » et de notifier la sortie du mobile à l'application du fournisseur de services.
 - *notifyExit* (*msid: String*)
- *collocate* (*msid: String, sujet: String*) : permet de trouver les mobiles du sujet interrogé se trouvant dans la même zone de localisation que « msid ».

➤ ***Les services de gestion de ressources***

Les services de gestion de ressources sont des applications qui utilisent la mesure de certaines grandeurs physiques afin de gérer les ressources d'un client mobile ou d'un groupe de clients mobiles. Les applications de gestion de ressources possibles sont par exemple la gestion de la logistique des sujets mobiles (voitures de transport ou de livraison de marchandises) ou la gestion des compétences des sujets mobiles (les médecins ambulants spécialisés en orthopédie). Les services de gestion de ressources nécessitent parfois que les objets mobiles soient équipés d'un équipement autre que le téléphone mobile pour mesurer la grandeur physique observée (par exemple : une balance, une vidéo, un thermomètre, un baromètre, etc.). Quelques exemples de ces applications:

- gestion de la logistique des objets mobiles; exemple : trouver le taxi de la compagnie «ABC» le plus proche du client X, n'ayant pas de passager, puis lui communiquer l'adresse actuelle du client X ?
- gestion des compétences; exemple : trouver dans un rayon de 50 kilomètres de ma position actuelle, le médecin ambulant ayant une spécialité en orthopédie et demander qu'il se rende à la localité « XYZ »?

La factorisation des sujets mobiles relevée dans les services de gestion de ressources est similaire au niveau traitement aux services informationnels. En effet, les applications de gestion de ressources sont traitées en quatre phases: localiser des clients mobiles du sujet observé, vérifier la condition sur la position, vérifier la condition sur la grandeur physique, sélectionner le membre du sujet remplissant les conditions (si nécessaire), puis donner les directives à suivre. Le GLWSM n'est concerné que par les phases 1 et 2. Vu sous cet angle, les services de gestion de ressources sont similaires au niveau traitement aux services informationnels du point de vue d'un GLWSM.

➤ ***Les services navigationnels***

Les services navigationnels indiquent à un client le meilleur chemin ou un trajet à suivre d'un point de départ à un point d'arrivée. Ils permettent également de connaître l'état de la circulation dans une zone de localisation spécifiée. Ils utilisent des informations SIG (*Services d'Informations Géographiques*) pour cartographier ou représenter une région du globe terrestre. Ils interrogent le serveur LCS via le GLWSM sur la position géographique d'un client mobile ou d'un groupe de clients mobiles. En ce qui a trait aux applications implémentant l'état de la circulation, les positions des mobiles interrogés peuvent être par la suite indiquées sur la carte géographique correspondante à la région où ils se trouvent. Quelques exemples de services navigationnels:

- quel est le meilleur trajet pour se rendre à New York à partir de ma position actuelle?
- m'informer continuellement de la position des unités mobiles membres du service *état de circulation* qui se trouvent sur l'autoroute A-20 dans la zone de Dorval ?

Par rapport au GLWSM, les services navigationnels reviennent à s'informer sur la position d'un mobile ou d'un groupe de mobiles en donnant ou non sa propre position et/ou à vérifier une condition événementielle. Cela signifie, au niveau traitement, que les services navigationnels utilisent les mêmes fonctionnalités que les services

informationnels ou de suivi de traces. Il n'y a donc pas de nouvelles fonctions à factoriser.

Nous avons ajouté au protocole MLPe des fonctionnalités API pour gérer la structure des données des éléments permettant de maintenir une recherche thématique des clients mobiles, soit : 'Theme' et 'ThemeMsid' (Figure 3.15).

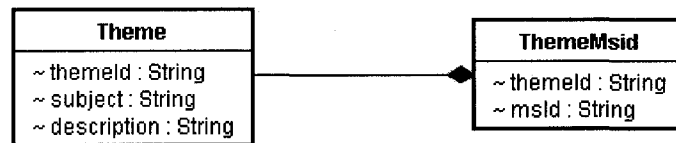


Figure 3.15 Structure de données de MLPe

Les API suivant ont été ajoutés pour la gestion de ces éléments:

- **addTheme**(themeId:String, subject: String, description: String) pour ajouter un objet Theme ;
- **removeTheme**(themeId: String) pour supprimer un objet Theme ;
- **findTheme** (themeId: String) pour rechercher un thème;
- **addThemeMsid** (themeId: String, msid: String) pour ajouter un objet ThemeMsid ;
- **removeThemeMsid** (themeId: String,msid: String) pour supprimer un objet ThemeMsid ;
- **findAllMsidOfTheme** (themeId: String) pour rechercher tous les identifiants des mobiles ayant souscrit au thème 'themeId'.

3.4 Formalisme mathématique du système GLWSA

Afin de faciliter l'implémentation les interactions des principales fonctionnalités du système avec les bases de données UDDIM et « ClientInfosDB », nous avons formalisé de manière algébrique le système GLWSA [13].

3.4.1 Ensembles des serveurs de découverte des services et de zones de localisation

Soit E l'ensemble des serveurs de découverte des services $GLWSM$ déployés aux feuilles de l'arbre topologique. Par définition:

$$E = \{GLWSM_1, GLWSM_2, \dots, GLWSM_p\} \quad (3.3)$$

où p représente le cardinal de E ou le nombre de $GLWSM$ déployés dans la topologie.

Soit Z l'ensemble des zones de localisation d'un réseau mobile couvertes par la topologie. On a :

$$Z = \{Z_1, Z_2, \dots, Z_t\} \quad (3.4)$$

où t représente le cardinal de Z ou le nombre de zones de localisation du réseau mobile associées à la topologie.

➤ Définitions

Réseau d'accès aux paquets: Un réseau d'accès aux paquets est un ensemble de ressources du réseau mobile permettant d'accéder au réseau IP. Les zones de localisation couvertes par un réseau d'accès aux paquets constituent un sous-ensemble de l'ensemble Z partageant un même MAP (*Mobile Anchor Point*).

Domaine: Nous appelons domaine un réseau d'accès aux paquets contrôlé par un nœud $GLWSM$.

Contiguïté: Deux serveurs de découvertes $GLWSM_j$ et $GLWSM_{j+1}$ sont dits contigus s'ils ont une frontière commune de zones de localisation.

Nœud racine: Nous définissons par nœud racine $GLWSM_r$, le serveur de découverte de services au sommet de la topologie, qui permet la publication des services Web, la gestion de la topologie des nœuds $GLWSM$ et l'enregistrement de tous les services Web publiés dans le système $GLWSA$.

Nœud de résidence: Nous définissons par nœud de résidence $GLWSM_h$, un serveur de découverte de services à la feuille de l'arbre de la topologie où un client mobile effectue les opérations d'authentification et d'autorisation afin d'utiliser les ressources du système $GLWSA$.

Nœud visité: Pour un client mobile donné, nous définissons par nœud visité $GLWSM_v$, un serveur de découverte de services à la feuille de l'arbre de la topologie différent du nœud de résidence $GLWSM_h$ du client mobile concerné.

➤ **Propriétés**

Ensembles finis: À tout instant, les ensembles E et Z sont des ensembles finis et déterminés. Le cardinal de E ou de Z peut s'accroître ou diminuer dépendamment s'il y a un ajout ou un retrait d'un nœud $GLWSM_i$ ou d'une zone de localisation Z_x dans la topologie.

Connaissance de la topologie: Le nœud racine est l'unique serveur de découverte des services qui gère (ajoute, retire, modifie) la structure topologique déployée dans le système $GLWSA$.

Couverture d'un nœud fils $GLWSM_i$: Chaque nœud fils $GLWSM_i$ est associé à un domaine D_i . On dit alors que $GLWSM_i$ couvre les zones de localisation de D_i .

Disjonction de couverture: Deux nœuds fils distincts $GLWSM_i$ et $GLWSM_j$ couvrent des zones de localisation disjointes. Ainsi, si ZL_i et ZL_j représentent les zones de localisation couvertes respectivement par $GLWSM_i$ et $GLWSM_j$, alors :

$$ZL_i \cap ZL_j = \phi \quad (3.5)$$

3.4.2 Ensembles des services déployés et des serveurs d'applications de la topologie

Soit F l'ensemble des serveurs d'applications des fournisseurs SAS déployés dans la topologie. Par définition :

$$F = \{SAS_1, SAS_2, \dots, SAS_m\} \quad (3.6)$$

où m représente le cardinal de F ou le nombre de SAS déployés dans la topologie.

Soit G l'ensemble des services Web déployés dans la topologie. Par définition :

$$G = \{S_1, S_2, \dots, S_z\} \quad (3.7)$$

où z représente le cardinal de G ou le nombre de services Web S_i déployés dans la topologie.

➤ **Définitions**

Vecteur de services: Nous appelons vecteur de services $\vec{V}_k$, l'ensemble des services déployés au nœud $GLWSM_k$. On a :

$$\vec{V}_k = [S_a S_b \dots S_g] \bullet \overline{GLWSM_k} \quad (3.8)$$

où $\{S_a, S_b, \dots, S_g\}$ est un sous ensemble de G .

Présence de services: Un service Web S_j est présent dans un nœud $GLWSM_k$ si et seulement si S_j est fourni par au moins un SAS associé à $GLWSM_k$. Par conséquent, si S_j est présent y fois dans $GLWSM_k$, alors la mesure de présence p est :

$$p = y \quad (3.9)$$

Vecteur de présence des services: Nous appelons vecteur de présence des services $\vec{PV}_k$, le vecteur matérialisant l'ensemble des services présents au nœud $GLWSM_k$.

$$\vec{PV}_k = [p_{1k} p_{2k} \dots p_{zk}] \bullet \overline{GLWSM_k} \quad (3.10)$$

où $p_{1k}, p_{2k}, \dots, p_{zk}$ représentent la présence des services $S_1, S_2, \dots, S_z$ de l'ensemble G dans $GLWSM_k$.

Matrice de présence des services: Nous appelons matrice de présence des services, la matrice M permettant d'identifier la présence des services à tout nœud $GLWSM$ de la topologie.

$$M = \begin{bmatrix} p_{11} & p_{12} & p_{1p} \\ p_{21} & p_{22} & p_{2p} \\ p_{z1} & p_{z2} & p_{zp} \end{bmatrix} \quad (3.11)$$

où p_{ik} représente la présence du service S_i au serveur $GLWSM_k$.

➤ **Propriétés**

Ensembles finis: À tout instant, les ensembles F et G sont des ensembles finis et déterminés. Le cardinal de F ou de G peut s'accroître ou diminuer, dépendamment s'il y a un ajout ou un retrait d'un serveur d'applications SAS_i ou d'un service Web S_x de la topologie.

Nombre de services offerts: Un serveur d'application SAS_k peut offrir un ou plusieurs services web S_i .

Nœud racine et matrice de présence des services: À tout instant, le nœud racine $GLWSM_r$ connaît entièrement la matrice de présence des services M .

➤ **Relations**

Association serveur d'application et serveurs de découverte des services: Chaque serveur d'applications SAS_k est associé à un serveur de découverte de services (nœud fils) $GLWSM_i$, et éventuellement à d'autres $GLWSM_c$ contigus à $GLWSM_i$ si sa distance par rapport aux nœuds de découverte des services est inférieure ou égale à la distance maximale tolérée entre un SAS et un nœud associé $GLWSM$ dans la topologie $GLWSA$.

$$D(SAS_k, GLWSM_i) \leq D_{\max} \quad (3.12)$$

où $D(SAS_k, GLWSM_i)$ est la distance entre SAS_k et $GLWSM_i$.

3.4.3 Publication d'un service Web dans le système $GLWSA$

La publication d'un service Web S_{z+1} dans le système $GLWSA$ implique trois opérations essentielles :

1. *unir* l'ensemble G et le singleton $\{S_{z+1}\}$, soit :

$$G = G \cup \{S_{z+1}\} = \{S_1, S_2, \dots, S_z, S_{z+1}\} \quad (3.13)$$

cette opération consiste au niveau de la base de données UDDIM à ajouter au nœud racine $GLWSM_h$ un enregistrement aux tables représentant le service et l'agrément à un service ;

2. *ajouter* une ligne L_{z+1} à la matrice de présence M , soit :

$$M = \begin{bmatrix} p_{11} & p_{12} & p_{1p} \\ p_{21} & p_{22} & p_{2p} \\ p_{(z+1)1} & p_{(z+1)2} & p_{(z+1)p} \end{bmatrix} \quad (3.14)$$

cette opération consiste au niveau de la base de données UDDIM à ajouter à tout nœud feuille de découverte de service $GLWSM_i$ ayant une présence de service $p_{z+1,i}$ non nulle un nouvel enregistrement aux tables représentant le service S_{z+1} , l'agrément au dit service, et l'agrément dudit service au nœud spécifique $GLWSM_i$;

3. pour tout nœud $GLWSM_i$ ayant une présence de service $p_{z+1,i}$ non nulle, *confirmer* l'enregistrement des informations publiées au nœud racine $GLWSM_h$.

3.4.4 Recherche d'un service Web dans le système GLWSA

La recherche d'un service Web S_d dans le nœud de résidence $GLWSM_h$ d'un client mobile CM implique six opérations essentielles :

1. *localiser* le client mobile CM en formulant une requête de position au serveur de localisation LCS ;
2. *déterminer* le serveur $GLWSM_{plusproche}$ couvrant la position géographique courante POS_c du client mobile ;

Cela équivaut en langage de base de données algébrique à sélectionner dans la table des nœuds $GLWSM$ le nœud $GLWSM_s$ qui couvre la position géographique courante POS_c du client. Soit l'instruction suivante:

$$\{T[GLWSM] \text{ where } POS_c \subset AREA\} \quad [GLWSM_s] \quad (3.15)$$

où $AREA$ matérialise la région géographique couverte par $GLWSM_s$.

3. *vérifier* dans $UDDIM_h$ (associé au nœud residence $GLWSM_h$) si la présence du service S_d est non nulle; du point de vue de la base de données, cette opération consiste à faire une projection sur l'axe des nœuds $GLWSM$ de la

table $T[S, GLWSM, ADRESSE]$ identifiant l'agrément d'un service à un nœud quand le service a pour valeur S_d . Soit l'instruction suivante:

$$\{T[S, GLWSM, ADRESSE] \text{ where } S = S_d\} \quad [GLWSM] \quad (3.16)$$

4. *sélectionner* tous les serveurs SAS associés à $GLWSM_{plusproche}$ et offrant le service concerné S_d ; cette opération consiste à faire une projection sur l'axe des adresses de service $ADRESSE$ de la table $T[S, GLWSM, ADRESSE]$ identifiant l'agrément d'un service à un nœud quand le service a pour valeur S_d . Soit l'instruction suivante:

$$\{T[S, GLWSM, ADRESSE] \text{ where } S = S_d\} \quad [ADRESSE] \quad (3.17)$$

5. *vérifier* si les critères de qualité de service rencontrent le niveau requis par le client mobile; cette phase est optionnelle si le client mobile n'a pas requis la QoS dans la requête découverte de services ;
6. *retourner* au client mobile CM , l'adresse du service S_d rencontrant le niveau de QoS requis.

3.4.5 Coordination de la migration d'un service Web dans le système GLWSA

La coordination de la migration d'un service Web S_d en exécution dans un serveur d'applications courant SAS_c associé au nœud courant $GLWSM_c$, débute lorsque le nœud $GLWSM_c$ est notifié par le serveur de localisation LCS , de la sortie du client mobile CM (impliqué dans l'exécution du service S_d), des zones de localisation couvertes par $GLWSM_c$. Les étapes suivantes sont exécutées par la suite :

1. *déterminer* le prochain nœud $GLWSM_p$ couvrant la position géographique courante POS_c du client mobile; cette opération consiste à vérifier lequel des nœuds $GLWSM$ de l'ensemble E couvre la position géographique courante du client. Cela équivaut en langage de base de données algébrique à l'instruction 3.15.

2. *vérifier* l'existence de S_d et *sélectionner* tous les serveurs SAS associés à $GLWSM_p$ et offrant le service S_d ; cela équivaut en langage de base de données algébrique à l'instruction 3.17 ;
3. *vérifier* si les critères de qualité de services rencontrent le niveau requis par le client mobile; cette phase est optionnelle si le client mobile n'a pas requis la QoS dans la requête découverte de services ;
4. *sélectionner* le prochain serveur d'applications SAS_p où la migration de service sera effectuée ;
5. le $GLWSM_c$ *notifie* à SAS_c l'adresse de migration du service S_d dans le serveur SAS_p ;
6. le SAS_c *notifie* à $GLWSM_c$ la fin de migration du service S_d dans le serveur SAS_p ;
7. le $GLWSM_c$ *envoie* à $GLWSM_p$ un message de continuer le suivi de la position du client CM des zones de localisation qu'il couvre.

CHAPITRE IV

IMPLÉMENTATION DU SYSTÈME GLWSA

Dans ce chapitre, nous allons décrire l'architecture fonctionnelle et l'implémentation du système GLWSA. Cette description nous permettra de découvrir l'organisation logique, les objets et les fonctionnalités externes ou API (*Application Program Interface*) du système. Dans un premier temps, nous allons présenter une vue globale de l'architecture fonctionnelle, puis nous décrirons les différentes couches de l'architecture, ensuite nous présenterons le modèle de persistance du système et les extensions faites au protocole MLP.

4.1 Architecture fonctionnelle du système GLWSA

L'architecture fonctionnelle du système GLWSA est illustrée à Figure 4.1.

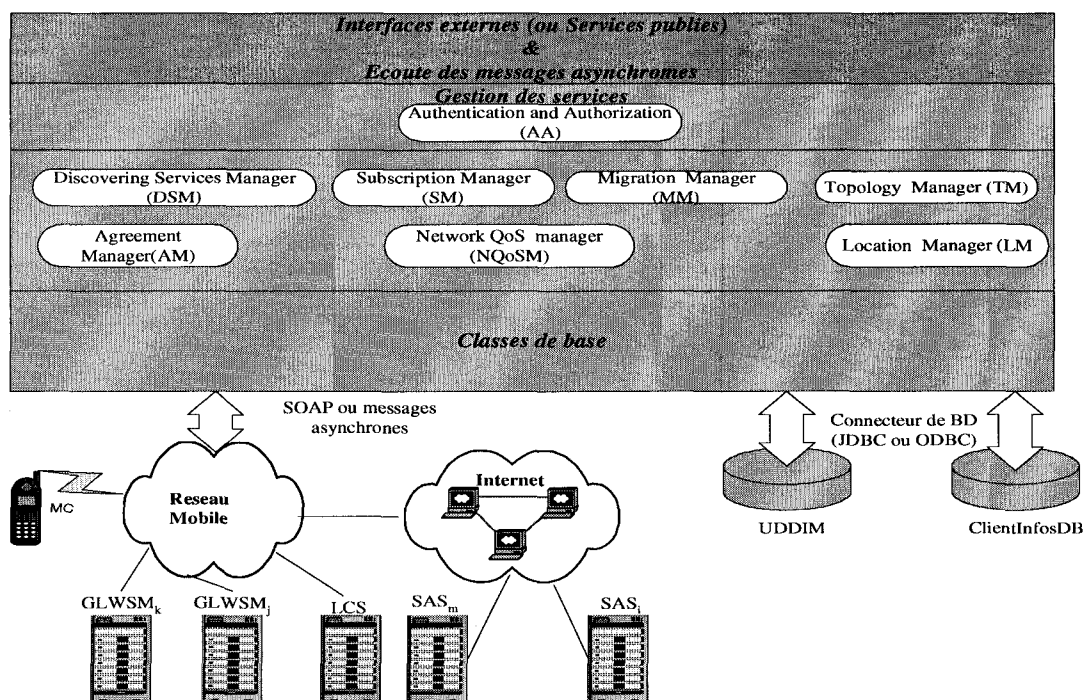


Figure 4.1 Architecture fonctionnelle du système GLWSA

L'architecture fonctionnelle proposée est orientée services et est subdivisée en trois couches : la couche des services publiés synchrones et d'écoute des messages asynchrones, la couche de gestion des services et la couche des classes de base [11], [13], [14].

Nous entendons par architecture orientée services, une architecture qui consiste à développer les applications comme des composants logiciels indépendants ayant des interfaces externes de services avec lesquels les usagers interagissent. Une telle architecture permet aussi de sélectionner des services désirés dans une collection donnée et de les assembler afin de réaliser une caractéristique fonctionnelle spécifique [20].

4.2 La couche des services publiés synchrones et d'écoute des messages asynchrones

La couche des services publiés synchrones et d'écoute des messages asynchrones permet d'exposer certaines méthodes de la couche de gestion des services que l'on souhaite rendre accessibles aux usagers du système et d'écouter les messages asynchrones échangés entre différents systèmes GLWSM. Le terme synchrone est utilisé pour mentionner le fait que le requérant d'une requête de service attend la réponse du serveur avant de continuer l'exécution de l'application; le terme asynchrone est utilisé pour mentionner le fait que le requérant n'attend pas la réponse à une requête de service soumise au serveur avant de continuer l'exécution de l'application.

4.2.1 Les services publiés ou interfaces externes synchrones

Pour les services publiés ou interfaces externes synchrones, les opérations d'exposition des méthodes consistent principalement à déployer ou à publier, dans un service Web ou dans une interface externe, les méthodes d'un objet de la couche de gestion des services. La gestion du déploiement des services Web est réalisée par le biais d'un conteneur Web. Le conteneur Web utilisé dans l'implémentation du système GLWSA est le conteneur Axis Apache version 1.2.

Pour déployer les méthodes d'un ou de plusieurs objets de la couche de gestion des services, nous avons préalablement créé un fichier de description et de déploiement des services Web que nous souhaitons exposer. Ce fichier est connu dans la littérature sous le sigle WSDD (*Web Service Deployment Descriptor*). Par exemple, le fichier de description et de déploiement des services Web ci-dessous de la Figure 4.2 permet de déployer les services Web des gestionnaires de découverte des services et de la notification de migration des services.

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="DvtService" provider="java:RPC">
    <parameter name="className" value="uddi.manager.DvtServiceManager"/>
    <parameter name="allowedMethods" value="*/>
    <parameter name="scope" value="Session"/>
    <beanMapping qname="ns:Node" xmlns:ns=" uddi:datatype."
      languageSpecificType="java: uddi.datatype.Node"/>
    serializer="org.apache.axis.encoding.ser.ElementSerializerFactory"
    deserializer="org.apache.axis.encoding.ser.ElementDeserializerFactory"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
  </service>
  <service name="MigrationNotification" provider="java:RPC">
    <parameter name="className" value="uddi.notification.MigrationNotification"/>
    <parameter name="allowedMethods" value="migrateTo"/>
    <parameter name="scope" value="Session"/>
  </service>
</deployment>
```

Figure 4.2 Exemple de contenu d'un fichier de description et de déploiement des services Web

L'élément « *deployment* » permet d'indiquer au conteneur Axis Apache que le fichier XML à interpréter est un fichier de description et de déploiement des services Web et de préciser le chemin du répertoire où se trouve les documents XML permettant

de transformer les messages SOAP reçus en objets Java. L'élément « *service* » permet de définir le service Web (par exemple « *service name="DvtService"* » pour la découverte des services ou « *service name="MigrationNotification"* » pour notifier la migration des services) que l'on veut publier ou déployer et d'indiquer comment sont effectués les appels de méthodes à distance pour un langage de programmation donné. Par exemple, « *provider="java:RPC"* » signifie qu'on utilise les appels de procédures à distance RPC (*Remote Procedure Call*) pour appeler les méthodes Java qui sont exposées dans un service Web.

Les paramètres d'un service Web permettent d'indiquer la classe dont on veut exposer les méthodes (par exemple `<parameter name="className" value="uddi.manager.DvtServiceManager"/>` pour le gestionnaire de découverte des services résidant dans le package «*uddi.manager*»), les méthodes qu'on veut publier (par exemple `<parameter name="allowedMethods" value="*/>` signifie que toutes les méthodes de la classe exposée sont publiées ou `<parameter name="allowedMethods" value="migrateTo"/>` signifie que seule la méthode «*migrateTo*» est publiée) et la portée d'un objet créé dans la couche de gestion des services. La portée d'un objet peut être: de type application (`<parameter name="scope" value="Application"/>`, ce qui revient à créer un objet unique partagé par tous les clients utilisant l'application); de type session (`<parameter name="scope" value="Session"/>`, ce qui revient à créer un objet pour chaque nouvelle session d'un client); de type requête (`<parameter name="scope" value="Request"/>`, ce qui revient à créer un objet pour chaque requête effectuée par un client).

Si la classe de gestion de services exposée utilise des classes complexes (classes différentes des types de base (entier, réel, chaîne de caractères, etc.) d'un langage de programmation), il faudra les déclarer dans le fichier WSDD afin de permettre au conteneur Web d'effectuer la correspondance entre les messages SOAP reçus et les classes complexes (par exemple, `beanMapping qname="ns3:Node" xmlns:ns3="uddi:datatype"` permet de déclarer l'élément « *node* » ainsi que son répertoire « *uddi:datatype* » et `languageSpecificType="java:uddi.datatype.Node"` permet

d'effectuer la correspondance avec la classe « *Node* » localisée dans le package « *uddi.datatype* » du projet Java).

L'ensemble des interfaces externes synchrones déployées dans le système GLWSA est récapitulé au Tableau 4.1.

Tableau 4.1 Interfaces externes synchrones du système GLWSA

Interface externe	Classe de gestion de service associée	Description
DvtService	DvtServiceManager	Offre des API permettant de rechercher un service le plus proche avec ou sans QoS, de publier un service.
AgreementService	AgreementManager	Offre des API permettant de rechercher, supprimer ou de créer un agrément de service et un agrément à un nœud. Il offre aussi la possibilité de créer des nœuds dans les feuilles de l'arbre.
SubscriptionService	SubscriptionManager	Offre des API permettant de rechercher, supprimer ou de créer un abonnement, un usager et son profil machine.
LocationService	LocationManager	Offre un API permettant de rechercher la position d'un client mobile.
TopologyService	TopologyManager	Offre des API permettant de rechercher, supprimer ou de créer un nœud à partir du GLWSM racine.

4.2.2 Écoute des messages asynchrones

L'écoute des messages asynchrones dans un serveur GLWSM consiste principalement à écouter les messages qui sont délivrés aux queues de destination d'un courtier de messages (communément appelé *message broker*). Pour échanger des messages asynchrones entre deux systèmes GLWSM, la technologie « Sun Java System Message Queue 3.5 » a été utilisée car elle s'intègre bien au langage de programmation Java. Fonctionnellement, un message de service est envoyé par un serveur GLWSM (un émetteur est identifié dans la littérature comme un producteur), à un courtier de messages. Ce message de service est écrit dans une queue destination du courtier qui est écoutée par un ou plusieurs clients GLWSM (un récepteur est identifié dans la littérature comme un consommateur) [86]. Chaque courtier de message a une adresse HTTP associée (par exemple, nous avons utilisé <http://localhost:7676> pour le suivi des mobiles). Dans un serveur GLWSM, nous distinguons principalement trois types de

messages asynchrones échangés (voir Tableau 4.2): le message de suivi des mobiles, le message de publication des services et le message de propagation des paramètres de QoS collectés.

Tableau 4.2 Messages asynchrones échangés dans le système GLWSA

Message de service	Producteurs	Consommateurs	Courtiers
Suivi des mobiles	La feuille de nœud GLWSM courant où le mobile était localisé.	La prochaine feuille de nœud GLWSM où la localisation du mobile a été détectée.	Un objet <i>courtierSuiviMobile</i> est associé à chaque GLWSM impliqué dans le message de suivi de mobiles.
Publication d'un service	Le nœud GLWSM racine	Toutes les « feuilles » de nœuds GLWSM où le service sera publié.	Au nœud GLWSM racine, un objet <i>courtierPublicationService</i> est créé, et pour chaque service publié dans le système GLWSA, une queue de publication associée est créée.
Propagation des paramètres de QoS collectés pour service spécifique	Toute « feuille » de nœud GLWSM qui fournit ledit service.	Toutes les autres « feuilles » de nœuds GLWSM (hormis le producteur) où le service est publié.	Pour chaque « feuille » de nœud GLWSM fournissant ledit service, un objet <i>courtierPropagationService</i> et une queue associée sont créés.

4.3 La couche de gestion des services

La couche de gestion des services est la couche qui implémente les services publiés et les messages de services écoutés. Elle est constituée des classes suivantes :

- la classe d'authentification et d'autorisation ;
- la classe de gestion de la topologie ;
- la classe de gestion d'agrément de services ;
- la classe de gestion des abonnements ;
- la classe de gestion de la qualité de service réseau ;
- la classe de gestion de découverte des services ;
- la classe de gestion de migration ;
- et la classe de gestion de localisation.

4.3.1 Authentification et autorisation

La classe d'authentification et d'autorisation permet d'identifier ou d'authentifier un usager, puis de l'autoriser à utiliser les objets du système. En effet, elle possède une méthode *verifyCredentials* qui permet de vérifier dans la base de données "ClientInfosDB", l'existence d'un usager ayant le nom d'usager et le mot de passe indiqués. Dépendamment du rôle de l'usager dans le système GLWSA, elle permet en cas de succès d'avoir accès aux objets de gestion de services autorisés. Dans l'implémentation du système GLWSA, on distingue trois rôles : *administrateur*, *fournisseur* et *client*. Un administrateur est autorisé à utiliser tous les objets de gestion de service du système dans un environnement de test. Toutefois, dans un environnement opérationnel ou de production, son rôle se limite à la gestion de la topologie et des agréments de services. Un fournisseur est autorisé à utiliser les objets de gestion de découverte de services afin de publier les services et les objets de gestion d'abonnement dans le but de gérer les clients. Un client est autorisé à utiliser les objets de gestion de découverte de services afin de rechercher un service et les objets de gestion de localisation dans le but de localiser un ou plusieurs mobiles.

4.3.2 Gestion de la topologie

La Gestion de la topologie « TopologyManager » permet d'ajouter, de supprimer, de modifier ou de rechercher un nœud GLWSM et ses zones de localisation dans la topologie. Cette classe interagit avec la base de données UDDIM qui est une extension du registre UDDI pour les mobiles. Pour implémenter le gestionnaire de la topologie, nous avons utilisé le module jUDDI [46] (java UDDI) qui implémente les spécifications du registre UDDI. Nous avons modifié le module jUDDI afin d'ajouter la structure de données de la topologie des nœuds GLWSM et d'exposer les API du gestionnaire de la topologie sous forme de services Web. Concrètement, la classe TopologyManager offre huit services Web (voir Annexe, Figure A.1, *addNode*, *removeNode*, *findNode* et *getRootNode*) pour gérer les nœuds (classe *Node*).

Les méthodes *addNode*, *removeNode* et *findNode* permettent d'ajouter, de supprimer et de rechercher un nœud. La méthode *getRootNode* permet de retourner les informations sur le nœud Racine de la topologie. Pour se connecter au registre UDDIM, nous utilisons une instance unique d'un objet *java.sql.Connection* en appelant la méthode statique *ConnectionUddiDB.getConnection()*.

4.3.3 Gestion des agréments de service

La classe de gestion des agréments de service « *AgreementManager* » permet d'ajouter, de supprimer, de modifier ou de rechercher un agrément de service et un agrément de nœud dans la base de données UDDIM. Pour implémenter le gestionnaire des agréments de service, nous avons modifié le module jUDDI afin d'ajouter la structure de données d'un agrément de service (*Agreement*) et d'un agrément de service à un nœud (*AgreementNode*), et d'exposer les API du gestionnaire d'agrément de service sous forme de services Web. Concrètement, la classe *AgreementManager* offre huit méthodes (voir Annexe, Figure A.2, *createAgreementSvc*, *removeAgreement*, *findAgreement*, *getRootNode*, *createAgreementNode*, *removeAgreementNode*, *findAgreementNode* et *loadAllNodeForAgreement*) pour gérer les agréments de service et les agréments de nœud.

Les méthodes *createAgreement*, *removeAgreement* et *findAgreement* permettent d'ajouter, de supprimer et de rechercher un agrément de service. Les méthodes *createAgreementNode*, *removeAgreementNode* et *findAgreementNode* permettent d'ajouter, de supprimer et de rechercher un agrément de nœud. La méthode *loadAllNodeForAgreement* permet de charger à partir de la base de données UDDIM les nœuds d'un agrément de service spécifique. La méthode *getInstance* retourne l'instance unique de la classe *AgreementManager*. Pour se connecter au registre UDDIM, nous utilisons une instance unique d'un objet *java.sql.Connection* en appelant la méthode statique *ConnectionUddiDB.getConnection()*.

4.3.4 Gestion de la QoS dans le réseau

Le gestionnaire de la QoS dans le réseau « *NetworkQoSManager* » a trois fonctionnalités majeures :

- la collecte des données dynamiques de QoS (la bande passante dans le réseau *BW* et le facteur d'utilisation d'un serveur d'applications *UF*) ;
- la réception des données de QoS collectées par le GLWSM requérant ;
- la propagation des paramètres dynamiques de QoS collectés pour un service donné *S* à un nœud GLWSM_i vers d'autres nœuds GLWSM_k offrant le même service *S*.

➤ *La collecte des données dynamiques de QoS*

Le principe de la collecte des données dynamiques de QoS a été décrit au chapitre précédent. Pour implémenter ce principe, nous avons utilisé le simulateur réseau NS-2.27 (Network Simulator version 2.27) et avons modifié le module RSVP en utilisant les langages C++ et TCL (*Tool Command Language*), comme le recommandent les spécifications d'une simulation dans NS [62]. Le module RSVP a été choisi car il s'adapte bien à la commande QUOT_REQ qui peut être associé à la commande PATH de RSVP. Pratiquement, nous avons modifié la méthode *process_path_message* afin de traquer l'évolution de la commande PATH dans les différents nœuds (routeurs) traversés et calculer la bande passante disponible minimum sur le chemin traversé d'un serveur d'applications SAS au MAP.

Nous avons également ajouté une classe appelée *QuotationBW* au module RSVP (Figure 4.3). Cette classe implémente les commandes COLLECT_TRAF, QUOT_REQ, TRANSMIT_QUOT et RELEASE_SES. Les commandes QUOT_REQ, TRANSMIT_QUOT et RELEASE_SES sont implémentées respectivement par le biais des méthodes *collectTraffic*, *quotationRequest*, *transmitQuotation* et *releaseSession* décrites ci-dessous. La méthode *collectTraffic* permet à un GLWSM d'envoyer une requête de collecte des paramètres de QoS (bande passante *BW*, facteur d'utilisation *UF* d'un SAS) à un SAS en interagissant avec la méthode *sendCollectTraffic*. Les

paramètres transmis par la commande COLLECT_TRAFFIC sont l'adresse *mapAddr* du MAP associé au GLWSM requérant et la période de rafraîchissement des données collectées *DR*. À la réception de cette commande, le serveur d'applications SAS exécute la procédure de la commande QUOT_REQ et celle de la détermination du facteur d'utilisation *UF* qui n'est pas implémentée par le biais du simulateur NS-2.27. L'appel de la méthode *quotationRequest* est transmise à la méthode *sendQuotationRequest* qui interagit avec le message PATH de RSVP afin d'envoyer la requête initiée par un SAS au MAP. Les paramètres transmis par la méthode *quotationRequest* sont : l'adresse de destination du nœud *dest*, la bande passante moyenne du « bucket » *rate* du serveur d'applications SAS, la taille *bucket* du « bucket » du serveur SAS et la durée de vie ou la période de temps *ttl* nécessaire au rafraîchissement du message PATH de RSVP.

```

void QuotationBW :: collectTraffic(nsaddr_t mapAddr){
ct=new CollectTraffic(mapAddr);
ct->setApp(this);
sendCollectTraffic(qr);
}
void QuotationBW :: quotationRequest(nsaddr_t dest, double rate, long bucket, char ttl){
qr =new QuotationRequest(dest,rate,bucket,ttl);
qr->setApp(this);
sendQuotationRequest(qr);
}
void QuotationBW :: releaseSession(int session){
Tcl& tcl = Tcl::instance();
tcl.evalf("releaseSession %d", session);}
void QuotationBW :: transmitQuotation(int session, float BW){
tq =new TransmitQuotation(session, BW);
tq->setApp(this);
sendTransmitQuotation(tq);
}

```

Figure 4.3 Extrait de l'implémentation de QuotationBW

Au niveau de chaque nœud traversé, nous calculons et enregistrons la bande disponible sur les liens aussi bien que le minimum de la bande passante du chemin traversé. Ainsi, à la fin du parcours du message PATH au nœud MAP, nous avons la soumission sur la bande passante *BW* disponible sur le chemin. Cette soumission est renvoyée au SAS par le biais de la méthode *transmitQuotation* qui transmet la bande

passante *BW* et l'identifiant de la session RSVP *session* en faisant appel à la méthode *sendTransmitQuotation*. À la réception du résultat de soumission, le serveur SAS exécute la méthode *releaseSession* qui libère la session RSVP préalablement établie.

➤ ***Réception des paramètres dynamiques de QoS collectées par le nœud requérant***

Un serveur d'applications SAS interagit avec le service Web «*NetworkQoSService*» par le biais de la méthode publiée *updateQuotation* du nœud GLWSM requérant, afin de mettre à jour les données dynamiques de QoS collectées. À la réception du message, le nœud requérant met à jour sa cache d'informations sur la qualité de service en faisant appel à la méthode *addQoSQuotation* du gestionnaire de découverte de services *DvtServiceManager*.

➤ ***Propagation des paramètres dynamiques de QoS collectées par le nœud requérant***

La propagation des paramètres dynamiques de QoS collectés par le nœud requérant aux autres nœuds clients GLWSM offrant le même service *S* se fait en faisant appel à la méthode *createMessageSendQoS* du gestionnaire de services *NetworkQoSManager* (Voir extrait du code Figure 4.4). Cette méthode permet d'envoyer un message asynchrone à tous les nœuds clients GLWSM offrant le service *S*. Pratiquement, on enregistre dans une phase de configuration préalable, l'adresse de chaque courtier de messages associé à un nœud client.

Chaque nœud client GLWSM est continuellement à l'écoute des messages de propagation de QoS par le biais d'un port alloué au courtier de propagation (nous avons utilisé les ports 7679, 7680 pour la simulation). L'écoute des messages de propagation des paramètres de QoS d'un service spécifique *S* se fait par le biais de la méthode *onMessage* du gestionnaire de QoS réseau *NetworkQoSManager* de chaque nœud GLWSM_k offrant un accès au service publié *S*. Le message de propagation consiste en une retransmission des paramètres de QoS collectés (bande passante *BW*, facteur

d'utilisation *UF*) dans un domaine particulier d'un serveur GLWSM_i offrant le service *S*. Les données collectées ainsi que les informations permettant d'identifier le service (l'identifiant du service *serviceId*), le nœud (l'identifiant du nœud *nodeId*) et l'adresse du serveur d'application (adresse URL *urlApp*) sont par la suite propagées à tout autre serveur client GLWSM_k offrant le même service *S*. Le modèle de Souscription/Publication est utilisé pour produire le message consommé par les nœuds clients. À la réception du message, chaque nœud client met à jour sa cache d'informations sur la qualité de service en faisant appel à la méthode *addQoSQuotation* du gestionnaire de découverte de services *DvtServiceManager*.

```

public void createMessageSendQoS(String serviceId, String nodeId, String urlApp, float BW, float
UF, java.util.Date date){
    ObjectMessage message=null;
    try {
        Hashtable params=new Hashtable();
        params.put("serviceId", serviceId);
        params.put("nodeId", nodeId);
        params.put("urlApp",urlApp);
        params.put("BW",BW);
        params.put("UF",UF);
        params.put("date",date);
        message=createMessage(params);
        sendMessage(message); } catch (Exception e) {
        System.out.println("Exception occurred: " +
            e.toString());
    }
}

public void onMessage(Message msg) {
    ObjectMessage message=null;
    try{
        message=(ObjectMessage) msg;
        Hashtable params= (Hashtable) message.getObject();
        String serviceId = (String) params.get("serviceId");
        String nodeId = (String) params.get("nodeId");
        String urlApp=(String) params.get("urlApp");
        float BW= (float) params.get("BW");
        float UF= (float) params.get("UF");
        Date date= (Date) params.get("date");
        dvtServiceManager.addQoSQuotation(serviceId, nodeId, urlApp, BW, UF, date);
    }catch (JMSEException e) {
        System.out.println("Exception occurred: " +
            e.toString());
    }
}

```

Figure 4.4 Extrait de l'implémentation de NetworkQoSManager

4.3.5 Gestion de la découverte des services

Le gestionnaire de la découverte des services « DvtServiceManager » permet d'ajouter, de supprimer, de modifier ou de rechercher un service Web géolocalisé dans la base de données UDDIM (voir Annexe, Figure A.3). Pour implémenter le gestionnaire de découverte des services, nous avons modifié le module jUDDI afin d'exposer les API recherche de service le plus proche avec (*findNearestServiceQoS*) ou sans QoS (*findNearestService*), ainsi que l'API de publication d'un service *createService*. La méthode publiée *findNearestService* permet de trouver le service le plus proche qui couvre la position géographique courante d'un client mobile. Elle a deux paramètres l'identifiant du mobile *msid* et l'identifiant du service *serviceId*. Chaque client mobile, suite à une authentification à son serveur GLWSM_h de résidence, peut interagir avec la méthode *findNearestService*. Dans son exécution, *findNearestService* communique avec le gestionnaire de localisation des mobiles *LocationManager* en appelant la méthode *locateMsid* qui permet de localiser la position courante du requérant. Par la suite, le nœud GLWSM qui couvre la position actuelle du client requérant le service est trouvé par appel à la méthode *findNearestNode*. Ce nœud est considéré comme le nœud GLWSM_{nn} le plus proche. L'adresse du serveur d'applications SAS associé à GLWSM_{nn} et qui offre le service requis (*serviceId*) est alors retournée. Le nœud le plus proche GLWSM_{nn} est notifié en utilisant un message asynchrone *suiviMsid* de traquer la position du client mobile localisé dans sa zone de couverture géographique.

La méthode publiée *findNearestServiceQoS* permet de trouver le service le plus proche avec QoS qui couvre la position géographique courante d'un client mobile. Hormis les paramètres sur l'identifiant du mobile *msid* et l'identifiant du service *serviceId*, elle possède trois paramètres permettant de transmettre les requis de QoS d'un client mobile (le coût *rcost*, la bande passante *rbw*, le facteur d'utilisation du serveur *ruf*). Chaque client mobile, suite à une authentification à son serveur GLWSM_h de résidence, peut interagir avec la méthode *findNearestServiceQoS*. Dans son exécution, *findNearestServiceQoS* communique avec le gestionnaire de localisation des mobiles

LocationManager en appelant la méthode *locateMsid* qui permet de localiser la position courante du requérant. Par la suite, le nœud GLWSM qui couvre la position actuelle du client requérant le service est trouvé par appel à la méthode *findNearestNode*. Ce nœud est considéré comme le nœud $GLWSM_{nn}$ le plus proche. Après une vérification de la condition de sélection par le biais de la méthode *verifySelectionCondition*, l'adresse du serveur d'applications SAS associé à $GLWSM_{nn}$ et qui offre les requis de QoS désirés par le client est alors retournée. Le nœud le plus proche $GLWSM_{nn}$ est notifié en utilisant un message asynchrone *suiviMsidQoS* de traquer la position du client mobile localisé dans sa zone de couverture géographique.

Pour publier un service S , un fournisseur autorisé FA obtient au préalable un agrément de service AS d'un administrateur du système. Concrètement, la publication se fait en deux étapes au nœud racine puis à tous les nœuds de la feuille de l'arbre topologique sélectionnés par le fournisseur autorisé FA en propageant les informations transmises par message asynchrone.

Au nœud racine $GLWSM_h$, un fournisseur autorisé FA fait appel à la méthode publiée ou l'API *createService*. Cette méthode a pour paramètres l'objet service (l'entité d'un service *businessService*), la confirmation de l'authentification de l'administrateur *authInfo*, les paramètres de l'agrément de service AS (l'identifiant de l'agrément de service *agreementID*, l'identifiant de l'administrateur responsable de la publication du service *adminId*, le système de coordonnées *systCoord*) et une liste des agréments de service à chaque nœud (un agrément de service à un nœud contient l'identifiant d'agrément *agreementID*, l'identifiant du nœud *nodeId*, l'adresse de l'application *urlApp*, le coût du service *serviceCost*, la bande passante minimum et maximum que le service peut fournir *BW_min* et *BW_max*, le délai de propagation minimum et maximum que le service peut fournir *PPD_min* et *PPD_max*, la taille maximum des données du service *SDU_smax* et le facteur d'utilisation du serveur d'application *UF*) dans le système GLWSA. À la réception du message, une validation de la valeur *authInfo* est effectuée, puis la méthode *saveService* est appelée dans le but de sauvegarder l'entité de service *businessService* dans la base de données UDDIM du

nœud $GLWSM_h$. On fait appel par la suite à la méthode *createAllAgreementNode* dans le but de créer tous les agréments de nœuds dans le registre UDDIM du nœud $GLWSM_h$.

Les informations publiées au nœud $GLWSM_h$ sont propagées à tous les nœuds $GLWSM$ sélectionnés dans le contrat des agréments aux nœuds. Pour envoyer le message asynchrone de publication d'un service aux nœuds sélectionnés, la méthode *createMessageSendPublication* est appelée. L'écoute des messages de publication se fait par le biais de la méthode *onMessage* du gestionnaire de découverte des services *DvtServiceManager* de chaque nœud $GLWSM_k$ offrant un accès au service publié S . À la réception de ce message de publication de service, chaque nœud de destination $GLWSM_k$ extrait les paramètres envoyés et fait appel à la méthode *createServiceAndAgreementNode* de la classe *DvtServiceManager* qui crée le service, l'agrément de service et les agréments de chaque nœud impliqué. Un message d'accusé de réception est retourné par chaque nœud $GLWSM_k$ au nœud racine $GLWSM_h$ pour confirmer la publication en faisant appel à la méthode *sendReceipt* qui écrit dans la queue de publication écoutée par le nœud racine.

4.3.6 Gestion de localisation des mobiles

Le gestionnaire de localisation des mobiles *LocationManager* a une méthode publiée *locateMsid* qui permet de localiser un mobile en interagissant avec un serveur de localisation LCS (voir Figure 4.5). Cette méthode retourne la position géographique d'un identifiant de mobile *msid*. Dans son exécution, elle fait appel à la méthode *prepareRequestContext* qui permet d'initialiser le contexte de localisation *requestContext* en précisant l'adresse du serveur LCS et de créer un objet de connexion *conn* pour interagir avec le serveur LCS. Puis, la localisation est effectivement obtenue en appelant la méthode *getLocation*. Les coordonnées sous le format chaîne de caractères sont transformées en format de type réel en appelant la *getSpatialData*. Le gestionnaire de localisation des mobiles a été développé en utilisant le module MPS6.0 de Ericsson qui offre des API permettant de localiser les mobiles. Le module MPS6.0 est compatible avec le protocole MLP 3.0.

```

public Spatial locateMsid(String msid) {
    Spatial spatial=null;
    try{
        prepareRequestContext();
        MobileStation ms=new MobileStation(msid);
        LocationTarget[] target=new MobileStation[]{new MobileStation(msid)};
        LocationResult result=conn.getLocation(target,requestContext);
        LocationData location =result.getLocationData(ms);
        if(location.getLocationStatus() ==LocationData.SUCCESSFUL_LOCATION){
            LocatedMobileStation lms =(LocatedMobileStation)location;
            spatial = getSpatialData(lms);
        }
        else{
            LocationFailure failure =(LocationFailure)location;
            printLocationFailure(failure);
        }
    }catch(Exception lex) {
        System.out.println(lex.getMessage());
    } finally{
        try{
            conn.close();
        }catch(Exception ex){
            ex.printStackTrace();
        }
    }
    return spatial;
}

```

Figure 4.5 Extrait de l'implémentation de LocationManager

4.3.7 Gestion de la coordination d'une migration de services

Le gestionnaire coordonnant la migration de services *MigrationManager* offre principalement une API *onMessage* qui est une méthode asynchrone permettant d'écouter le message de suivi indiquant l'entrée d'un client mobile dans les zones de localisation couvertes par un GLWSM_j (voir Annexe, Figure A.4). Les messages de suivi des mobiles sont envoyés soit par le serveur de résidence d'un client mobile GLWSM_h lors d'une recherche de service, soit par le précédent nœud GLWSM_i où le client mobile était précédemment localisé lors d'une coordination de migration de services. Ces messages sont destinés au nœud GLWSM_j couvrant la position courante d'un client mobile. Les méthodes utilisées pour envoyer un message de suivi sont *trackMsidQoS* ou *trackMsid* pour le suivi d'un mobile avec ou sans QoS respectivement. Le nœud

GLWSM_j est considéré comme le nœud courant le plus proche de la position du client mobile et est associé à un serveur d'applications SAS_{iq} où le service requis par le client est en cours d'exécution (SAS_{iq} est considéré comme le serveur d'applications courant).

À la réception du message, les paramètres envoyés dans le cas d'un service avec QoS (l'identifiant du mobile *msId*, l'identifiant du service *serviceId*, l'adresse du serveur d'application *urlApp*, le coût du service *serviceCost*, la bande passante *BW* et le facteur d'utilisation du serveur d'application *UF*) ou dans le cas d'un service sans QoS (l'identifiant du mobile *msId*, l'identifiant du service *serviceId*) sont récupérés en vérifiant la taille des paramètres de la variable *params* de la méthode *onMessage*. L'identifiant du mobile *msid* est ajouté à la liste des mobiles *listeMsid* et la position du mobile est par la suite traquée en faisant appel à la méthode *ifExit* qui interagit avec le serveur de position LCS dans le but de déterminer la position du client mobile et de vérifier si la position obtenue est couverte par les zones de localisation du GLWSM_j courant. Dans le cas où le client sort des zones couvertes par GLWSM_j, le prochain nœud GLWSM_k qui couvre sa position actuelle et qui possède un accès au service en cours d'exécution est recherché et sélectionné de la base de données UDDIM_j par la méthode *ifExit* et sa valeur est transférée dans la variable *nextNode*. Le nœud GLWSM_k est considéré comme le prochain nœud le plus proche de la position courante du client mobile.

Par la suite, l'adresse *agreementNextNode.getUrlApp* du serveur d'applications SAS_{kp} associé au nœud GLWSM_k (qui vérifie la condition de migration de service dans le cas d'un service avec QoS) est alors sélectionnée. Le serveur d'applications SAS_{iq} en cours d'exécution est alors notifié par le biais de la méthode *notifyExit* de migrer le service en cours au serveur d'applications SAS_{kp}. À la fin de la migration de service, le nœud GLWSM_j envoie un message de suivi du mobile en écrivant dans la queue de suivi du nœud GLWSM_k et l'identifiant du mobile *msid* est enlevé de la liste des mobiles en cours de migration *listeMsid* en utilisant la méthode *removeMsid*.

4.3.8 Gestion d'abonnement

Le gestionnaire d'abonnement *SubscriptionManager* permet de créer, de rechercher et de supprimer un abonnement et un usager. Les différentes API offerts par *SubscriptionManager* sont : *createSubscription*, *findSubscription*, *removeSubscription*, *createUser*, *findUser* et *removeUser*. Le détail de ces méthodes étant trivial, leur contenu n'est pas donné ici.

4.4 La couche des classes de base

La couche des classes de base permet d'interagir avec les bases de données (UDDIM et "ClientInfosDB") et de définir les structures de données de base utilisées par le système GLWSA. La description de chacune de ces classes de base est donnée au Tableau 4.3 et leur illustration est représentée par le diagramme de classes de la Figure 4.6.

Tableau 4.3 Classes de base de l'architecture fonctionnelle de GLWSA

Nom de la classe	Description	Base de données
Address	Définit la structure d'une adresse d'un usager.	"ClientInfosDB"
Agreement	Définit la structure d'un agrément de service	UDDIM
AgreementNode	Définit la structure d'un agrément de service à un nœud spécifique.	UDDIM
BusinessService	Définit la structure d'un service dans la base de données UDDIM.	UDDIM
MigrationNotification	Définit une classe permettant de notifier un service Web d'un fournisseur d'applications de l'entrée ou de la sortie d'un mobile. Les méthodes <i>migrateTo</i> et <i>pushAnnounceEntry</i> doivent être modifiées par le fournisseur de services pour donner un comportement spécifique.	
MobileEquipment	Définit la structure d'un équipement mobile.	"ClientInfosDB"
Node	Définit la structure d'un nœud GLWSM.	UDDIM
QoSQuotation	Définit la structure d'une soumission de la qualité de service d'un service spécifique à un nœud GLWSM donné.	
QoSRequest	Définit la structure d'une requête de QoS d'un client donné.	
Role	Définit la structure du rôle d'un usager.	"ClientInfosDB"
Service	Définit la structure d'un objet de service. Il est utilisé pour échanger les informations entre les objets du système.	
ServiceCache	Définit une cache d'informations des soumissions de QoS obtenues.	
Spatial	Définit la structure d'une position géographique en coordonnées cartésiennes.	

Subscription	Définit la structure d'un abonnement à un service donné.	"ClientInfosDB"
Theme	Définit la structure d'un thème.	"ClientInfosDB"
ThemeMsid	Définit la structure d'un abonné à un thème.	"ClientInfosDB"
User	Définit la structure d'un usager dans le système.	"ClientInfosDB"
Zone	Définit la structure d'un modèle de zone (rectangulaire).	

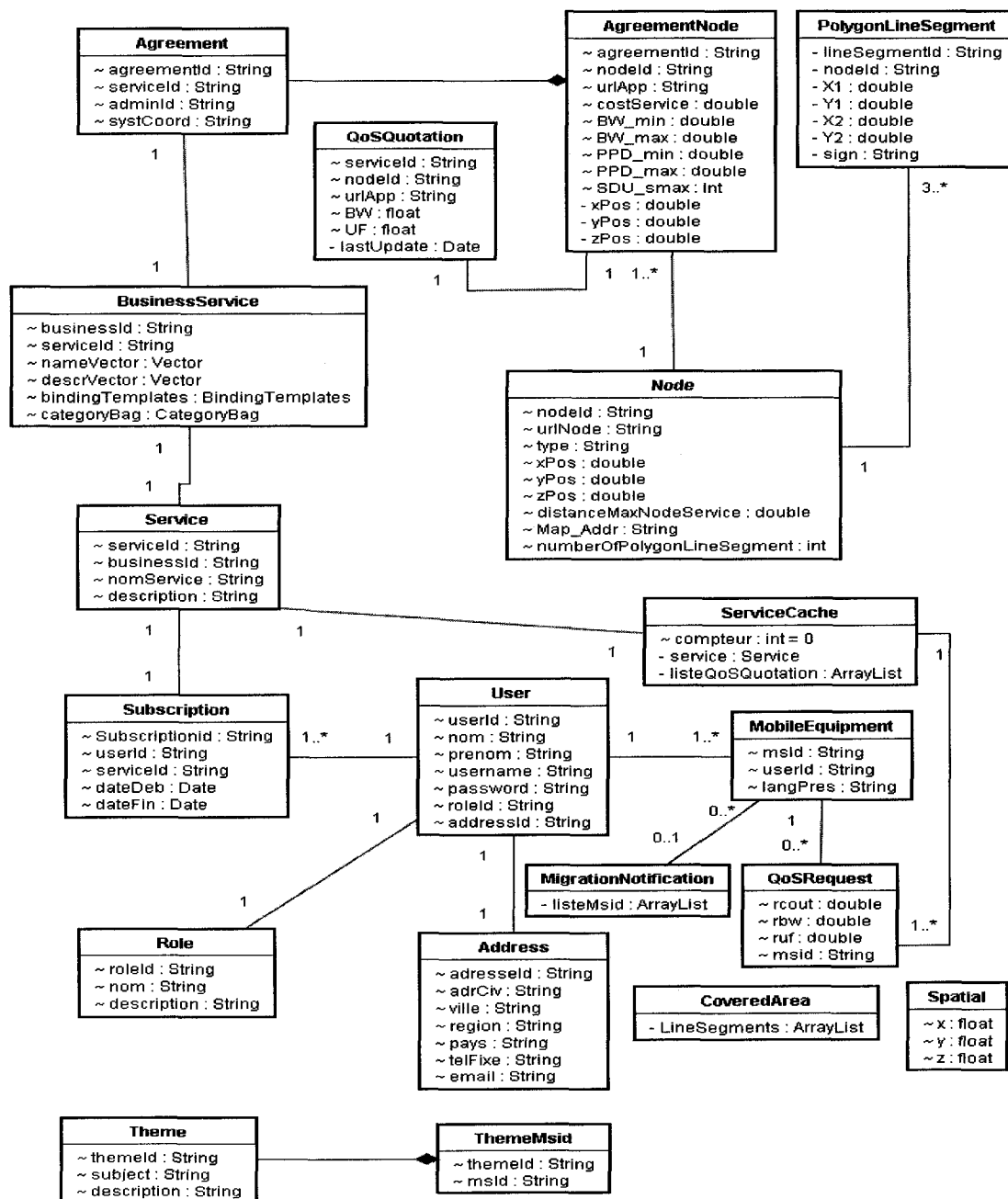


Figure 4.6 Diagramme de classes des classes de base de l'architecture fonctionnelle

Les objets matérialisant les classes de base dans le système GLWSA sont soit persistants soit volatiles. Les objets volatiles sont des objets qui ont une durée de vie liée à l'exécution de l'application. Par contre, les objets persistants ont une durée de vie prolongée et nécessite un système de gestion de données pour les manipuler.

4.5 La persistance dans le système GLWSA

Dans le système GLWSA, on distingue deux types de persistance : la persistance locale et la persistance globale (Figure 4.7). La persistance locale consiste à gérer l'état d'un objet sauvegardé dans un serveur GLWSM local sans interaction avec les autres serveurs GLWSM. Les objets des classes de base qui utilisent une persistance locale sont les représentants des classes *Subscription*, *User*, *Address*, *Role* et *MobileEquipment*. La consistance des objets dans une persistance locale est prise en charge par le SGBD utilisé (dans notre cas Oracle 9i).

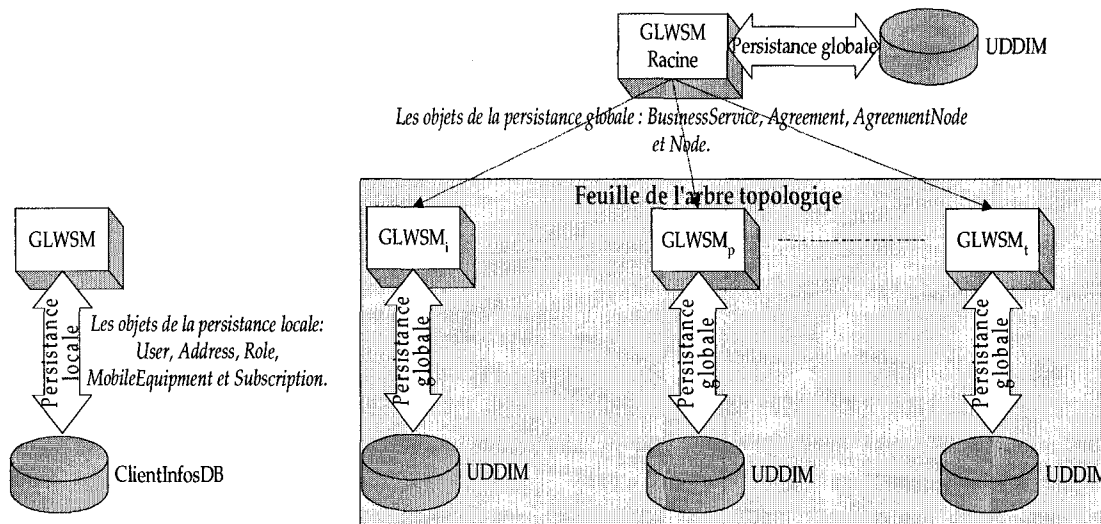


Figure 4.7 Persistance des données dans le système GLWSA

Quant à la persistance globale, elle consiste à gérer globalement l'état d'un objet persistant dans plusieurs serveurs GLWSM. Elle nécessite la mise en place d'un mécanisme de réplication permettant de préserver la consistance de l'objet dans les

SGBD locaux qui manipulent l'état de l'objet. C'est le cas des objets transmis lors d'une publication de service *BusinessService* et *AgreementNode* ou des objets créés lors de l'établissement d'un agrément de service *Agreement* et *Node*. Le modèle de réplication utilisé pour réaliser la persistance globale est un modèle Maître/Esclaves. Le serveur GLWSM racine joue le rôle de maître et coordonne la transmission de l'état d'un objet persistant globalement à tous les serveurs GLWSM de la feuille de l'arbre topologique impliqués dans la mise jour de l'objet. Les serveurs GLWSM de la feuille de l'arbre topologique jouent le rôle d'esclave. Ainsi, lors de la publication d'un service, le contrôle de la consistance des données persistantes dans le système GLWSA est effectué par la méthode *createService* du gestionnaire de découverte de services *DvtServiceManager*, tandis que lors de la création d'un agrément de service, le contrôle de la consistance des données persistantes dans le système GLWSA est effectué par la méthode *createAgreementAndNode* du gestionnaire d'agrément de service *AgreementManager*.

La connexion aux bases de données est réalisée en utilisant un connecteur JDBC pour chacune des bases de données (UDDIM et "ClientInfosDB"). Ces connecteurs sont statiques dans le développement du système afin de partager leur instance unique entre différents objets voulant accéder à une base de données spécifique.

4.6 Extension du protocole MLP : MLPe

Le protocole MLPe étend le protocole MLP afin de permettre une localisation thématique des mobiles. Cette extension nous a conduit à ajouter deux structures de données (les classes *Theme* et *ThemeMsid*), des API pour gérer la structure proposée et des méthodes permettant la localisation thématique des mobiles.

4.6.1 Structures de données proposées

Deux classes ont été ajoutées au protocole MLP afin de permettre une localisation thématique des mobiles. Il s'agit des classes *Theme* et *ThemeMsid* dont les structures de données et leurs relations sont matérialisées à la Figure 3.15. La classe

Theme permet d'ajouter un thème spécifique. La classe *ThemeMsid* permet de lier un mobile identifié *msid* à un thème identifié *themeId*. La gestion et la persistance des objets de type *Theme* et *ThemeMsid* est effectué par un serveur LCS associé à une base de données (par exemple, dans le cadre de cette implémentation nous avons utilisé la base de données *ClientInfosDB*).

4.6.2 Gestion des structures Theme et ThemeMsid

Le protocole MLPe fournit des API permettant de gérer les classes « Theme » et « ThemeMsid ». Pour la classe « Theme », on distingue la méthode *addTheme* qui a comme paramètres d'entrées tous les attributs de « Theme » et qui permet de créer un objet de type « Theme »; la méthode *removeTheme*(*themeId*: String) et la méthode *findTheme*(*themeId*: String) qui permettent respectivement de supprimer et de rechercher un thème. En ce qui concerne la classe « ThemeMsid », on distingue comme APIs la méthode *addThemeMsid* qui a comme paramètres d'entrées tous les attributs de « ThemeMsid » et qui permet de créer un objet de type « ThemeMsid »; la méthode *removeThemeMsid* (*themeId*: String, *msid*: String) permettent de supprimer un objet « ThemeMsid » et la méthode *findAllMsidOfTheme*(*themeId*: String) permet de rechercher tous les mobiles liés à un thème donné.

4.6.3 Méthodes de localisation thématique

Pour ajouter les fonctionnalités ou méthodes proposées (voir Annexe, Figure A.5), nous avons modifié la classe *com.ericsson.mps.jml.DefaultConnection* du module MPS 6.0 de Ericsson qui offre une implémentation du protocole MLP version 3.0.

Pratiquement, nous avons ajouté les méthodes *ifEntry* ou *ifExit* qui permettent de vérifier l'entrée et la sortie des clients mobiles liés par un sujet (ou un thème) d'une zone de localisation et d'en informer un serveur d'applications SAS en utilisant respectivement une notification d'entrée *notifyEntry* ou une notification de sortie *notifyExit*. En effet, la méthode *ifExit* (ou *ifEntry*), dans son exécution, fait appel à la méthode *locateSubject* qui localise tous les mobiles liés au sujet *subject* donné comme

paramètre d'entrée de la méthode. La position de chaque mobile trouvé est par la suite comparée à la zone de référence *zone* donnée comme paramètre d'entrée à la fonction appelée *ifExit* (ou *ifEntry*) pour déterminer si le mobile n'est pas (ou est) dans la zone de référence.

Les méthodes de localisation thématique (*locateSubject*, *locateSubjectInArea*, *locateSubjectNearOf*, *collocate*) permettant de déterminer la position d'un groupe de mobiles liés par un sujet y ont été aussi ajoutées. La méthode *locateSubject* permet de déterminer la position d'un groupe de mobiles liés par un sujet. Elle détermine au préalable tous les identifiants de mobile liés au sujet *subject*, puis obtient la localisation de chacun des mobiles en faisant appel à la méthode *getLocation*. La méthode *locateSubjectInArea* permet de trouver les mobiles d'un sujet *subject* se trouvant dans une zone de localisation donnée *zone*. La méthode *locateSubjectNearOf* permet de trouver les mobiles d'un sujet *subject* dont la distance par rapport à un point de référence *spatial* est inférieure à une précision donnée *precision*. La méthode *collocate* permet de trouver les mobiles d'un sujet *subject* dont la distance par rapport à un client mobile de référence *msid* est inférieure à une précision donnée *precision*. Pour des contraintes de programmation liées au fait que la communication du serveur LCS de MPS 6.0 exige un paramètre matérialisant le contexte de la requête *requestContext*, nous l'avons ajouté comme paramètre d'entrée à toutes les méthodes de localisation thématique que nous avons présentées dans cette section.

4.6.4 Modèle de sécurité

Un modèle de sécurité est nécessaire à la localisation thématique proposée pour protéger les informations relatives à la position géographique et l'identité des clients mobiles. Ces informations sont échangées entre le système étudié GLWSA et les applications des fournisseurs de services (SAS). La politique d'accès à ces données suit principalement deux règles :

1. Un sujet (personne autorisée à utiliser une application d'un SAS) a le droit d'utiliser les fonctions de localisation thématique afin de localiser un ou

plusieurs clients mobiles si et seulement si ces clients sont abonnés à cette application.

2. Un sujet (personne autorisée à utiliser une application d'un SAS) a le droit d'utiliser les fonctions d'identification d'un client si et seulement si le client est abonné à cette application.

Le contrôle d'accès à ces données ne peut être fait par le modèle classique MAC (*Mandatory Access Control*) tel que Bell-LaPadula car les sujets (personnes autorisées à utiliser une application d'un SAS) et les objets (les clients mobiles) ne peuvent être réduits à des compartiments classifiés et hiérarchisés en niveaux de sécurité. En effet, si les objets sont classés par exemple en deux compartiments P_1 , P_2 et que l'intersection de P_1 et P_2 soit vide tel que suggère les règles de MAC, ceci signifierait que toute application a le groupe P_1 ou P_2 pour abonnés. En pratique, cela n'est pas vrai, car les abonnés des applications sont rarement tous identiques.

Le contrôle d'accès discrétionnaire (DAC) n'est pas non plus adapté à la politique que nous souhaitons appliquer, car elle ne permet pas d'octroyer l'accès sous certaines conditions. En effet, le modèle DAC est de la forme:

<sujet> <opérations permises> <objet cible>

Or, dans la politique de sécurité à mettre en place le contrôle d'accès obéit à la logique suivante:

<sujet> <opérations permises> <objet cible> <condition sur l'objet cible>

Exemple : <Personne autorisée à utiliser l'application de suivi des médecins ambulants ABC><localiser un client abonné> <le client *msid* > <si le client recherché *msid* est dans le groupe de clients de cette application dont il a droit>.

Le modèle d'accès par capacité (*capability*) est indiqué pour la politique souhaitée. En effet, pour l'allocation des opérations, il faudra créer au préalable des groupes de clients. Avec la possibilité qu'un client mobile puisse appartenir à plusieurs groupes. Pour chaque client donné C , un ou plusieurs groupes lui seront alloués, ce qui

signifie en clair signifie que le client C n'a le droit d'interroger le serveur LCS rien que sur la position des mobiles appartenant aux groupes qui lui ont été alloués. Ainsi, lors d'un contrôle d'accès, l'identité du requérant sera déterminée, puis l'on vérifiera si l'opération appelée lui est allouée et si le client cible CC dont la position est recherchée est dans le groupe des clients qui lui est autorisé pour cette application.

CHAPITRE V

ÉVALUATION DE L'ARCHITECTURE ET RÉSULTATS

Le système GLWSA a été évalué afin d'apprécier les résultats obtenus. Nous avons évalué les requis fonctionnels (recherche de service, publication de service, coordination d'une migration de service, localisation thématique). La collecte des données dynamiques de QoS a été également simulée afin d'évaluer le concept proposé. Ce chapitre présente l'évaluation des requis fonctionnels et l'évaluation de la collecte des données dynamiques de QoS.

5.1 Évaluation des requis fonctionnels

Pour évaluer les requis fonctionnels (rechercher un service Web géolocalisé le plus proche sans QoS, recherche de service Web géolocalisé avec QoS, publier un service Web géolocalisé, coordonner la migration d'un service Web géolocalisé sans QoS, coordonner la migration d'un service Web géolocalisé avec QoS), nous avons utilisé comme indice de performance la métrique temps ou délai. Le délai RTT (*Round Trip Time*) mesuré correspond à un aller-retour soit la différence de temps entre la réception de la requête par le client et l'envoi de la requête par le client. Dans l'organisation de cette section, nous présentons d'abord les paramètres de configuration de l'environnement de test, puis nous présentons et analysons les résultats obtenus.

5.1.1 Paramètres de configuration de l'environnement de test

Afin d'évaluer les requis fonctionnels du système GLWSA, nous avons réalisé un prototype de l'architecture, en utilisant le langage de programmation Java sous un environnement intégré JBuilder 7 version entreprise. Le prototype implémente les fonctionnalités de l'architecture fonctionnelle décrite au chapitre 4. Pour cela, nous avons utilisé cinq machines ayant une configuration semblable. Les machines sont

connectées à un IEEE 802.11b WLAN (*Wireless Local Access Network*). Le WLAN a une capacité de transmission de 11 Mbits/s. Les caractéristiques de chacune de ces machines sont les suivantes :

- fréquence Pentium 3 à 1.2 GHz,
- RAM de 512 Mo,
- JRE (*Java Runtime Environment*) version 1.4,
- Windows 2000 professionnel.

La configuration des différents scénarios de tests est donnée au Tableau 5.1. La communication avec le serveur de position LCS a été réalisée en utilisant l'émulateur MPS 6.0 de ERICSSON. Cet émulateur implémente le protocole MLP et simule le calcul de la position d'un client ou d'un groupe de clients se déplaçant sur le réseau 3G d'un opérateur.

Tableau 5.1 Rôle des machines dans l'environnement de test

Fonctionnalité testée	Machine 1	Machine 2	Machine 3	Machine 4	Machine 5
Rechercher un service Web géolocalisé sans QoS	GLWSM _h de résidence	GLWSM _c où se trouve présentement le client mobile.	LCS		
Rechercher un service Web géolocalisé avec QoS	GLWSM _h de résidence	GLWSM _c où se trouve présentement le client mobile.	LCS		
Publier un service Web géolocalisé	GLWSM _r de résidence	GLWSM _{fi} un nœud feuille où le service est publié.	GLWSM _{f2} un nœud feuille où le service est publié.	SAS	
Coordonner la migration d'un service Web géolocalisé sans QoS	GLWSM _c est le nœud courant couvrant la position d'un mobile.	GLWSM _s est le nœud suivant couvrant la position d'un mobile.	LCS	SAS _c est le serveur d'applications courant.	SAS _s est le serveur d'applications suivant.
Coordonner la migration d'un service Web géolocalisé avec QoS	GLWSM _c est le nœud courant couvrant la position actuelle d'un mobile.	GLWSM _s est le nœud suivant couvrant la position actuelle d'un mobile.	LCS	SAS _c est le serveur d'applications courant.	SAS _s est le serveur d'applications suivant.

Pour effectuer les tests, nous avons généré une topologie réseau ayant les caractéristiques suivantes (Figure 5.1): le système de coordonnées par défaut est le système géographique (*latitude, longitude, altitude*), la forme des cellules est polygonale, la densité du trafic est initialisée à urbain (*urban*), la distance entre deux stations de base a pour valeur 5000 mètres. Nous avons également défini la trajectoire que doit emprunter le mobile cible à une vitesse constante de déplacement. Nous avons fait varier la vitesse en affectant les valeurs 50 km/h, 100km/h et 200 km/h. La densité du trafic peut prendre l'une des valeurs (urbaine dense, urbaine ou rurale). Par défaut l'altitude a une valeur égale à 0. À la Figure 5.1, le domaine $GLWSM_h$ représente le nœud de résidence GLWSM du client mobile cible et la région géographique qu'il couvre, les domaines $GLWSM_1$ et $GLWSM_2$ représentent des nœuds GLWSM visités et leurs régions géographiques couvertes associées.

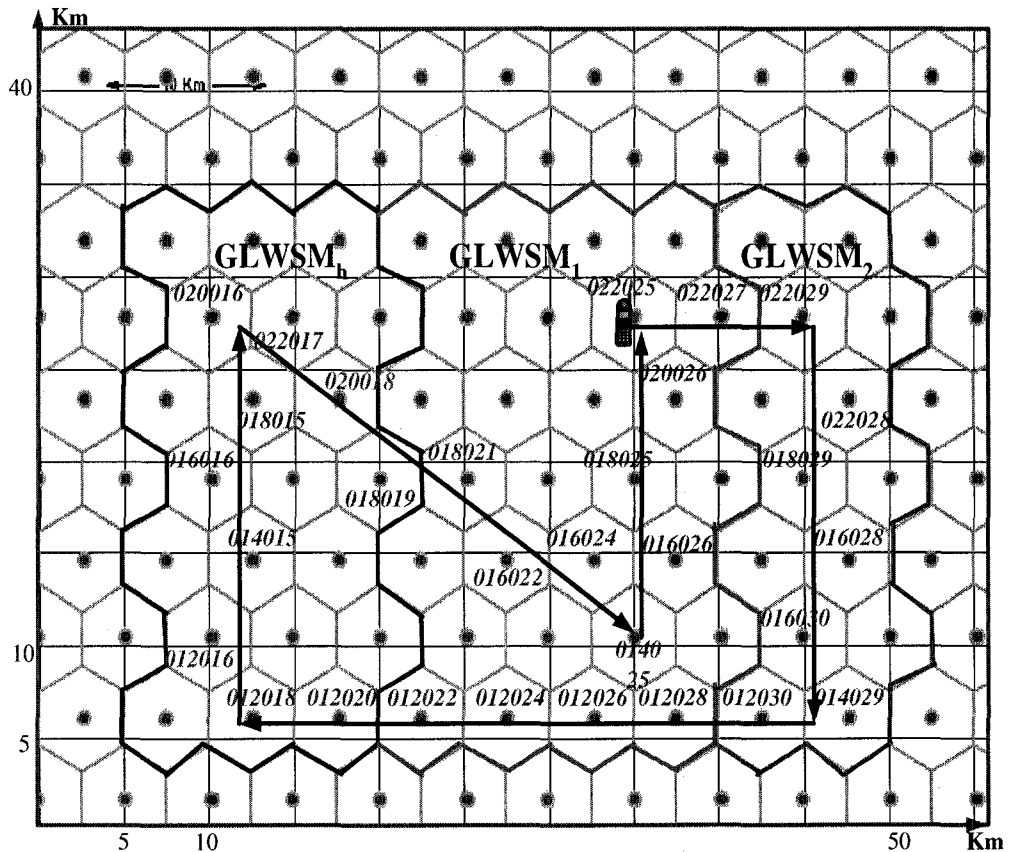


Figure 5.1 Topologie générée et trajectoire d'un client cible

En utilisant cette configuration, l'émulateur génère deux fichiers : le premier indique comment sont disposées les cellules de la topologie (Tableau 5.2) et le second contient les informations sur le trajet emprunté par les mobiles (Tableau 5.3).

Tableau 5.2 Extrait du fichier de la topologie des cellules

Identifiant de la cellule	Longitude	Latitude	Type d'antenne	Distance entre deux stations de base (en mètres)	Angle de couverture de la station de base
000002	W01207	S01050	Omni	5000	360
000004	W00925	S01050	Omni	5000	360
000006	W00644	S01050	Omni	5000	360
000008	W00402	S01050	Omni	5000	360
000010	W00121	S01050	Omni	5000	360
000012	E00121	S01051	Omni	5000	360
000014	E00402	S01051	Omni	5000	360
000016	E00644	S01051	Omni	5000	360
000018	E00925	S01051	Omni	5000	360
000020	E01207	S01051	Omni	5000	360
000022	E01449	S01051	Omni	5000	360
000024	E01730	S01051	Omni	5000	360
000026	E02012	S01051	Omni	5000	360
000028	E02253	S01051	Omni	5000	360
000030	E02535	S01051	Omni	5000	360
000032	E02816	S01051	Omni	5000	360
000034	E03058	S01051	Omni	5000	360
000036	E03340	S01051	Omni	5000	360
000038	E03621	S01051	Omni	5000	360
000040	E03903	S01051	Omni	5000	360

Le fichier généré sur la topologie des cellules contient 459 enregistrements. Les champs de ce fichier sont l'identifiant de la cellule, la position du centre de la cellule exprimée en coordonnées spatiales (longitude, latitude) dans le système UTM (*Universal Transverse Mercator*) [93], le type d'antenne indique si l'antenne diffuse autour de lui (*Omni*) ou dans une portion de son champ d'action (*CircleSector*), la distance entre la station de base courante et une station de base adjacente, l'angle de couverture de la station de base (*360° pour une antenne Omni*). Une station de base d'une cellule est placée au centre de la cellule. Le système UTM divise le globe terrestre en 60 lignes

verticales espacées de 6° et 20 lignes horizontales espacées de 8°. Une valeur de la longitude telle que W00121 signifie que le point en question est à l'ouest du globe terrestre (signe W) et que l'angle est de 0° 01' 21'' par rapport à l'origine (l'horizontale *Équateur*, la verticale *Greenwich*). De même pour la latitude, une valeur de S01050 signifie que le point en question est à l'hémisphère sud (signe S) et que son angle est de 0° 10' 50'' par rapport à l'origine.

Le fichier qui indique la route empruntée par un mobile est généré pour une vitesse constante donnée. Dans l'extrait du fichier de la route représenté au Tableau 5.3, le mobile a une vitesse de 10 Km/h.

Tableau 5.3 Extrait du fichier du trajet emprunté par les mobiles

Identifiant du mobile	Instant de prélèvement	Identifiant de la cellule	Distance par rapport à la station de base courante	Longitude	Latitude
46708123456789	0	022025	476	E01848	N01444
46708123456789	10	022025	510	E01857	N01444
46708123456789	20	022025	670	E01906	N01444
46708123456789	30	022025	890	E01915	N01444
46708123456789	40	022025	1135	E01924	N01444
46708123456789	50	022025	1393	E01933	N01444
46708123456789	60	022025	1657	E01942	N01444
46708123456789	70	022025	1925	E01951	N01444
46708123456789	80	022025	2195	E02000	N01444
46708123456789	90	022025	2467	E02009	N01444
46708123456789	100	022027	2347	E02018	N01444
46708123456789	110	022027	2076	E02027	N01444
46708123456789	120	022027	1806	E02036	N01444
46708123456789	130	022027	1540	E02045	N01444
46708123456789	140	022027	1278	E02054	N01444
46708123456789	150	022027	1025	E02103	N01444
46708123456789	160	022027	788	E02112	N01444
46708123456789	170	022027	589	E02121	N01444
46708123456789	180	022027	476	E02130	N01444
46708123456789	190	022027	510	E02139	N01444
46708123456789	200	022027	670	E02148	N01444
46708123456789	210	022027	890	E02157	N01444
46708123456789	220	022027	1135	E02206	N01444
46708123456789	230	022027	1393	E02215	N01444
46708123456789	240	022027	1657	E02224	N01444
46708123456789	250	022027	1925	E02233	N01444
46708123456789	260	022027	2195	E02242	N01444

Le fichier généré contient les champs d'informations suivants : l'identifiant du mobile qui se déplace dans la topologie, les instants de prélèvements de la position qui servent à intervalle régulier (la période est de 10 secondes), l'identifiant de la cellule où se trouve le mobile, la distance du mobile par rapport à la station de base de la cellule courante, la position du mobile en coordonnées spatiales (longitude, latitude) du système UTM. L'émulateur utilise ces données pour effectuer une interpolation linéaire entre deux mesures successives afin de trouver la position du mobile en un instant compris entre deux temps successifs enregistrés dans le fichier. La formule utilisée pour effectuer l'interpolation n'est pas donnée dans les spécifications de l'outil.

Pour la persistance de données, le système de gestion de base de données Oracle 9i a été utilisé pour les bases de données UDDIM et « ClientInfosDB ». Les mesures de la performance des requis fonctionnels a été faite en utilisant l'outil *SoapTest* [80] qui permet d'interroger un serveur Web service sous différents scénarios.

5.1.2 Résultats et analyse des performances

Les résultats ont été captés en utilisant l'outil *SoapTest* en interrogeant le serveur GLWSM sous différents scénarios. Nous avons effectué des mesures sur les fonctionnalités suivantes :

- rechercher un service Web géolocalisé sans QoS ;
- rechercher un service Web géolocalisé avec QoS ;
- publier un service Web géolocalisé ;
- coordonner la migration d'un service Web géolocalisé sans QoS ;
- coordonner la migration d'un service Web géolocalisé avec QoS.

➤ *Rechercher un service Web géolocalisé sans QoS*

Le principe de la recherche d'un service Web géolocalisé consiste à envoyer auprès du serveur GLWSM de résidence, les paramètres de la méthode publiée, soient

l'identifiant du service et l'identifiant du mobile. Pour mesurer le délai RTT, nous avons effectué différents scénarios.

Scénario 1: un client envoie une requête toutes les deux secondes

Ce scénario nous permet de mesurer le temps moyen de recherche d'un service dans le cas où le serveur n'a qu'un client connecté. Les résultats obtenus sont représentés à la Figure 5.2. Nous avons simulé un client qui envoie toutes les deux secondes une requête de recherche de service Web géolocalisé sans QoS pendant dix minutes. Nous trouvons un délai moyen RTT de recherche de service de 28 millisecondes, avec un minimum de 20 millisecondes et un maximum de 52 millisecondes. Les mesures effectuées ont une dispersion ou un écart-type de 14.16 millisecondes. Nous avons constaté que les fluctuations des mesures RTT varient dans le même sens que le temps de localisation d'un mobile au LCS. Le temps de localisation d'un mobile est en moyenne égale 23 millisecondes et représente environ 82% du temps du temps RTT.

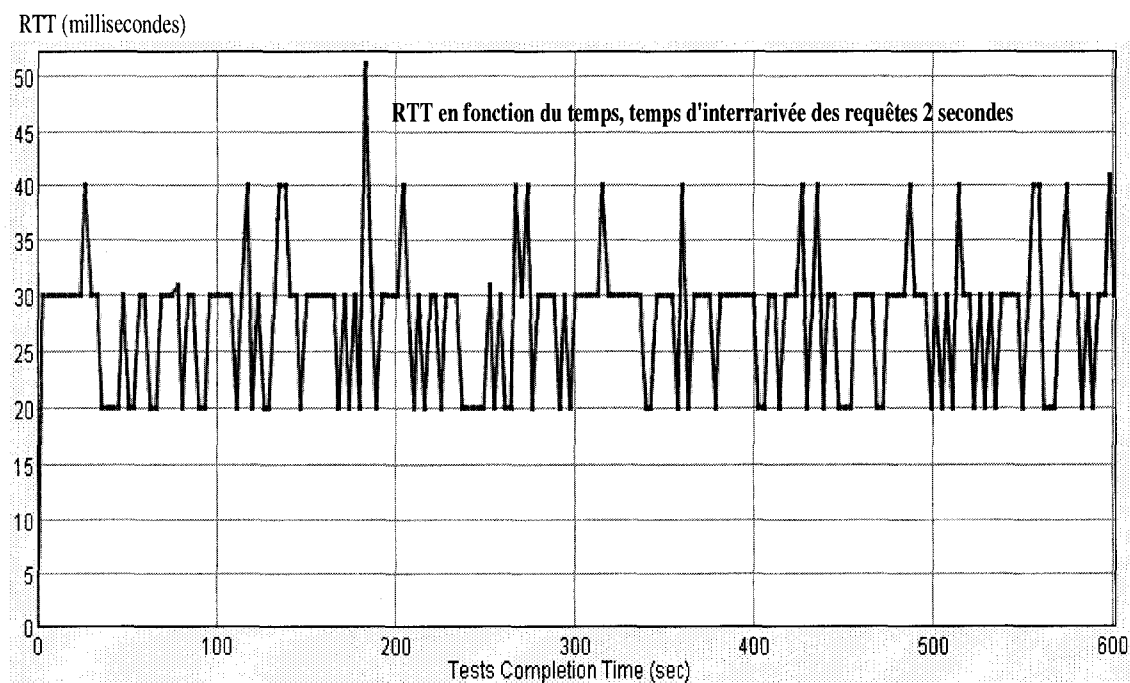


Figure 5.2 Recherche d'un service Web géolocalisé sans QoS

Pour mesurer l'influence du "buffer" du serveur GLWSM sur les performances RTT de la recherche de service Web géolocalisé, nous avons effectué le scénario décrit ci-dessous.

Scénario 2: croissance linéaire du nombre de clients effectuant une requête de 1 à 50

Ce scénario nous permet de mesurer l'influence du nombre d'utilisateurs en communication avec le serveur sur les performances RTT de la recherche d'un service Web géolocalisé sans QoS. Les requêtes, comme précédemment sont envoyées toutes les deux secondes par chaque client. Les résultats obtenus sont présentés par la Figure 5.3. Nous obtenons un RTT moyen 51 millisecondes avec un maximum de 620 millisecondes et un minimum de 30 millisecondes. Le serveur GLWSM est stable (c'est à dire que les résultats obtenus sont à l'intérieur des mesures obtenues précédemment en utilisant l'écart-type de 14.16 millisecondes comme incertitude absolue par rapport à la moyenne précédente) quand le nombre de clients en communication est inférieur ou égale à 28. Dans le cas où le nombre de clients est supérieur à 28, le serveur est instable car le "buffer" est généralement plein et génère des résultats peu fiables. Il serait conseillé lors de la configuration du serveur de déterminer ce nombre limite d'utilisateurs pour appliquer une politique d'accès aux ressources du serveur.

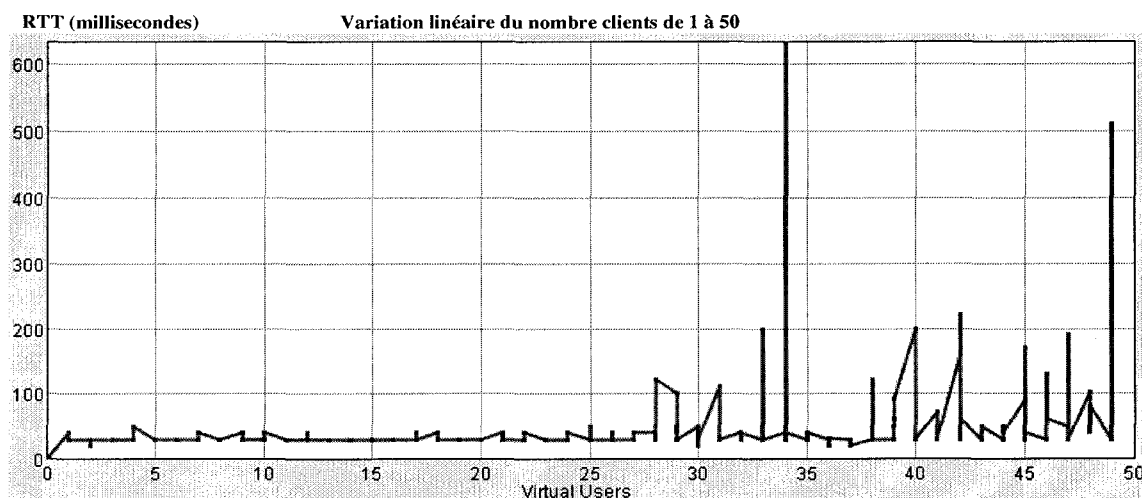


Figure 5.3 Variation linéaire du nombre de clients lors d'une recherche d'un service Web géolocalisé sans QoS

Nous avons aussi comparé les résultats obtenus à la recherche d'un service avec les protocoles SLP et Jini [38]. Pour effectuer la comparaison, nous avons ajouté aux résultats de la recherche de service de Jini et SLP, le temps de localisation moyen d'un mobile, soit 23 millisecondes. Les résultats obtenus sont représentés à la Figure 5.4. Ces résultats ne signifient pas que le système GLWSA est meilleur que SLP ou Jini, mais plutôt que le système a des meilleures performances parce que la localisation des mobiles et la structure de données UDDIM ne sont pas intégrées dans ces deux protocoles.

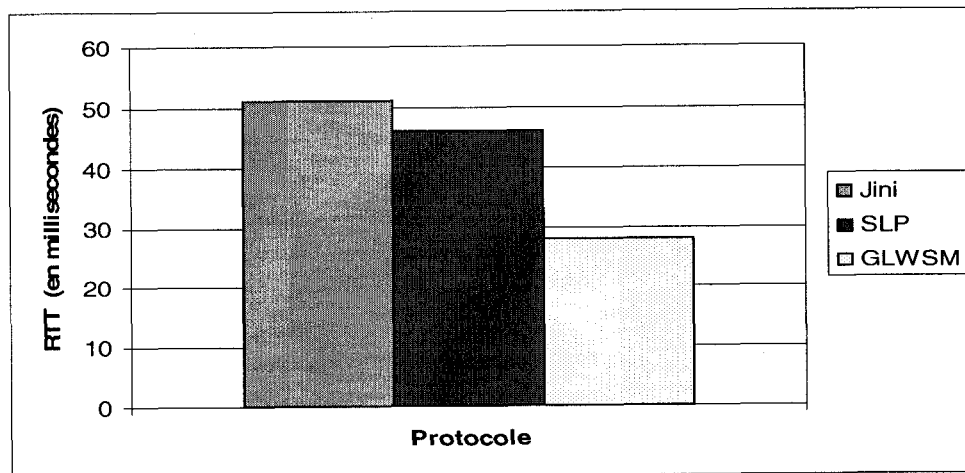


Figure 5.4 Comparaison GLWSM, SLP et Jini lors d'une recherche d'un service Web géolocalisé sans QoS

➤ ***Rechercher un service Web géolocalisé avec QoS***

Le principe de la recherche d'un service Web géolocalisé consiste à envoyer auprès du serveur GLWSM de résidence, les paramètres de la méthode publiée soient l'identifiant du service, l'identifiant du mobile, le coût du service requis, la bande passante requise et le facteur d'utilisation du serveur SAS requis. Pour mesurer le délai RTT, nous avons effectué les scénarios ci-après.

Scénario 1: un client envoie une requête toutes les deux secondes

Ce scénario nous permet de mesurer le temps moyen de recherche d'un service avec QoS dans le cas où le serveur n'a qu'un client connecté. Les résultats obtenus sont représentés à la Figure 5.5. Ces résultats matérialisent un client qui envoie toutes les deux secondes une requête de recherche de service Web géolocalisé avec QoS et pendant dix minutes. Nous obtenons une moyenne RTT de 41 millisecondes, un minimum RTT de 30 millisecondes et un maximum de 70 millisecondes. La dispersion ou l'écart-type des mesures est de 15.57 millisecondes par rapport à la moyenne. La portion de temps de la localisation d'un client sur le résultat RTT moyen obtenu est environ de 56.1%.

En comparant les résultats obtenus à la recherche de service Web géolocalisé sans QoS, on constate que la moyenne s'est déplacée d'une valeur de 13 millisecondes. Ce déplacement de la moyenne est surtout lié à un surcoût des lignes de code ajoutées dans le système pour traiter la qualité de service et un effet minime peut être attribué à l'augmentation de la taille des messages transmis (ajout de 3 paramètres de type double par rapport à une recherche sans QoS, soit 24 octets au total).

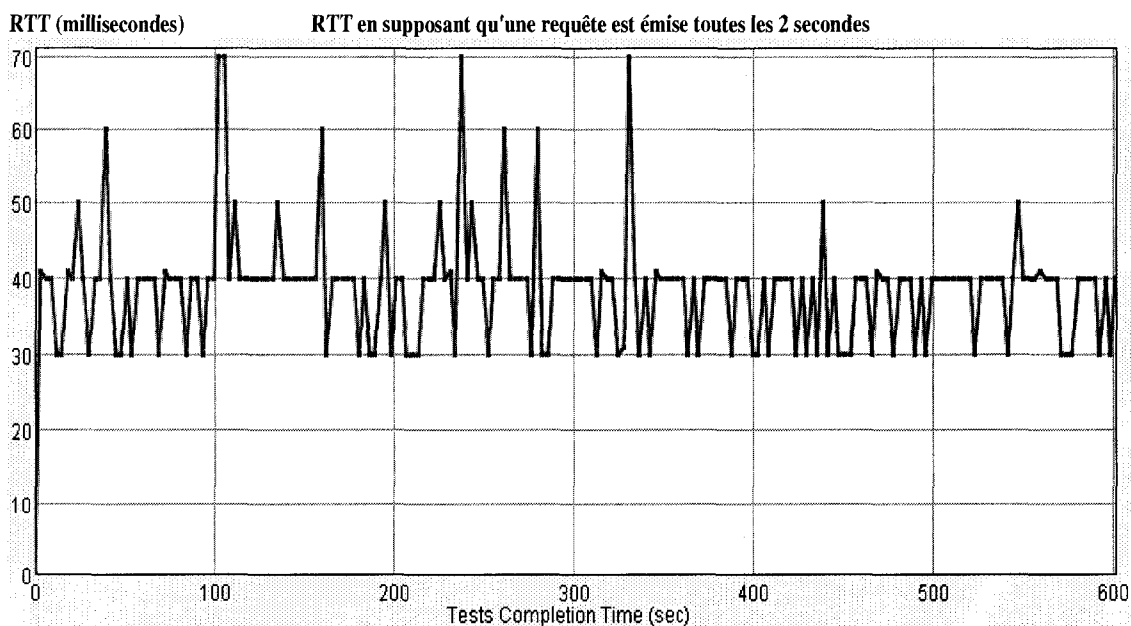


Figure 5.5 Recherche d'un service Web géolocalisé avec QoS

Pour mesurer l'influence du "buffer" du serveur GLWSM sur les performances RTT de la recherche de service Web géolocalisé, nous avons effectué le scénario décrit ci-dessous.

Scénario 2: croissance linéaire du nombre de clients effectuant une requête de 1 à 50

Ce scénario nous permet de mesurer l'influence du nombre d'utilisateurs en communication avec le serveur sur les performances RTT de la recherche d'un service Web géolocalisé avec QoS. Les requêtes, comme précédemment, sont envoyées toutes les deux secondes. Les résultats obtenus sont présentés par la Figure 5.6.

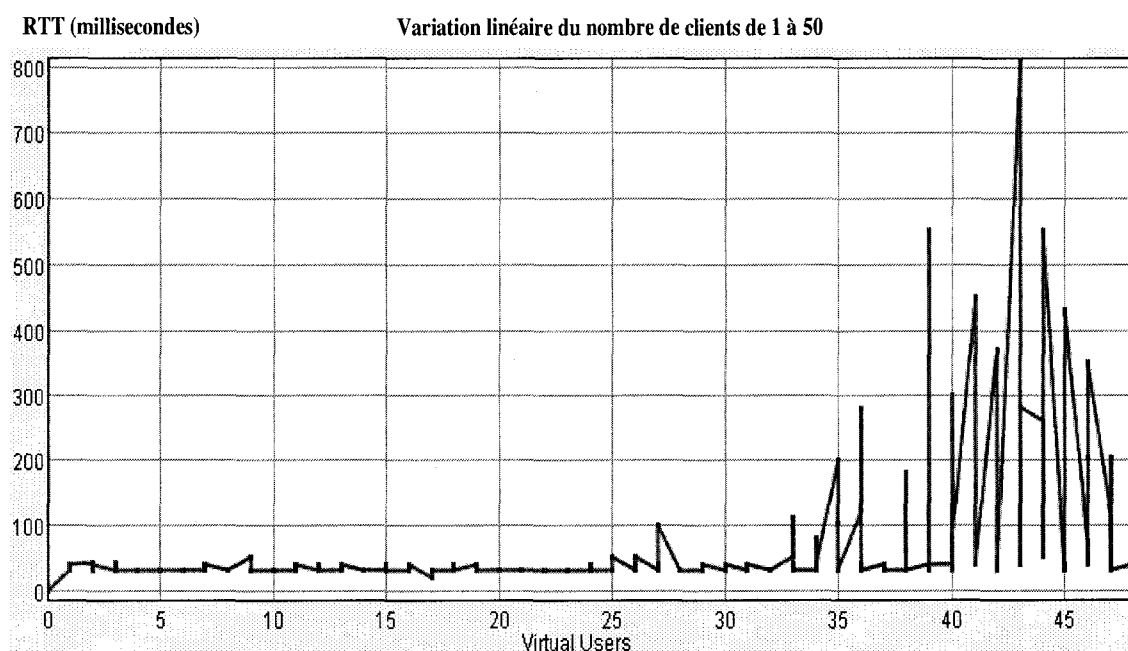


Figure 5.6 Variation linéaire du nombre de clients lors d'une recherche d'un service Web géolocalisé avec QoS

Nous obtenons un RTT moyen de 83 millisecondes avec un maximum de 810 millisecondes et un minimum de 35 millisecondes. Le serveur GLWSM est stable (c'est à dire que les résultats obtenus sont à l'intérieur des mesures obtenus précédemment en utilisant l'écart type de 15.57 millisecondes comme incertitude absolue par rapport à la moyenne précédente) quand le nombre de clients en communication est inférieur ou

égale à 28. Dans le cas où le nombre de clients est supérieur à 28, le serveur est instable car le “buffer” est généralement plein et génère des résultats peu fiables.

➤ ***Publier un service Web géolocalisé***

Pour publier un service Web géolocalisé, nous avons envoyé les six paramètres de type String totalisant 492 caractères, soit 984 octets, et deux structures d’agrément de nœuds ayant une taille totale de 184 octets. Les requêtes sont envoyées toutes les cinq secondes. Les informations sont envoyées au serveur GLWSM racine qui les enregistre, puis, par le biais d’un message asynchrone, propage les informations reçues à deux serveurs GLWSM situés au niveau des feuilles de la topologie du système GLWSA. Nous avons mesuré le délai RTT à la complétude de l’opération (Figure 5.7).

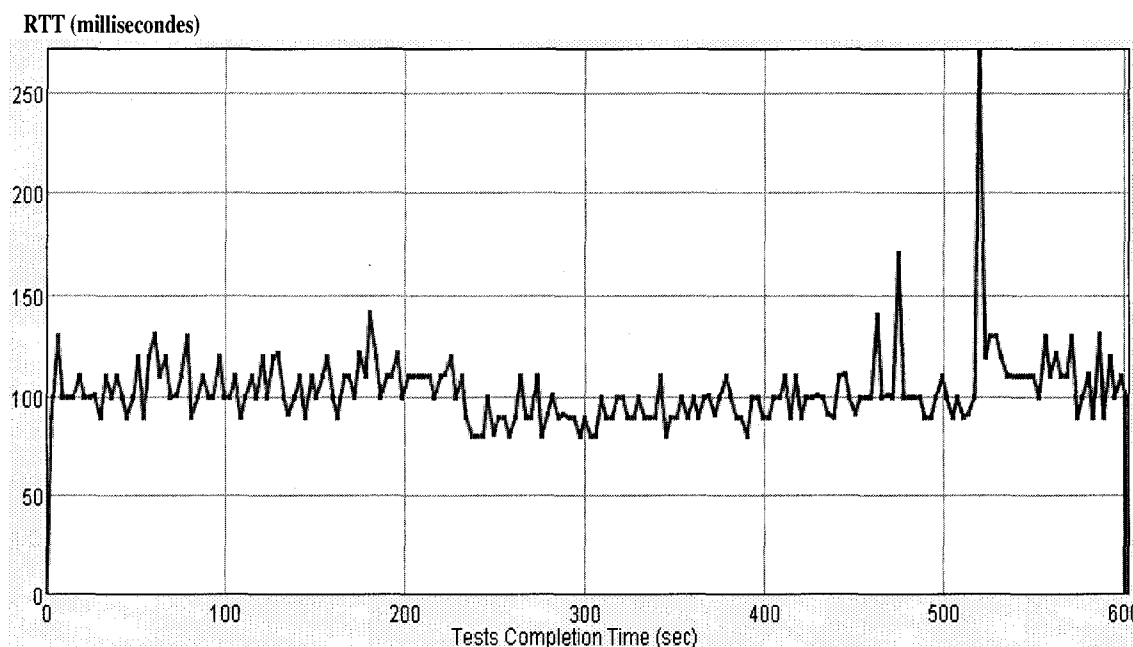


Figure 5.7 Publication d’un service Web géolocalisé

Nous obtenons un RTT moyen de 105 millisecondes avec un minimum de 80 millisecondes et un maximum de 300 millisecondes. La dispersion des mesures est environ de 18.30 millisecondes. La réalisation d’une opération de publication se fait suivant le requis *RNF.3* car tout nœud feuille GLWSM sélectionné lors de la publication

connaîtra après publication les autres nœuds où le service a été publié. Par conséquent, la sélection d'un service spécifique sera faite localement.

Nous avons aussi évalué la variation du temps de publication en fonction du nombre de nœuds de publication sélectionnés pour les cas 5 (Figure 5.8), 10 (Figure 5.9) et 20 nœuds (Figure 5.10). Nous constatons que RTT augmente en fonction du nombre de nœuds sélectionnés. Nous trouvons des RTT moyens de 110, 143 et 285 millisecondes respectivement pour 5, 10 et 20 nœuds.

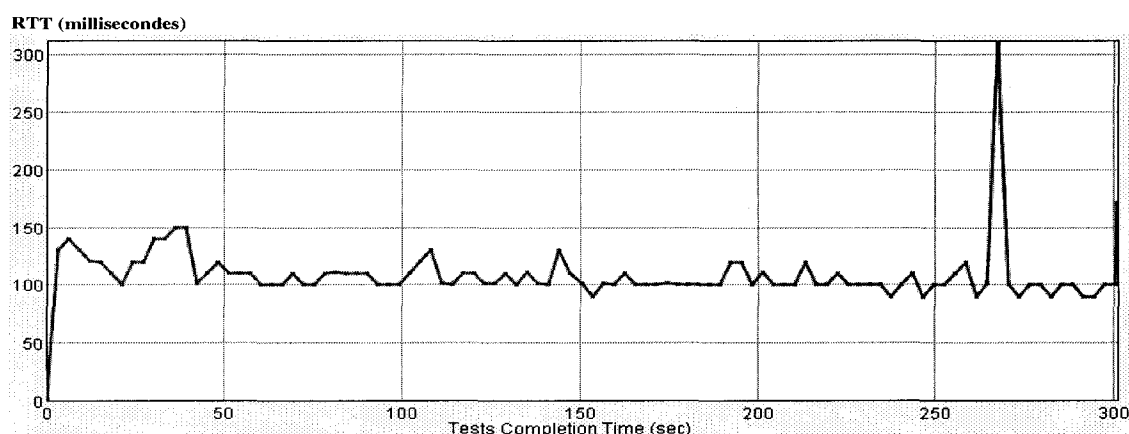


Figure 5.8 Publication d'un service Web géolocalisé à 5 nœuds « feuille »

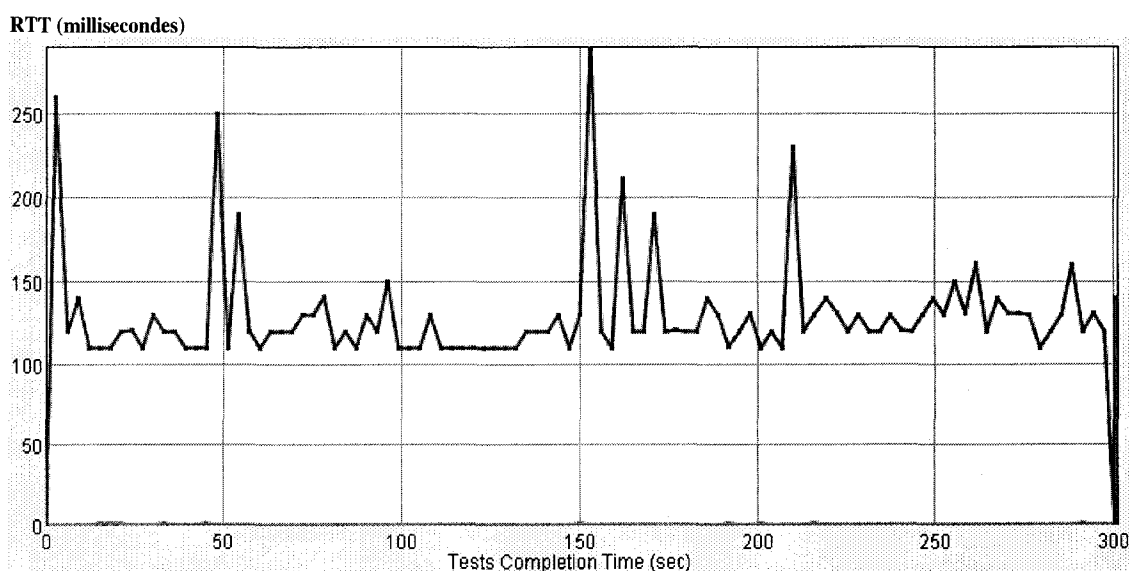


Figure 5.9 Publication d'un service Web géolocalisé à 10 nœuds « feuille »

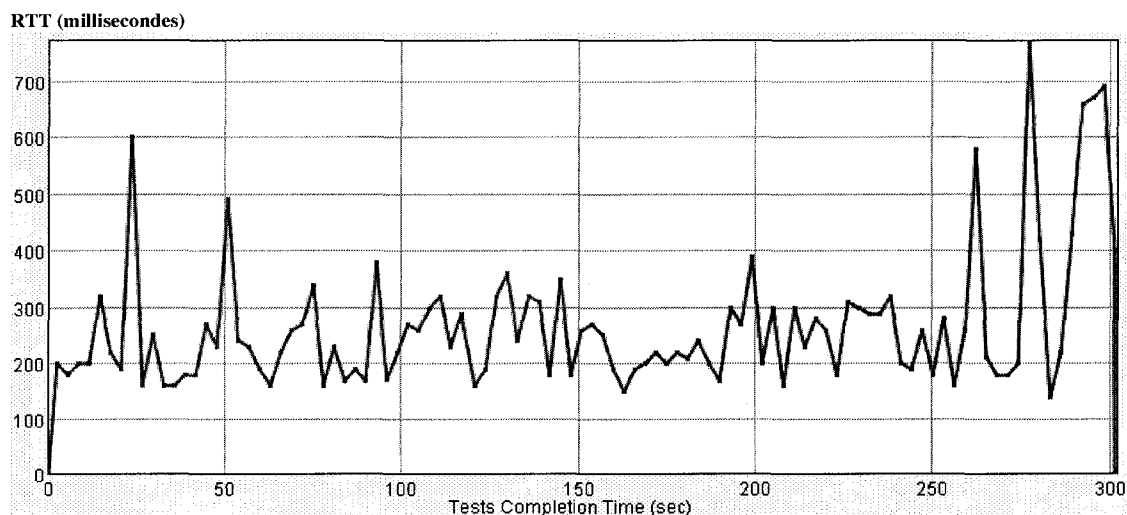


Figure 5.10 Publication d'un service Web géolocalisé à 20 nœuds « feuille »

Nous avons constaté qu'en effectuant une même série de mesures, les valeurs maximales varient mais la moyenne reste sensiblement la même. Toutefois, la croissance du RTT quand le nombre de nœuds croît n'est pas une mesure très sensible pour un fournisseur autorisé car, dans la conception du système nous supposons que la publication de services se fait à partir d'une machine fixe.

➤ ***Coordonner la migration d'un service Web géolocalisé sans QoS***

Le test de la coordination d'une migration sans QoS consiste à notifier le serveur d'applications SAS courant (où le service requis par un client est en cours d'exécution) de l'adresse URL du prochain serveur SAS où se trouve présentement le client et à envoyer les informations de suivi (l'identifiant du mobile, l'identifiant du service et l'URL du serveur d'applications où l'exécution du service a été migrée) au nœud suivant GLWSM qui couvre la position actuelle du mobile. Quand le serveur SAS courant a terminé la migration de service au SAS suivant, il envoie une notification de fin de migration au GLWSM courant qui coordonne la migration. Le GLWSM courant envoie un message de suivi de mobile au GLWSM suivant qui couvre la position actuelle du mobile. Du fait que la migration de SAS à SAS n'est pas de la portée du système

GLWSA, nous avons considéré, comme scénario pour les mesures, que le SAS courant retourne immédiatement un message de fin de migration au GLWSM courant après avoir reçu la notification de migrer le service au SAS suivant. Ce scénario permet de mesurer l'impact de l'architecture du système sur la migration de service sans QoS. Les résultats obtenus sont présentés à la Figure 5.11.

Sommairement, on obtient un temps moyen RTT de 11 millisecondes pour coordonner la migration d'un service. Il varie toutefois sur une plage de 7 millisecondes à 40 millisecondes. La dispersion des mesures est de 5.80 millisecondes par rapport à la moyenne. Nous constatons également que 98% des mesures effectuées sont inférieures au délai de coordination de migration de 30 millisecondes. Ce qui est bon indicateur que le surcoût de la coordination de migration de service est négligeable.

Pour la migration de service d'un SAS à un autre SAS, la migration de session ou de tâche peut être utilisée. Les résultats relevés dans la littérature [17] montrent que la migration de service d'un serveur d'applications varie en moyenne entre 3 millisecondes et 300 millisecondes dépendamment de la taille du service à migrer. Nous suggérons toutefois que la migration de tâche soit utilisée pour migrer un service de SAS à SAS car elle permet de réaliser une migration forte, c'est à dire une migration de service où les données et l'état d'exécution d'un service sont transférés.

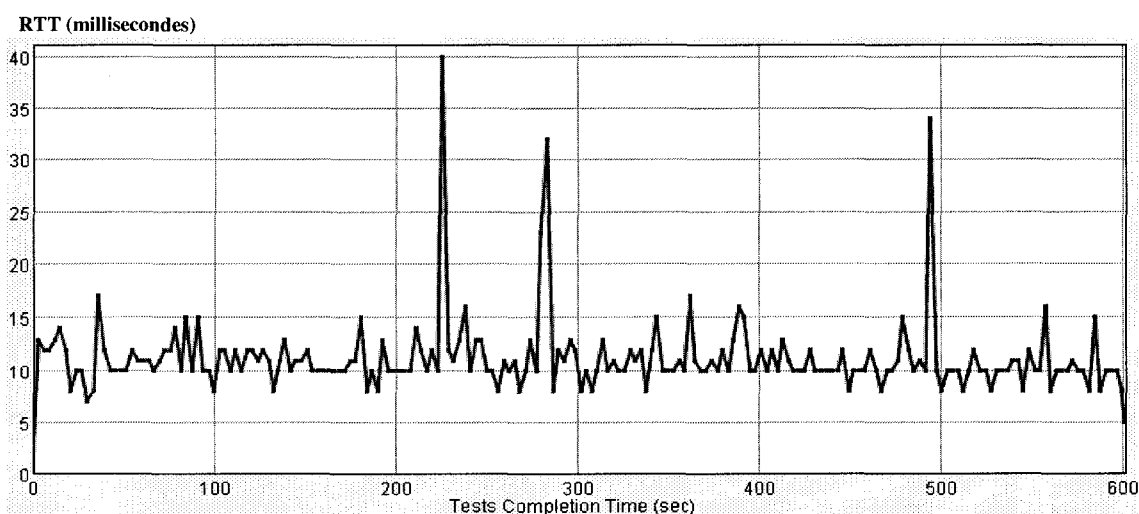


Figure 5.11 Coordination de migration d'un service sans QoS

Pour conserver la notion de géolocalisation du service quand le client sort de la zone de couverture d'un SAS courant, nous avons imposé lors de l'implémentation une continuité de services de 2 secondes maximum au SAS courant pendant laquelle la migration de service au prochain SAS doit être effective. En effet, pendant ce délai de migration, le client peut rester associé au SAS courant. Au delà de cette valeur limite, il faut que la migration de service soit effective.

Toutefois, du fait que pendant cette période le client mobile continue à se déplacer, nous avons analysé l'impact de la vitesse de déplacement sur les résultats. Il en ressort que la vitesse n'a pas d'impact direct sur les résultats relevés, par contre, il y a un risque que le client entre et sort d'un domaine contrôlé par le prochain nœud GLWSM couvrant la position courante du client mobile avant que la migration de services soit effective. En réalité, ce risque est négligeable. En effet, si on suppose par exemple que, pendant cette contrainte de temps que le client se déplace à 300 Kilomètres par heure, il ne pourra parcourir que 166.67 mètres durant ce temps. Cette distance est négligeable pour qu'on puisse dédier un MAP (qui agrège le trafic de plusieurs zones de localisation, sa capacité est de l'ordre des Térabits/s) juste pour cet espace géographique ayant un diamètre de couverture de 166.67 mètres.

En ajoutant la moyenne la plus élevée des résultats de migration de service (soit 300 millisecondes) à la plus grande valeur observée de la coordination de service (soit 55 millisecondes), nous trouvons une valeur de 355 millisecondes qui est bien inférieure au délai imposé de continuité de service au SAS courant, soit 2 secondes.

➤ ***Coordonner la migration d'un service Web géolocalisé avec QoS***

Le test de la coordination d'une migration avec QoS consiste à notifier le serveur d'applications SAS courant (où le service requis par un client est en cours d'exécution) de l'adresse URL du prochain serveur SAS où se trouve présentement le client et à envoyer les informations de suivi (l'identifiant du mobile, l'identifiant du service, URL du serveur d'applications où l'exécution du service a été migrée, le coût maximum du

service désiré, la bande passante requise, le facteur d'utilisation requis au SAS) au nœud suivant GLWSM qui couvre la position actuelle du mobile. Quand le serveur SAS courant a terminé la migration de service au SAS suivant, il envoie une notification de fin de migration au GLWSM courant qui coordonne la migration. Le GLWSM courant envoie un message de suivi de mobile au GLWSM suivant qui couvre la position actuelle du mobile. Du fait que la migration de SAS à SAS n'est pas de la portée du système GLWSA, nous avons considéré, comme scénario pour les mesures, que le SAS courant retourne immédiatement un message de fin de migration au GLWSM courant après avoir reçu la notification de migrer le service au SAS suivant. Ce scénario permet de mesurer l'impact de l'architecture du système sur la migration de service avec QoS. Les résultats obtenus sont présentés à la Figure 5.12.

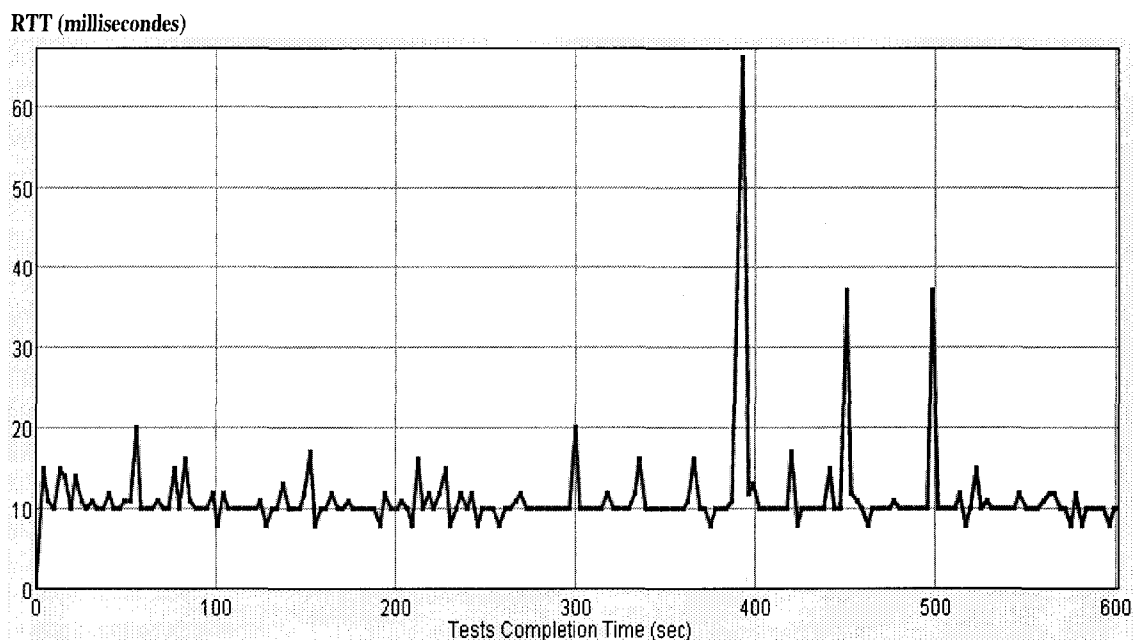


Figure 5.12 Coordination de migration d'un service avec QoS

Nous trouvons un temps moyen RTT de 13 millisecondes pour coordonner la migration de service avec QoS. Le temps maximal mesuré est de 65 millisecondes et le temps minimal est de 8 millisecondes. Les valeurs relevées ont une dispersion de 6.47 millisecondes. L'infime variation du RTT moyen ($\Delta RTT = 2$ millisecondes) par rapport

à la migration de service sans QoS est liée à l'augmentation (soit 24 octets) de la taille globale des paramètres échangés lors de transmission du message de suivi des mobiles au GLWSM suivant. Tout comme pour la coordination de migration sans QoS, nous constatons que 99% des mesures effectuées sont inférieures au délai de coordination de migration de 30 millisecondes. Ce qui est bon indicateur que le surcoût induit par la coordination de migration de service est négligeable.

➤ ***Localisation thématique des mobiles***

La localisation des groupes de mobiles a été réalisée afin d'évaluer le concept de localisation thématique. Pour effectuer les tests, nous avons développé deux méthodes *locateSubject* et *locateMsids*. La méthode *locateSubject* permet de localiser un groupe de mobiles en envoyant au serveur LCS le nom du sujet qui les lie. La méthode *locateMsids* permet de localiser un groupe de mobiles en envoyant au serveur LCS une liste des mobiles. Le code client a consisté à construire un code simple appelant les méthodes à tester en utilisant le langage SOAP (plus précisément celui de Apache Axis).

Pour calculer le temps mis pour que l'appel de la méthode *locateSubject* soit reçu au serveur, nous avons effectué la somme entre:

- la différence de temps entre le temps de réception de la requête par le serveur et le temps d'envoi de la requête ;
- et le temps de recherche des mobiles liés par le sujet dans la base de données « ClientInfosDB » côté serveur.

Pour calculer le temps mis pour que l'appel de la méthode *locateMsids* soit reçu au serveur, nous avons effectué la somme entre:

- le temps de lecture des identifiants des mobiles dans la base de données « ClientInfosDB » côté client ;
- et la différence de temps entre le temps de réception de la requête par le serveur et le temps d'envoi de la requête.

Le temps de localisation des mobiles proprement dite n'a pas été pris en compte car il est le même dans les deux méthodes. En effet, la localisation des mobiles côté serveur est une localisation individuelle par identifiant de mobile.

La Figure 5.13 montre les résultats obtenus avec un sujet mis à sa taille maximum, soit 250 caractères, et la taille d'un identifiant fixé à 20 caractères. De manière générale, la localisation thématique offre de meilleurs résultats principalement à cause de la différence entre le temps de réception et le temps d'émission de la requête. Le gain de la localisation augmente quand le nombre de mobiles croît.

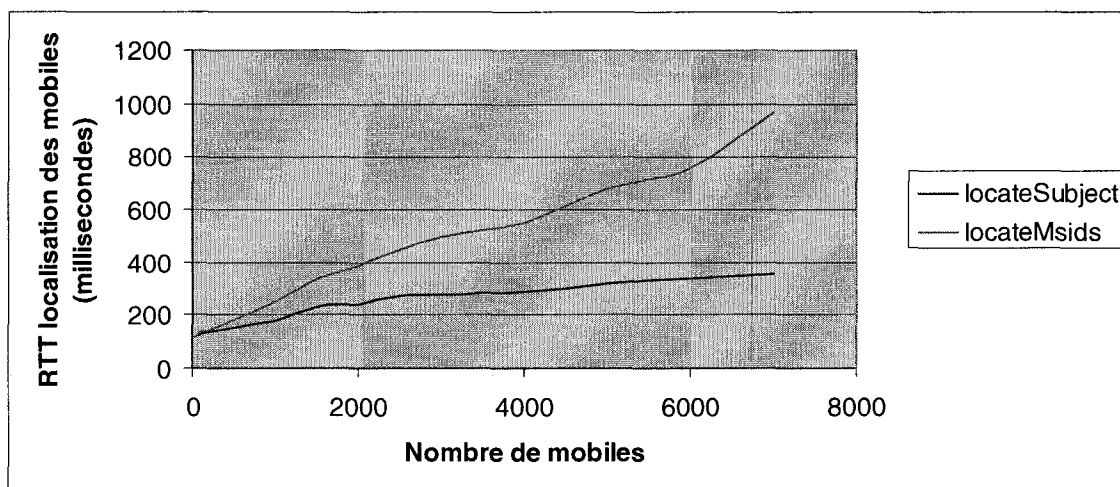


Figure 5.13 Localisation thématique versus localisation par groupe de mobiles

5.2 Évaluation de la collecte des données dynamiques de QoS

Pour évaluer le principe de la collecte des données que nous avons proposé, le simulateur NS-2.27 (Network Simulator version 2.27) a été utilisé. Nous avons modifié le code RSVP afin d'y ajouter les commandes de soumission de la bande de passante.

Les commandes implémentées sont COLLECT_TRAF, QUOT_REQ, TRANSMIT_QUOT et RELEASE_SES. Le rôle de chacune de ces commandes est expliqué à la section (4.3.4) du chapitre 4. La configuration topologique utilisée est illustrée à la Figure 5.14. La topologie est constituée de quatre serveurs : un GLWSM et

trois SAS. Les données des liens permettant l'interconnexion du réseau sont contenues au Tableau 5.4.

Nous supposons que les nœuds SAS_1 , SAS_2 et SAS_3 génèrent un trafic CBR (Constant Bit Rate) de 2 Mbits/s respectivement à 2 secondes, 7 secondes et 11 secondes aux mobiles MN_1 , MN_2 et MN_3 . Nous avons fixé les tailles des paquets à 1000 octets. À $t=1, 5$ et 10 secondes, les nœuds MN_1 , MN_2 et MN_3 réservent chacune une bande passante de 2Mbits/s. Au temps $t=13$ secondes, le nœud GLWSM envoie une requête de collection de trafic COLLECT_TRAF aux trois serveurs d'applications soient SAS_1 , SAS_2 et SAS_3 . Le GLWSM fournit, lors de cette transmission de message, l'adresse du MAP aux SAS. Alors, chacun des SAS envoie une requête au MAP, une demande de collection de la bande passante QUOT_REQ. À la réception de cette soumission, chaque serveur SAS libère la session et envoie le résultat obtenu au serveur GLWSM.

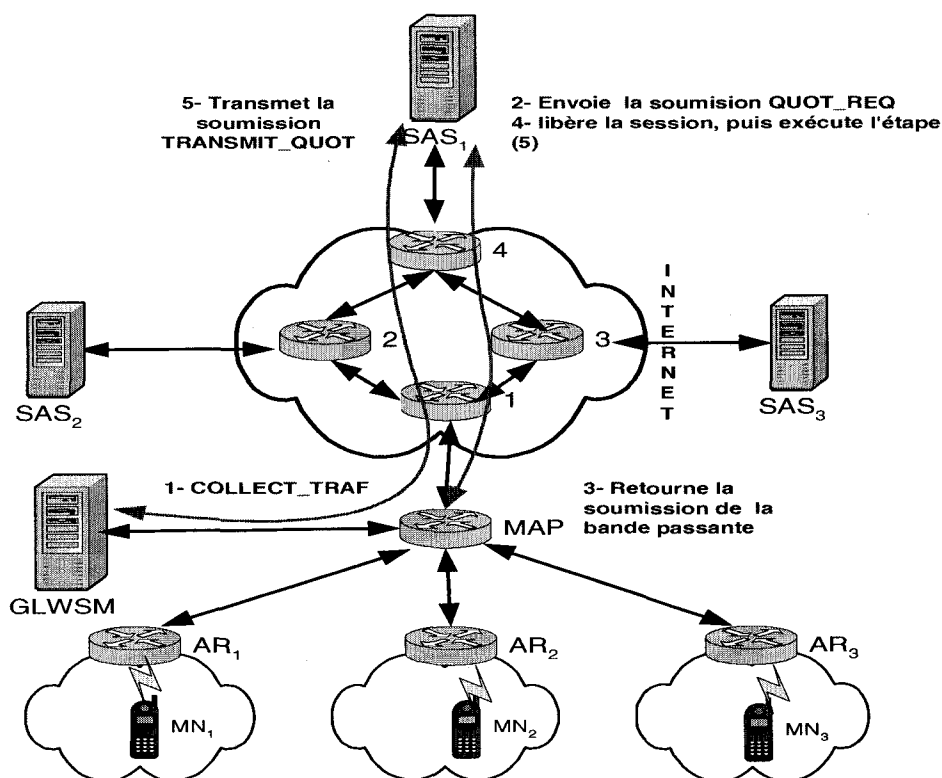


Figure 5.14 Topologie de la simulation

Tableau 5.4 Valeurs des liens

Lien	Bande passante	Délai de propagation	Taille de la queue
SAS ₁ -Node ₄	6Mbits/s	0.25ms	6Kbits
Node ₄ -Node ₃	25Mbits/s	3ms	300Kbits
Node ₄ -Node ₂	25Mbits/s	1ms	100Kbits
SAS ₂ -Node ₂	8Mbits/s	0.25ms	8Kbits
SAS ₃ -Node ₃	10Mbits/s	0.25ms	10Kbits
Node ₃ -Node ₁	25Mbits/s	2ms	200Kbits
Node ₂ -Node ₁	25Mbits/s	1ms	100Kbits
Node ₁ -MAP	40Mbits/s	1ms	160Kbits
GLWSM-MAP	10Mbits/s	0.25ms	10Kbits
MAP-AR ₁	5.4Mbits/s	0.25ms	5.4Kbits
MAP-AR ₂	5.4Mbits/s	0.25ms	5.4Kbits
MAP-AR ₃	5.4Mbits/s	0.25ms	5.4Kbits
AR _i -MN _i i=1, 2, 3	2Mbits/s	5ms	2Kbits

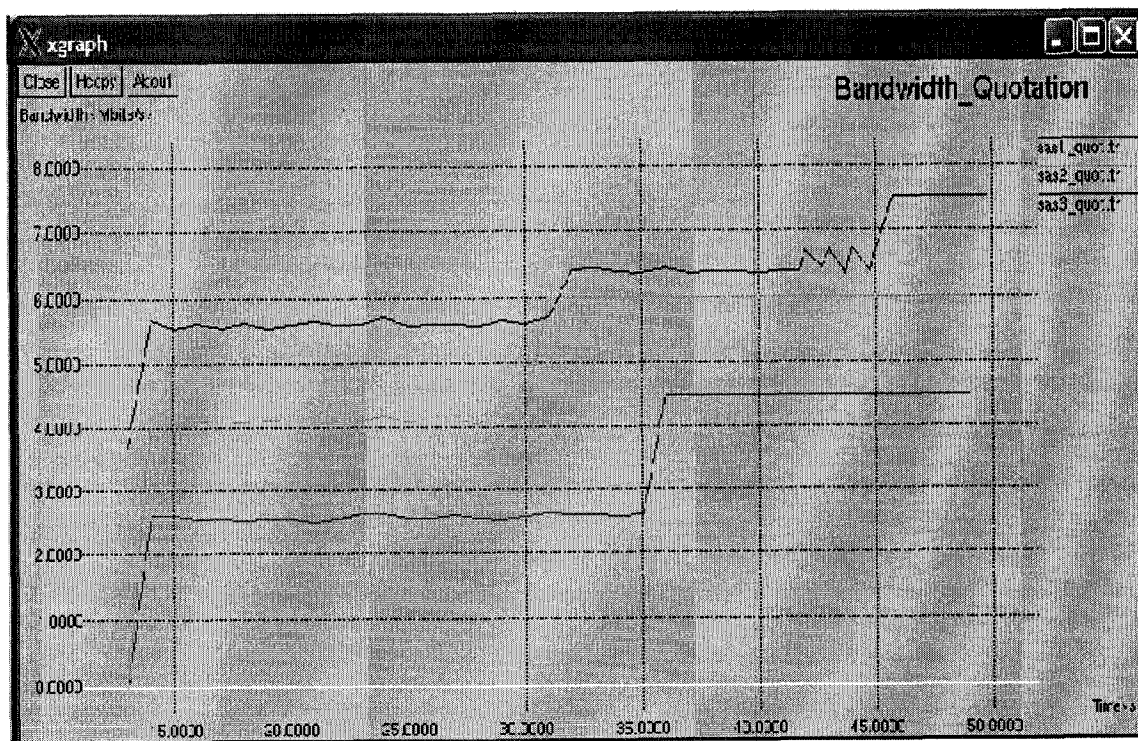


Figure 5.15 Soumission de la bande passante

Quand le serveur GLWSM est en possession de tous les résultats, il peut sélectionner le serveur offrant la meilleure bande passante lors d'une recherche de

service. Nous avons mesuré, en fonction du temps, la bande passante disponible que renverrait chacun des serveurs SAS au GLWSM (Figure 5.15). Dans cette configuration, le meilleur serveur en terme de bande passante disponible est le serveur SAS₃. L'acquisition des données est périodique (rafraîchissement des données toutes les 30 secondes).

Le temps moyen pour propager les données dynamiques de la qualité de services de service de GLWSM à GLWSM, est déterminé en utilisant une queue de publication. Ce temps moyen est environ de 10 millisecondes. Dans le système GLWSA, les erreurs de communication liées à un défaut du réseau (par exemple, une charge élevée du réseau) subi par un nœud GLWSM n'affectent pas les autres nœuds GLWSM contrairement au système QoS-Aware [30].

Pour mesurer la consistance des données de soumission de la QoS dans le système GLWSA et QoS-Aware, nous avons utilisé la métrique de la « réponse du rafraîchissement » (*update responsiveness*) [28]. La réponse du rafraîchissement pour un serveur de découverte de services qui reçoit la soumission est la proportion du temps restant au délai de propagation quand les données de QoS soumis sont reçues. La réponse du rafraîchissement pour un serveur de découverte de services DS_j est donnée par les équations (5.1) et (5.2).

$$R_j = 1 - L_j \quad (5.1)$$

$$L_j = \frac{t_j - t_0}{DD - t_0} \quad (5.2)$$

t_0 est le temps correspondant à l'émission de la soumission des données de la QoS; t_j est le temps correspondant à la réception de la soumission des données de la QoS au serveur de découverte de services DS_j.

La réponse du rafraîchissement moyen du système (GLWSA ou QoS-Aware) est donné par:

$$\bar{R} = \frac{1}{n-1} \sum_{j=1}^{n-1} R_j \quad (5.3)$$

n est le nombre de serveurs de découverte de services qui maintiennent les données de QoS soumises dans le système.

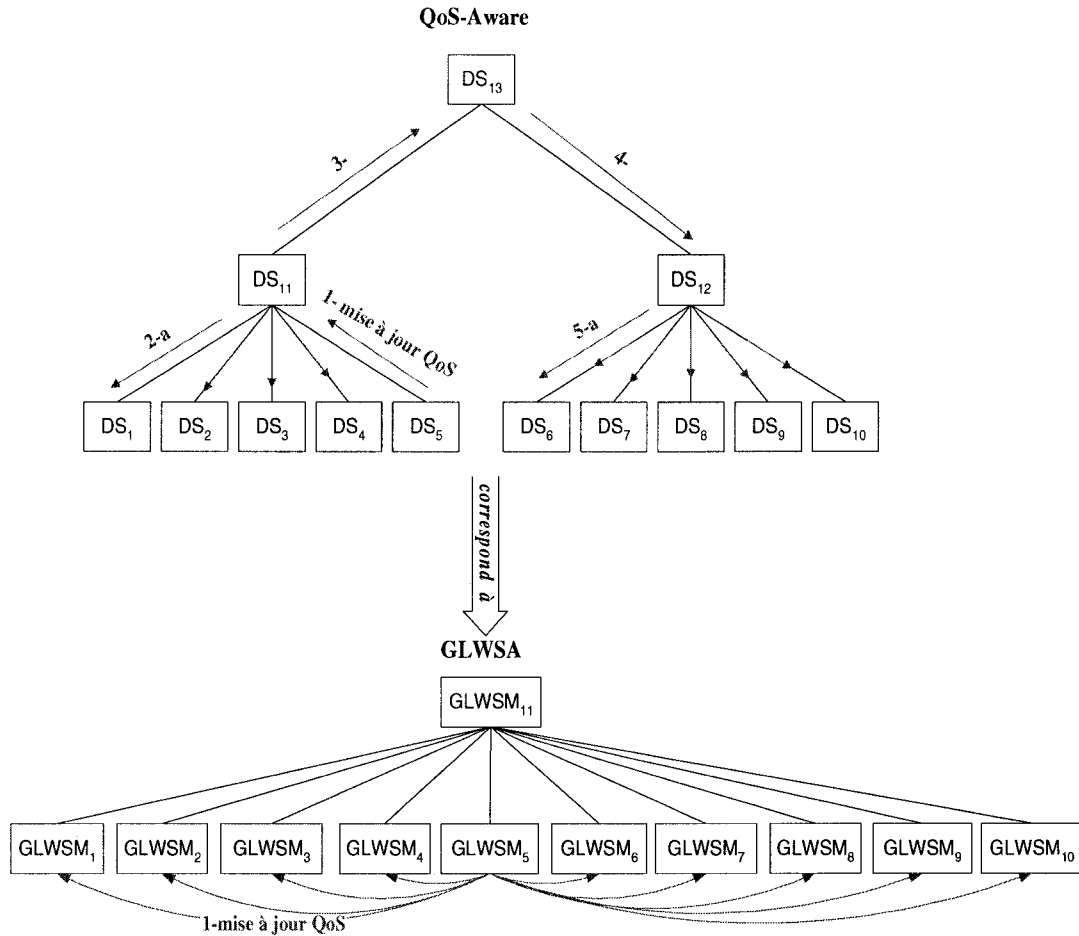


Figure 5.16 Mise à jour et propagation de la soumission des données de QoS dans les systèmes QoS-Aware et GLWSA

Pour comparer les deux systèmes, nous avons utilisé la topologie de QoS-Aware (Figure 5.16) et l'avons transformée en une topologie GLWSA équivalente. Le système a 13 serveurs de découverte de services et nécessite cinq étapes pour propager les données de soumission de la QoS envoyées par un serveur du niveau feuille à tous les

autres serveurs du niveau feuilles (le nombre des autres serveurs niveau feuille est 12) qui conservent les données de QoS émises. La topologie équivalente du système GLWSA a besoin d'une seule étape pour propager la soumission de données de QoS émises par un nœud feuille de découverte de services aux autres nœuds feuilles qui conservent les données de QoS soumises (le nombre des autres nœuds feuille est 9).

Dans la transformation les nœuds intermédiaires DS_{11} et DS_{12} du système QoS Aware ne sont pas nécessaires dans le système GLWSA. Une autre distinction entre les deux systèmes est que le nœud racine n'est pas impliqué dans la propagation de la soumission de QoS dans le système GLWSA.

À la Figure 5.17, nous comparons la réponse du rafraîchissement moyen dans les deux systèmes. Nous supposons que s'il n'y a pas de délai d'attente QD_j à un serveur de découverte de services serveur DS_j ou $GLWSM_j$, le temps de réception de la soumission t_j est égale au temps d'émission t_0 . Il est à noter que QD_j représente la différence de temps entre t_j et t_0 .

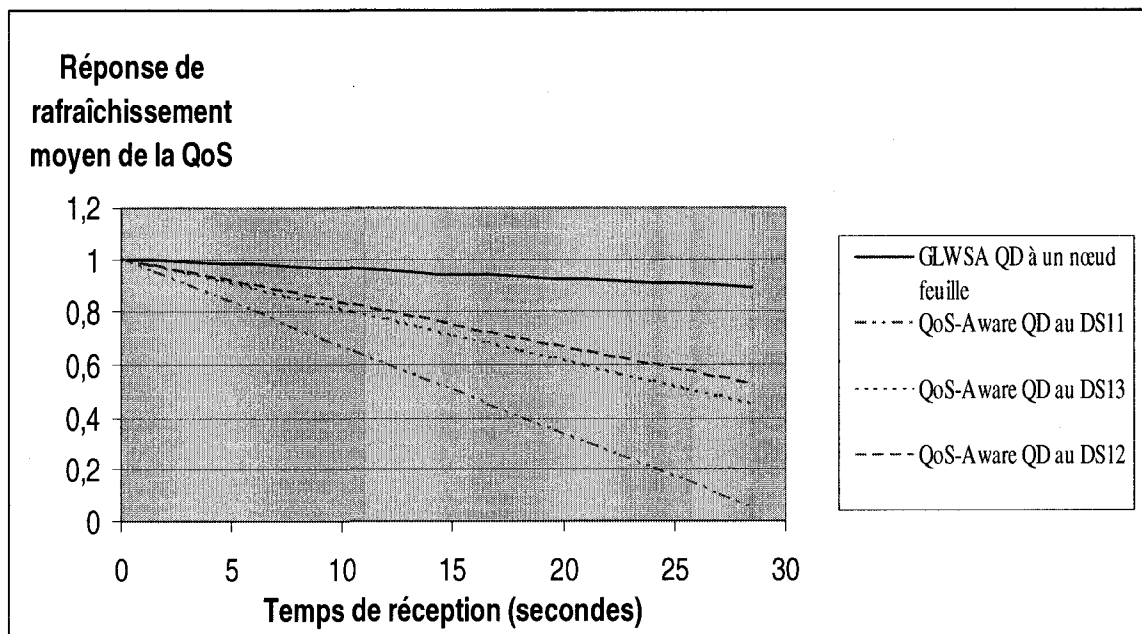


Figure 5.17 Réponse du rafraîchissement moyen

En pratique, nous avons choisi un nœud feuille qui initie la soumission des données de QoS (par exemple, le nœud 5). Ensuite, nous générons un délai QD à un nœud de découverte des services qui maintient les données de QoS soumises et mesurons la réponse du rafraîchissement moyen. Nous avons représenté la réponse du rafraîchissement moyen avec une valeur $t_0=0$ et en augmentant progressivement la valeur de QD de 0% à 95% du délai maximum de propagation toléré $DD=30$ secondes. En comparant les deux systèmes, le système GLWSA donne de meilleurs résultats en utilisant la métrique réponse du rafraîchissement moyen si le délai QD apparaît à un nœud intermédiaire ou au nœud racine du système QoS-Aware.

CHAPITRE VI

CONCLUSION

Les services Web géolocalisés dans les réseaux mobiles font face à trois problèmes majeurs: la découverte des services quand un client mobile est dans une région donnée, la maintenabilité de l'exécution du service quand un client sort de la région géographique couverte par un serveur d'applications et l'interrogation d'un groupe de mobiles lié par un thème. Nos travaux ont tenté d'apporter des solutions à ces problèmes. En effet, le système GLWSA est une architecture adéquate de par les résultats pour découvrir les services Web géolocalisés, coordonner la migration de service au serveur d'applications le plus proche et localiser thématiquement les clients mobiles. Il permet de rechercher un service Web géolocalisé avec un temps RTT moyen de 28 millisecondes (pour un service Web géolocalisé sans QoS) et 41 millisecondes (pour un service Web géolocalisé avec QoS). Les performances de la coordination de la migration sans ou avec qualité de service permet d'effectuer la migration de service en respectant le requis non fonctionnel RNF.6 (délai de contrainte de migration inférieur ou égal à deux secondes). Nous avons démontré que le mécanisme de collection et de soumission des données de la QoS du système GLWSA est consistant par rapport à d'autres systèmes effectuant également la collection des données dynamiques de la QoS (par exemple, la bande passante). Nous démontrons également dans les résultats obtenus que la localisation thématique offre un gain substantiel en temps par rapport à la localisation ordinaire d'un groupe de mobiles. Dans l'organisation de ce chapitre, nous faisons au préalable une synthèse de notre contribution, nous relevons par la suite les limites de l'architecture et indiquons des pistes de travaux futurs.

6.1 Synthèse des travaux

Le système GLWSA vient résoudre les trois défis majeurs soulevés par la découverte des services Web géolocalisés dans les réseaux mobiles. Afin de réaliser ces défis, nous les avons transformés en objectifs de la thèse. Les objectifs que nous avons utilisé sont les suivants :

- de proposer un modèle de découverte des services Web géolocalisés : ce modèle permet de publier les services géolocalisés, de rechercher un service géolocalisé sur la base du contexte de localisation de l'utilisateur et de lier le client mobile au serveur d'applications géographiquement le plus proche ;
- de définir d'une topologie de répartition des nœuds ;
- de proposer un modèle pour coordonner la migration de services ;
- de proposer un mécanisme de collecte de données de QoS (bande passante et facteur d'utilisation des serveurs d'application) pour un service donné et dans un domaine spécifique ;
- de proposer un modèle de factorisation thématique des services géolocalisés communs.

Dans l'analyse et la conception du système, les objectifs et les contraintes des défis à résoudre ont été transformés en des requis fonctionnels et non fonctionnels. Les requis fonctionnels nous ont permis de développer les fonctionnalités du système. Les requis non fonctionnels nous ont permis d'explicitier les contraintes de performances et de communication inter systèmes telles que (l'expansion des nœuds de la topologie (*RNF. 1*), l'interopérabilité du système proposé avec les entités externes (*RNF. 2*), la réduction du temps global de transit entre un serveur de découverte des services et son MAP associé (*RNF. 3*), l'autonomie de prise de décision dans la sélection d'un serveur d'applications (*RNF. 4*), la consistance des données de QoS (*RNF. 5*) et la contrainte sur le délai de coordination de migration (*RNF. 6*).

Les requis fonctionnels et non fonctionnels nous ont conduit à la définition d'une architecture ayant une topologie hiérarchique à deux niveaux : le niveau racine et le

niveau des feuilles. Le niveau racine est affecté à un nœud unique GLWSM (serveur de découverte des services Web géolocalisés) et le niveau des feuilles est occupé par différents GLWSMs associés chacun à un MAP du réseau mobile. Le GLWSM racine permet de maintenir la consistance des données réparties dans le système lors d'une publication de service. Les opérations de gestion de la topologie et de publication de services sont coordonnées par ce nœud. Le niveau des feuilles permet également à un client de rechercher un service et de coordonner la migration d'un service en cours d'exécution. Les nœuds feuille gèrent aussi le mécanisme de collecte de la qualité des services dans un domaine donné et pour un service donné, et de la propager aux autres nœuds du système offrant le même service. Nous avons effectué une modification du protocole MLP afin d'y ajouter la localisation thématique proposée. Nous appelons le protocole ainsi étendu MLPe.

Le système GLWSA est constitué principalement de trois entités (le serveur GLWSM, les bases de données UDDIM et "ClientInfosDB") réparties à chaque nœud GLWSM de la topologie. Le serveur GLWSM est l'organe central contenant la logique du système (fonctionnalités du système). La base de données UDDIM est une extension du registre UDDI dont le protocole associé UDDI a été choisi comme protocole de base pour construire les fonctionnalités de découverte de services du système. La base de données "ClientInfosDB" contient les informations personnelles d'un client.

L'architecture fonctionnelle proposée est orientée services et offre des API synchrones et asynchrones. La communication synchrone est utilisée entre le système GLWSM et les entités externes (client, le serveur LCS, le serveur SAS). La communication asynchrone est utilisée pour transmettre des informations de suivi d'un mobile ou de propagation de la QoS entre différents serveurs GLWSMs. Le registre UDDIM ajoute des structures de données (agrément d'un service, agrément d'un nœud, le nœud, et le segment de droite permettant d'identifier une limite de la couverture géographique d'un nœud) et des API pour gérer ces structures de données. Le protocole MLPe ajoute deux nouvelles structures (le thème et la souscription à un thème d'un

client donné) au protocole MLP et des API pour interroger un groupe de mobiles lié par un thème et gérer les deux structures de données proposées.

Nous avons évalué les performances du système GLWSA afin de mesurer le temps requis pour effectuer les opérations proposées par le système. Globalement, le système offre de bons résultats pour la recherche d'un service Web géolocalisé avec ou sans QoS (41 et 28 millisecondes), la coordination d'une migration de service Web géolocalisé avec ou sans QoS (15 et 11 millisecondes), la publication d'un service Web géolocalisé (105 millisecondes pour une publication à deux nœuds feuilles GLWSM). Nous avons également évalué la localisation d'un groupe de mobiles afin de mesurer le concept de localisation thématique. Il en ressort que la localisation thématique offre des meilleurs résultats que la localisation par envoi d'une liste de mobiles au serveur LCS. Ce gain s'accroît quand le nombre de mobiles s'accroît. Nous avons aussi démontré que le mécanisme de collecte et de propagation des données de QoS permet de maintenir des données consistantes de soumission de la QoS par rapport au système QoS-Aware qui offre également un mécanisme de collecte de données de QoS.

6.2 Limitations des travaux

Pour effectuer les travaux de cette thèse, nous avons fait certaines hypothèses sur l'utilisation de MLP comme protocole d'interopérabilité afin d'interagir avec le serveur LCS. Un autre groupe chercheurs OGC (*Open GeoSpatial Consortium*) [63] propose l'utilisation d'un protocole appelé OpenLS (*Open Location Service*) pour effectuer l'interopérabilité avec les serveurs LCS. L'architecture proposée ne permet pas d'interagir avec le protocole OpenLS. Toutefois, des travaux sont en cours dans le groupe OGC pour permettre une transparence de l'utilisation de MLP dans OpenLS ou une adaptation du protocole OpenLS au protocole MLP.

Le système Ericsson MPS 6.0 utilisé lors des tests pour localiser les services retournent la position des mobiles sous forme de données spatiales (longitude, latitude, altitude), ce qui ajoute un coût de transformation des données en coordonnées

cartésiennes afin de les rendre compatibles avec les informations sur la couverture régionale d'un nœud GLWSM. La méthode de calcul pour effectuer la détermination de la position d'un mobile à partir du fichier statique de son déplacement dans la topologie, n'est pas fournie par le logiciel, ce qui rend difficile l'interprétation de la mesure de la position retournée par le serveur LCS.

Le système proposé a été développé pour apporter les solutions aux défis soulevés. Il va de soi que certains aspects conceptuels n'ont pas été abordés car n'apportant aucune valeur ajoutée par rapport aux objectifs. Il s'agit de la création des interfaces graphiques pour rendre le système convivial, la personnalisation de la présentation des messages aux clients mobiles et la confidentialité et l'intégrité des messages échangés entre les différentes entités du système.

6.3 Travaux futurs

Certains aspects de l'architecture n'ont pas été développés car ne faisant pas partie de nos objectifs. Il s'agit par exemple de la flexibilité dans la présentation des messages ou données à un client mobile. La flexibilité ou adaptabilité de la présentation des données permet de tenir compte des caractéristiques de la machine pour le choix de la présentation/visualisation des messages dans les interfaces graphiques. Par exemple, elle permettrait d'adapter la présentation des messages au format WML (*Wireless Markup Language*, cas d'un équipement utilisant le système WAP, *Wireless Application Protocol*), au format cHTML (*compact HTML*, utilisé par des équipements dans la région asiatique) ou au format xHTML (*XML HTML*) qui est plus générique dans l'utilisation de XML avec le HTML. Le présent système GLWSA est adapté à une présentation des données sous le format SOAP. L'adaptabilité de la présentation des données du système GLWSA à d'autres langages de présentation des données peut se faire en utilisant les feuilles de transformation de style XSLT (*eXtensible Style Language Transformation*) pour mettre le format sous une forme compatible à un autre

langage. Cette transformation est possible car tous ces langages utilisent des balises du langage XML.

Pour rendre le système convivial, il faudra lui adjoindre des interfaces graphiques. Les interfaces graphiques permettront à des utilisateurs non familiers aux langages de programmation et aux différents protocoles intégrés dans le système, de réduire le temps d'apprentissage à l'utilisation et à la compréhension des interactions avec les fonctionnalités du système GLWSA. Les interfaces graphiques à développer seront étroitement liées au module de flexibilité ou d'adaptabilité de la présentation des données. En effet, l'interface graphique sera personnalisée par rapport à la technologie de présentation supportée par l'équipement mobile. Ce qui signifie que, lors de la génération d'un objet interface graphique, l'attribut *langPres* de la classe *MobileEquipment* sera consulté pour vérifier le langage de présentation de l'équipement mobile. Ainsi pour chaque équipement d'un client donné, un profil de présentation sera associé aux services souscrits [10].

La confidentialité et l'intégrité des données transmises entre entités devront être développées afin de garantir que les données sensibles telles que la position des clients et l'identité des équipements mobiles ne soient transmises en clair ou altérées pendant leur transmission vers une entité réceptrice. Pour réaliser ces fonctionnalités de sécurité, une politique de sécurité devra être établie pour garantir un échange sécuritaire et protéger le système contre les attaques potentielles (écoute des messages, altération des messages). Afin de garantir la non répudiation des messages, nous pensons que les messages devraient être signés pour vérifier l'authenticité de leur origine. Pour implémenter la sécurité des données transmises, on pourrait s'inspirer des travaux de recherche [47] qui abordent cet aspect de la sécurité dans un environnement mobile en utilisant une infrastructure distribuée de clés publiques PKI (*Public Key Infrastructure*), des certificats autorisés par une autorité de certification et des clés privées.

Des tests peuvent être aussi effectués en faisant varier le temps d'interarrivée des requêtes afin d'apprécier l'influence de la taille du « buffer » sur les résultats obtenus.

Références bibliographiques

- [1] 3GPP, *Functional stage 2 description of location services in UMTS*, Technical Specifications TS 23.171 V3.9.0, Reference http://www.arib.or.jp/IMT-2000/ARIB-spec/ARIB/23171_300.PDF, September 2002.
- [2] 3GPP2, *“IP Network Architecture Model for cdma2000 Spread Spectrum Systems”*, Technical Specification S-R00-37.0 v2.0, Reference http://3gpp2.org/Public_html/specs/S.R0037-0_v2.0.pdf, May 2000.
- [3] 3GPP2, *“Location Based Services System (LBSS), Stage 1 Description”*, S.R0019 v1.0.0, http://www.3gpp2.org/Public_html/specs/S.R0019_LBSS_Stage_1_v1.0.0.pdf, 2000.
- [4] R. Al-Ali, O. Rana, D. Walker, S. Jha and S. Sohail., *“G-QoS: Grid service discovery using QoS properties”*, Computing and Informatics Journal, Special Issue on Grid Computing, pp 363–382, 2002.
- [5] R. Al-Ali, A. Shaikh Ali, O. Rana and D. Walker, *“Supporting QoS-based discovery in service-oriented Grids”*, Parallel and Distributed Processing Symposium, 2003. Proceedings International, pp 9, 22-26 April 2003.
- [6] A. Alonistioti, S. Panagiotakis, N. Houssos and A. Kaloxylos, *“Issues for the provision of Location-dependent services over 3G networks”*, International Symposium on 3rd Generation Infrastructure and Services, Athens, Greece, Reference http://www.cnl.di.uoa.gr/publications/3gis_LocationServices_final.pdf, July 2001.
- [7] G. Ballintijn, A. S. Tanenbaum and M. R. Van Steen, *“Locating Objects in a Wide-area System”*, PhD Thesis, Amsterdam University, Globe Project, Reference http://www.cs.vu.nl/res/theses/ballintijn_thesis.pdf, 2003.
- [8] S. Banerjee, S. Agarwal, K. Kamel, A. Kochut, C. Kommareddy, T. Nadeem, P. Thakkar, B. Trinh, A. Youssef, M. Youssef, R. Larsen, A. U. Shankar and A. Agrawala, *“Rover Technology: Enabling Scalable location-aware computing”*, UMIACS-TR 2001 and CS-TR 4312, Reference http://citeseer.nj.nec.com/cache/papers/cs/25982/http:zSzzSzwww.cs.umd.edu/zSz~moustafa_zSzpaperszSzcs_tr_4312.pdf/banerjee01rover.pdf, December 2001.

- [9] M. Barbeau, "*Bandwith Usage Analysis of Service Location Protocol*", Workshop on Pervasive Computing, International Conference on Parallel Processing, Toronto, Reference <http://citeseer.nj.nec.com/barbeau00bandwidth.html>, August 2000.
- [10] A.C Bayomock Linwa, B. Kerhervé et M. Éttoré, « *Gestion de la qualité de service : profil utilisateur et la persistance de données* », Mémoire de maîtrise en génie informatique, École Polytechnique de Montréal, Avril 2001.
- [11] A.C Bayomock Linwa and S. Pierre, "*Discovering Architecture of Geo-Located Web Services for Next Generation of Mobile Networks*", submitted to IEEE Mobile Computing Transactions, March. 2005.
- [12] A.C Bayomock Linwa and S. Pierre, "*Discovering with QoS the Geo-Located Web Services over Next Generation of Mobile Networks*", WiMob conference, Accepted April 2005.
- [13] A.C Bayomock Linwa and S. Pierre, "*Discovering Architecture Formalism of Geo-Located Web Services for Next Generation of Mobile Networks*", submitted to Intellcom conference, May 2005.
- [14] A.C Bayomock Linwa and S. Pierre, "*Concepts and Implementation of Geo-Located Web Service Architecture for Next Generation of Mobile Networks*", submitted to Mobile Information Systems, IOS Press May 2005.
- [15] C. Bettstetter, "*Toward Internet based Car Communications: On Some System Architecture and Protocol Aspects*", In Proceedings EUNICE 2000, Sixth EUNICE Open European Summer School, Twente, Netherlands, September 2000.
- [16] Bluetooth Forum, "*Bluetooth Specification Part E. Service Discovery Protocol (SDP)*", <http://www.bluetooth.com>, November 1999.
- [17] S. Bouchenak, D. Hagimont, S. Krakowiak, N. De Palma and F. Boyer. "*Experiences Implementing Efficient Java Thread Serialization, Mobility and Persistence*", INRIA Technical Report No. RR-4662, December 2002.
- [18] A.M Bravo, J.I Moreno and I. Soto, « *Advanced positioning and location based services in 4G mobile-IP radio access networks* », Personal, Indoor and Mobile Radio Communications, 2004. PIMRC 2004. 15th IEEE International Symposium on Volume 2, 5-8 Sept. 2004 Page(s):1085 - 1089 Vol.2.

- [19] T. Brinkhoff and J. Weitkämper, "Continuous queries within an architecture for querying XML-represented moving objects", C.S Jensen et al(Eds), SSTD 2001, LNCS 2020, pp. 136-154, 2001.
- [20] A.W. Brown, "*Large scale, component based development*", ISBN:013088720X , Prentice Hall, 2000.
- [21] D. Carlson, "*Modeling the UDDI Schema with UML*", CTO Ontogenics Corp.Boulder, Colorado, <http://XMLModeling.com>, 2001
- [22] P. Castro, B. Greenstein, R. Muntz, C. Bisdikian, P. Kermani, and M. Papadopouli, "*Locating Application Data Across Service Discovery Domains*", ACM SIGMOBILE MOBICOM 2001, 7th Annual Int. Conf. Mobile Computing and Networking, Rome, Italy, 2001.
- [23] A. Chakraborty, "*A Distributed Architecture for Mobile, Location-Dependent Applications*", master's thesis, Engineering in Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 2000.
- [24] H. Chen, *An intelligent broker architecture for context-aware systems*, PhD. dissertation proposal, Reference <http://umbc.edu/~hchen4/>, 2003.
- [25] S. Cheshire, "*Discovering Named Instances of Abstract Services using DNS*", IETF, Internet Draft draft-cheshire-dnsext-nias-00.txt, July 13, 2001, <http://ftp.ist.utl.pt/pub/drafts/draft-cheshire-dnsext-nias-00.txt>.
- [26] W. Chun-Te and M. Hsing, "*Location-Based-Service Roaming Based on Web Services*", Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on Volume 2, 25-30 March 2005 Page(s):277 - 280
- [27] S. Czerwinski, B. Y. Zhao, T. Hodes, A. Joseph, and R. Katz, "*An Architecture for a Secure Service Discovery Service*", fifth Annual International Conference on Mobile Computing and Networks (MobiCom '99), pp. 24-35, Seattle, WA, 1999.
- [28] C. Dabrowski, K. L. Mills and J. Elder, "*Understanding consistency maintenance in service discovery architectures during communication failure*", Workshop on Software and Performance 2002, pp. 168-178.
- [29] G. M. Djuknic and R. E. Richton, "*Geolocation and assisted-GPS*", IEEE Computer Society Press, pp. 123-125, 2001.

- [30] X. Dongyan, K. Nahrstedt and D. Wichadakul, "*QoS-aware discovery of wide-area distributed services*", Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium, pp 92–99, 15-18 May 2001.
- [31] R. Droms, "Automated Configuration of TCP/IP with DHCP". IEEE Internet Computing,, July 1999.
- [32] M.A Dru and S. Saada, "Location based mobile services: the essentials", Alcatel Telecommunications review, pp. 71-76, 1st quarter 2001.
- [33] M. H. Dunham A.Y Seydim and V. Kumar, "*An Architecture for Location Dependent Query Processing*", DEXA Workshop 2001, pp. 549-555, 2001.
- [34] M. H. Dunham and V. Kumar, "*Location Dependent Data and its Management in Mobile Databases*", Database and Expert Systems Applications 1998, Proceedings Ninth International Workshop, pp. 414 – 419, August 1998.
- [35] Ericsson, "*Mapping user selected QoS to RSVP and DiffServ Attributes*", <http://ing.ctit.utwente.nl/WU5/D5.16/pa5.pdf>, July 30, 2001.
- [36] Forum développeurs, Référence <http://xml.developpez.com/soap.htm>.
- [37] N. Gibbins and W. Hall, "*Scalability Issues for Query Routing Service Discovery*", Proceedings of the Second Workshop on infrastructure for Agents, MAS and Scalable MAS, 209-217, May 2001.
- [38] J. Govia and M. Barbeau, *Results of Comparing Bandwidth Usage and Latency: Service Location Protocol and Jini*, Workshop on Ad hoc Communications, held in conjunction with the Seventh European Conference on Computer Supported Cooperative Work (ECSCW 2001), Bonn, Germany, Reference <http://www.scs.carleton.ca/~barbeau/Publications/2001/WAHC/govea.pdf>, September 16-20, 2001.
- [39] E. Guttman, "*Service Location Protocol: Automatic Discovery of IP Network Services*", IEEE Internet Computing, pp 71-80, July 1999.
- [40] E. Guttman, C. Perkins, and J. Kempf, "*Service Templates and Service: Schemes*", IETF, RFC 2609, June, 1999, available at <http://www.faqs.org/rfcs/rfc2609.html>.
- [41] N. Harashima, T. Okoshi, J. Nakazawa, Y. Tobe, and H. Tokuda, "*AMRB: Toward Location Migration Transparency of Services*", IEEE International Conference on Parallel and Distributed Systems, pp. 305-314, ICPADS 2001.

- [42] F. J. Hauck, M. Van Steen and A. S. Tanenbaum, "*A location service for worldwide distributed objects*", Technical report, Dept. of Math. and Computer Science, Vrije Universiteit, Amsterdam, The Netherlands, <http://citeseer.nj.nec.com/hauck97location.html>, 1997.
- [43] P. Homburg, A.S Tanenbaum and M.R Van Steen, "*The Architecture of a Worldwide Distributed System*", PhD Thesis, Academisch Proefschrift, Globe Project, Reference <http://www.cs.vu.nl/~steen/globe/>, March 2001.
- [44] J. Hwang and P. Aravamudham , "*Middleware Services for P2P Computing in Wireless Grid Networks*", IEEE Internet Computing, July-August 2004.
- [45] B. Ip, "*3G Wireless Architecture UMTS vs CDMA2000*", Wireless and Mobile Network II, Columbia University, Reference <http://www.columbia.edu/itc/ee/e6951/2002spring/Projects/CVN/report14.pdf>, 2002.
- [46] jUDDI, <http://ws.apache.org/juddi/> .
- [47] G. Kollios, D. Gunopulos and V. J. Tsotras, "*On Indexing Mobile Objects*", Proceedings ACM Symposium on Principles of Database Systems, pp. 261-272, Philadelphia, 1999.
- [48] D.M Konidala, Y.Y Chan and K. Kwangjo, "*A secure and privacy enhanced protocol for location-based services in ubiquitous society*", Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE Volume 4, 29 Nov.-3 Dec. 2004 Page(s):2164 - 2168 Vol.4.
- [49] M. Koutsopoulou, A. Kaloxylos, A. Alonistioti, "*Charging, accounting and billing as a sophisticated and reconfigurable discrete service for next generation mobile networks*", Vehicular Technology Conference, 2002. Proceedings. VTC 2002-Fall. Vol. 4, IEEE 56th, 24-28 Sept. 2002.
- [50] S. Kyriazakos, D. Drakoulis and M. Theologou, "*Localization of Mobile Terminals, based on a hybrid Satellite-assisted and Network-based Techniques*", IEEE Wireless Communications and Networking Conference, WCNC 2000, Vol. 2, pp. 798 – 802, Chicago, September 2000.
- [51] A. Leonhardi and U. Kubach, "*An architecture for distributed universal location service*", Proceedings of the European Wireless '99 Conference, Munich, Germany, pp. 351-355, ITG Fachbericht, VDE Verlag, 1999.

- [52] Location Inter-operability Forum (LIF), "*Mobile Location Protocol*", LIF TS 101 Specification, Version 3.0.0 6, Reference <http://dan.greening.name/profession/manuscripts/LIF%20TS%20101%20v3.0.0.pdf>, June 2002.
- [53] P. Lucisnick, "*Bluetooth*", http://www.freebsd.org/doc/fr_FR.ISO8859-1/books/handbook/network-bluetooth.html
- [54] H. Maass, "*Location-aware mobile applications based on directory services*", In Proceedings of the Third Annual ACM/IEEE International Conference on Mobile Computing and Networking, pp. 23-33, ACM Press, Budapest Hungary, September 1997.
- [55] MapInfo Corporation, "*Developing Mobile aware services using Java and XML*", white paper, Reference <http://milocate.mapinfo.com/mobile/papers/LocationJavaXML.pdf>, August 2001.
- [56] C. Mascolo, L. Capra and W. Emmerich, "*Mobile Computing Middleware*", Advanced Lectures on Networking, ISBN 3-540-00165-4, pp 20-58, 2002.
- [57] F. Michahelles, M. Samulowitz and B. Schiele, "*Detecting Context in Distributed Sensor Networks by Using Smart Context-Aware Packets*". In International Conference on Architecture of Computing Systems, Reference <http://www.vision.ethz.ch/publ/arcs02.html>, ARCS 2002.
- [58] B. Miller and R. Pascoe, "*Mapping Salutation Architecture APIs to Bluetooth Service Discovery Layer*", <http://www.bluetooth.com>, July 1999.
- [59] P. Mougin and C. Barriolade, "*Web Services, Business Objects and Component Models*", Orchestra Networks, White Paper, July 2001, http://www.xml.org/xml/feature_articles/orchestra_webservice.pdf.
- [60] D. Mountain and J. Raper, "*Designing geolocation services for next generation mobile phone systems*", AGI 2000, <http://www.soi.city.ac.uk/~dmm/research/pubs/W2.4.pdf>, 27 Sept 2000.
- [61] K. Nahrstedt, D. Xu, D. Wichadakul, and B. Li, "*QoS-aware middleware for ubiquitous and heterogeneous environments*", IEEE Communications Magazine, vol. 39, no. 11, November 2001 pp. 140-148.
- [62] NS-2, <http://www.isi.edu/nsnam/ns/>.
- [63] OGC, "*OpenLS protocol*" <http://www.opengeospatial.org/functional/?page=ols>

- [64] S. Panagiotakis and A. Alonistioti, *"Intelligent service mediation for supporting advanced location and mobility aware service provisioning in reconfigurable mobile networks"*, IEEE Wireless Communications Magazine, October 2002.
- [65] C. E. Perkins, *"Dynamic Host Configuration Protocol for IPv6 (DHCPv6)"*, Internet Draft, draft-ietf-dhc-dhcpv6-14.txt, May 2000.
- [66] C. E. Perkins and E. Guttman, *"DHCP Options for Service Location Protocol. Internet"*, RFC 2610, June 1999.
- [67] PeterIndia, *"Introduction to SOAP"*, SOAP Simple Object Access Protocol, <http://www.peterindia.net/SOAPOverview.html>.
- [68] D. Pfoer, C. S. Jensen and Y. Theodoridis, *"Novel Approaches to the Indexing of Moving Object Trajectories"*, Proceedings 26th International Conference on Very Large Databases, pp. 395-406, Egypt, 2000.
- [69] D. Porcino, *"Performance of a OTDOA-IPDL positioning receiver for 3GPP-FDD mode"*, 3G Mobile Communication Technologies, Second International Conference, pp. 221 – 225, 26-28 March 2001.
- [70] G. Pospischil, H. Kunczler and A. Kuchar, *"LoL@: a UMTS location based service"*, Reference, Proceedings International Symposium on 3G Infrastructure and Services, Athens, http://lola.ftw.at/homepage/content/a40material/LoLa_a_location_based_service.pdf, July 2001.
- [71] R. Raman, M. Livny, and M. Solomon, *"Matchmaking: Distributed Resource Management for High Throughput Computing"*, Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, Chicago, IL, 1998.
- [72] Q. Ren and M. H. Dunham, *"Using semantic caching to manage location dependent data in mobile computing"*, In MobiCom'00: The Sixth Annual International Conference on Mobile Computing and Networking, pp. 210-221, August 2000.
- [73] Salutation Consortium, *"Salutation Architecture: Overview"*, White Paper, <http://www.salutation.org/whitepaper/originalwp.pdf>, 1998.
- [74] A. Y. Seydim, M. H. Dunham, and V. Kumar, *"An architecture for location dependent query processing"*, Proceedings in 12th International Conference on Database and Expert System Applications , pp. 549-555, DEXA 2001.

- [75] SFR, «*SFR lance la première offre 3G grand public en France*», <http://www.sfr.fr/FR/rdv/actualites/actualite09.jsp>, 16 Juin 2004.
- [76] C. Shahabi, M. R. Kolahdouzan, S. Thakkar,, J. L. Ambitey and C. A. Knoblocky, “*Efficiently Querying Moving Objects with Predefined Paths in a Distributed Environment*”, Geographic Informations System Proceedings of the ninth ACM international symposium on Advances in geographic information systems, pp. 34-40, ISBN 1-58113-443-6, 2001.
- [77] T.Shimoe and T. Sano, “*IMT-2000 Network Architecture*”, Fujitsu science journal 38.2, pp. 134-139, December 2002.
- [78] A.P Silva and G.R Mateus, “*Location-based taxi service in wireless communication environment*”, Simulation Symposium, 2003. 36th Annual 30 March-2 April 2003 Page(s):47 – 54.
- [79] SOAP, “*Simple Object Access Protocol Specifications V1.2*”, 24 June 2003, <http://www.w3.org/TR/2003/REC-soap12-part0-20030624/#L1149>.
- [80] SoapTest, <http://www.parasoft.com/jsp/products/home.jsp>
- [81] T. Sollazzo, S. Handshuh, S. Staab, M. Frank and N. Stojanovic, “*Semantic Web Service Architecture-Evolving Web service toward the Semantic Web*”, InstituteAIFB, American Association for Artificial Intelligence, FLAIRS Conference 2002, pp. 425-429, Reference <http://www.informatik.uni-trier.de/~ley/db/conf/flairs/flairs2002.html#SollazzoHSFS02>.
- [82] T.S Stamoulakatos and E.D Sykas, “*A network architecture to obtain traffic information for location based service applications*”, Telecommunications, 2003. ConTEL 2003. Proceedings of the 7th International Conference on Volume 1, 2003 Page(s):197 – 203.
- [83] D. Steer and D. Fauconnier, “*Location Services Architecture for future mobile networks*”, Vehicular Technology Conference Proceedings, VTC 2000-Spring Tokyo, 2000 IEEE 51st, Vol.2, pp. 1362 – 1366, May 2000.
- [84] D. H. Stojanovic and S. J. Djordjevic-Kajan, “*Developing location based services from GIS perspective*”, In 5th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Service (TELSIKS 2001), vol. 2, pp. 459-462, 2001.
- [85] Sun Microsystems, “*Jini Architectural Overview*”, Technical White Paper, <http://www.sun.com/jini/>, 1999.

- [86] Sun Microsystems, "*Message Queue 3.5 Administration Guide*", <http://docs-pdf.sun.com/817-6024/817-6024.pdf>, 2004.
- [87] G. Swedberg, "*Ericsson's Mobile Location Solution*", Ericsson Review, Issue 4, pp. 214-221, 1999.
- [88] S. Takino, "*GIS on the fly, to realize wireless GIS Network by Java Mobile Phone*", Second International Conference On Web Informations Systems Engineering, Vol. 2, pp. 76-81, 2001.
- [89] Y. Theodoridis, T. Sellis, A. N. Papadopoulos, and Y. Manolopoulos, "*Specifications for efficient indexing in spatiotemporal databases*", In International Conference on Scientific and Statistical Database Management, pp.123-132, 1998.
- [90] TSAC, "*Update on IMT-2000 Development*", <http://www.ofta.gov.hk/ad-comm/tsac/ts-paper/ts2000p21.pdf>, TSAC Paper No. 21/2000.
- [91] Universal Plug and Play Forum, "*Universal Plug and Play Device Architecture*", Version 0.91, March 2000.
- [92] R. Unni and R. Harmon, "*Location-based services: models for strategy development in M-commerce*", Management of Engineering and Technology, 2003. PICMET '03. Technology Management for Reshaping the World. Portland International Conference on 20-24 July 2003 Page(s):416 – 424.
- [93] UTM, <http://193.251.89.13/fnrsec/UTM.htm>
- [94] Valco Electronics, "*What is GPS and how does it work?*", Reference <http://www.valcoelectronics.com/Tips/gps.htm>, 2003.
- [95] M. Vazirgiannis, Y. Theodoridis and T. Sellis, "*Spatio-Temporal Composition and Indexing for Large Multimedia Applications*", ACM Multimedia Systems, Vol. 6, No. 4, pp. 284-298, November 1998.
- [96] B. S Vinnakota and M. D Mickunas, "*Enhancing location management in smart spaces*". Technical Report UIUCDCS-R-2001-2216, UIUC, May 2001.
- [97] J.N Weatherall and A. Hopper, "*Predator : a distributed location service and example applications*", Proceedings of Cooperative Buildings 1999, Springer-Verlag Lecture Notes in Computer Science, Reference <http://www.uk.research.att.com/pub/docs/att/tr.1999.5.pdf>, September 1999.

- [98] O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha, "*Updating and querying databases that track mobile units*", Distributed and Parallel Databases, pp. 257-287, 1999.
- [99] W. Xiaotao and H. Schulzrinne, "*Location-based services in internet telephony*", Consumer Communications and Networking Conference, 2005. CCNC. 2005 Second IEEE 3-6 Jan. 2005 Page(s):331 – 336.
- [100] Y. Yaron, Goland, T. Cai, P. Leach, Y. Gu, and S. Albright, "*Simple Service Discovery*", Protocol. Internet Draft, draft-cai-ssdp-v1-03.txt, October 1999.
- [101] J. Ye, J. Hou and S. Papavassiliou, "*A comprehensive resource management framework for next generation wireless networks*", IEEE Transactions on mobile computing Vol1, No.4, pp. 249-264, 2002.
- [102] Y. Zhao, "*Mobile Phone Location Determination and Its Impact on Intelligent Transportation Systems*", Intelligent Transportation Systems, pp 55 – 64, IEEE Transactions, Volume 1, Issue 1, March 2000.
- [103] F. Zhu, M. Mutka and L. Ni, "*Classification of Service Discovery in Pervasive Computing Environments*", Michigan State University, MSU-CSE-02-24, 2002, <http://www.cse.msu.edu/~zhufeng/ServiceDiscoverySurvey.pdf>.
- [104] F. Zhu, M. Mutka and L. Ni, "*Splendor: A Secure, Private, and Location-aware Service Discovery Protocol Supporting Mobile Services*", Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom'03), pp 235-242, March 2003.

Annexe

```

public void addNode(String nodeKey, String url, String type, double X, double Y, double Z, String
Map_Addr){
    try{
        Node noeudCourant=new Node(ConnectionUddiDB.getConnection());
        if (noeudRacine!=null && type!= "racine" ){
            noeudCourant.create(nodeKey,url,type,X, Y, Z,Map_Addr);
        }
        else if(noeudRacine==null && type == "racine" ){
            noeudCourant.create(nodeKey,url,type,X, Y, Z,Map_Addr);
        }
        else {
            System.out.println("Le noeud racine est déjà créé. Changer de type...");
        }
    }catch(Exception e){
        e.printStackTrace();
    }
}

public static Node getRootNode() {
    return noeudRacine;
}

public void removeNode(String nodeKey) {
    try{
        AgreementNode agreement=new AgreementNode(ConnectionUddiDB.getConnection());
        Node noeudCourant=new Node(ConnectionUddiDB.getConnection());
        if (agreement.findAllAgreementForNoeud(nodeKey)==null){
            noeudCourant=noeudCourant.findByPrimaryKey(nodeKey);
            noeudCourant.remove(nodeKey);
        }
    }catch(Exception e){
        e.printStackTrace();
    }
}

public Node findNode(String nodeKey) {.... }

```

Figure A.1 Extrait de l'implémentation de TopologyManager

```

public void createAgreement(String agreementKey, String serviceKey, String systCoord, String
adminKey) {
    try{
        Agreement agreementCourant=new Agreement(ConnectionUddiDB.getConnection());
        agreementCourant.create(agreementKey,systCoord, adminKey,serviceKey);
    }catch(Exception e){
        e.printStackTrace();
    }
}

```

```

    }
}

public void createAgreementNode(String agreementKey, String nodeKey, String urlApp, double
costService, double BW_min, double BW_max, double PPD_min, double PPD_max, int
SDU_smax) {
    try{
        AgreementNode agreementNoeudCourant=new
        AgreementNode(ConnectionUddiDB.getConnection());
        agreementNoeudCourant.create(agreementKey,nodeKey,
urlApp,costService,BW_min,BW_max,PPD_min,PPD_max,SDU_smax);
    }catch(Exception e){
        e.printStackTrace();
    }
}

public void removeAgreement(String agreementID) {
    try{
        Agreement agreementCourant=new Agreement(ConnectionUddiDB.getConnection());
        AgreementNode agreementNoeudCourant=new AgreementNode(ConnectionUddiDB.getConnection());
        agreementNoeudCourant.remove(agreementID);
        agreementCourant.remove(agreementID);
    }catch(Exception e){
        e.printStackTrace();
    }
}

public void removeAgreementNode(String agreementID, String nodeID) {
    try{
        AgreementNode agreementNoeudCourant=new AgreementNode(ConnectionUddiDB.getConnection());
        agreementNoeudCourant.remove(agreementID, nodeID);
    }catch(Exception e){
        e.printStackTrace();
    }
}

public Agreement findAgreement(String agreementKey) {
    Agreement agreement=null;
    try{
        Agreement agreementCourant=new Agreement(ConnectionUddiDB.getConnection());
        agreement= agreementCourant.findByPrimaryKey(agreementKey);
    }catch(Exception e){
        e.printStackTrace();
    }
    return agreement;
}

public static AgreementManager getInstance()
{
    if (instance==null) {
        instance=new AgreementManager();
    }
    return instance;
}

public ArrayList loadAllNodeForAgreement(String agreementID) {

```

```

    ArrayList listeNoeud=null;
    try{
        AgreementNode agreementNoeudCourant=new AgreementNode(ConnectionUddiDB.getConnection());
        listeNoeud=(ArrayList) agreementNoeudCourant.findAllNodeForAgreement(agreementID);
    }catch(Exception e){
        e.printStackTrace();
    }
    return listeNoeud;
}

```

Figure A.2 Extrait de l'implémentation de AgreementManager

```

public String findNearestService(String msid, String serviceId) {
    String urlApp=null;
    Node noeud=new Node(ConnectionDB.getConnection());
    Agreement agreement= new Agreement(ConnectionDB.getConnection());
    AgreementNode agreementNode = new AgreementNode(ConnectionDB.getConnection());
    String agreementId="";
    try{
        LocationManager gst=LocationManager.getLocationManager();
        gst.setConnection(gst.getConnFact().getConnection("aUser","aPwd"));
        ObjetMobile obj= gst.locateMsid(msid) ;
        if (obj !=null){
            agreementId=agreement.findAgreementByServiceID(serviceId).getId();
            Node myNoeud= noeud.findNearestNode(agreementId, obj);
            if (myNoeud !=null) {
                urlApp = agreementNode.findByPrimaryKey(agreementId,myNoeud.getNoeudID()).getUrlApp();
                MigrationManager migrate= new MigrationManager();
                migrate.initSender("queueSuivi"+
myNoeud.getNoeudID(),"queueConnectionFactory"+myNoeud.getNoeudID());
                migrate.suiviMsid(serviceId,msid);
            }
        }
    }catch(Exception e){
        e.printStackTrace();
    }
    return urlApp;
}

public String findNearestServiceQoS(String msid, String serviceId, double rcout, double rbw,
double ruf) {
    String theUrlApp=null;
    Node noeud=new Node(ConnectionDB.getConnection());
    Agreement agreement= new Agreement(ConnectionDB.getConnection());
    String agreementId="";
    ArrayList noeudTrouves=new ArrayList();
    ServiceCache serCac=null;
    QoSQuotationBD quot=null;
    try{

```

```

LocationManager gst=LocationManager.getLocationManager();
gst.setConnection(gst.getConnFact().getConnection("aUser","aPwd"));
ObjetMobile obj= gst.locateMsid(msid) ;
if (obj !=null){
    agreementId=agrement.findAgrementByServiceID(serviceId).getId();
    noeud=noeud.findNearestNode(agreementId ,obj);
    theUrlApp=verifySelectionCondition(agreementId, serviceId, noeud, rcout, rbw, ruf);
    if (theUrlApp !=null) {
        MigrationManager migrate= new MigrationManager();
        migrate.initSender("queueSuivi"+
noeud.getNoeudID(),"queueConnectionFactory"+noeud.getNoeudID());
        migrate.suiviMsidQoS(serviceId,msid, theUrlApp, new Double(rcout), new Double(rbw), new
Double(ruf));
    }
}
} catch (Exception e){
    e.printStackTrace();
}
return theUrlApp;
}

public void createService(AuthInfo authInfo, BusinessService businessService, String agreementId,
String adminId, String systCoord, ArrayList agreementNodeList){
    DataStore datastore = DataStoreFactory.getDataStore();
    try
    {
        datastore.beginTransaction();
        // validate authentication parameters
        Publisher publisher = getPublisher(authInfo,dataStore);
        String publisherID = publisher.getPublisherID();
        String businessKey = businessService.getBusinessKey();
        String serviceKey = businessService.getServiceKey();
        // Confirm that 'publisherID' controls the BusinessEntity that this
        // BusinessService belongs to. If not then throw a UserMismatchException.
        if (!dataStore.isBusinessPublisher(businessKey,publisherID))
            throw new UserMismatchException("businessKey="+serviceKey);
        saveService(businessService,datastore);
        createAgreementNode (agreementId, adminId, systCoord, agreementNodeList);
        initSender("publication"+serviceKey,"queueConnectionFactory"+serviceKey);
        createMessageSendPublication(businessService,agreementId,adminId,systCoord,agreementNodeList);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void createMessageSendPublication(BusinessService businessService, String agreementId,
String adminId, String systCoord, ArrayList agreementNodeList){
    ObjectMessage message=null;
    try {
        Hashtable params=new Hashtable();
        params.put("businessService", businessService);
        params.put("agreementId", agreementId);
        params.put("adminId",adminId);

```

```

        params.put("systCoord", systCoord);
        params.put("agreementNodeList ", agreementNodeList);
        message=createMessage(params);
        sendMessage(message);
    } catch (Exception e) {
        System.out.println("Exception occurred: " +
            e.toString());
    }
}

public void onMessage(Message msg) {
    ObjectMessage message=null;
    try{
        message=(ObjectMessage) msg;
        Hashtable params= (Hashtable) message.getObject();
        BusinessService businessService= (BusinessService) params.get("businessService");
        String agreementId = (String) params.get("agreementId");
        String adminId=(String) params.get("adminId");
        String systCoord= (float) params.get("systCoord");
        ArrayList agreementNodeList= (ArrayList) params.get("agreementNodeList");
        createServiceAndAgreementNode(businessService,agreementId,adminId,systCoord,agreementNodeList);
    }catch (JMSEException e) {
        System.out.println("Exception occurred: " +
            e.toString());
    }
}

```

Figure A.3 Extrait de l'implémentation de DvtServiceManager

```

public void onMessage(Message msg) {
    ObjectMessage message=null;
    String msid = null;
    String serviceId=null;
    String urlAppCur=null;
    String agreementId=null;
    double rcout=0.0;
    double rbw=0.0;
    double ruf=0.0;
    boolean drapeau=false;
    Agreement agreement=new Agreement();
    AgreementNode agreementNoeud=new AgreementNode();
    ArrayList agreementList=null;
    try{
        message=(ObjectMessage) msg;
        Hashtable params= (Hashtable) message.getObject();
        if (params.size()==3) {
            msid = (String) params.get("msid");
            serviceId = (String) params.get("serviceFrsId");
            urlAppCur= (String) params.get("urlApp");
            agreementId=agreement.findAgreementByServiceId(serviceId).getAgreementId();
            agreementNextNode=agreementNoeud.findByPrimaryKey(agreementId,nextNode.getNodeId());

```

```

    }
    if (params.size()==5){
        msid = (String) params.get("msid");
        serviceId = (String) params.get("serviceFrsId");
        urlAppCur= (String) params.get("urlApp");
        rcout=((Double)params.get("rcoutService")).doubleValue();
        rbw=((Double)params.get("rbw")).doubleValue();
        ruf=((Double)params.get("ruf")).doubleValue();
        listeQoSRequest.add(new QoSRequest(msid,rcout,rbw,ruf));
        agreementId=agreement.findAgreementByServiceId(serviceId).getAgreementId();
    }
    listeMsid.add(msid);
    while (!drapeau){
        drapeau=ifExit(serviceId,msid,noeudId);
    }
    if (nextNode !=null && params.size()==5) {

agreementList=agreementNoeud.findAllByNodeIdByAgreementId(agreementId,nextNode.getNodeId());
        for (int i=0; i<agreementList.size();i++){
            agreementNextNode=(AgreementNode) agreementList.get(i);
            QoSQuotation quot=((ServiceCache)
DvtServiceManager.mapServiceCache.get(serviceId)).findQosQuotation(nextNode.getNodeId(),
serviceId, agreementNextNode.getUrlApp());
            //verifier la condition migration
            if (agreementNextNode.getCoutService()<=rcout && quot.getBW()>=rbw && quot.getUF()>=ruf)
                break; }
            notifyExit(msid,urlAppCur,agreementNextNode.getUrlApp());
            initSender("queueSuivi"+
nextNode.getNodeId(),"queueConnectionFactory"+nextNode.getNodeId());
            trackMsidQoS(serviceId,msid, urlAppNew, new Double(rcout), new Double(rbw), new
Double(ruf));
        }
        if (nextNode !=null && params.size()==3){
            notifyExit(msid,urlAppCur,agreementNextNode.getUrlApp());

initSender("queueSuivi"+nextNode.getNodeId(),"queueConnectionFactory"+nextNode.getNodeId());
            trackMsid(serviceId,msid,agreementNextNode.getUrlApp());
            fermerConnection();
        }
        removeMsid(msid);
    }catch (JMSEException e) {
        System.out.println("Exception occurred: " +
            e.toString());
    }
}
}

public boolean ifExit(String serviceId, String msid, String noeudIdCur) {
    Agreement agreement=null;
    AgreementNode agreementNoeud=null;
    Spatial currentObject=null;
    boolean flag=false;
    try{

```

```

LocationManager lm=LocationManager.getLocationManager();
agrement=new Agreement();
agrementNoeud=new AgreementNode();
Node currentNode =new Node();
currentNode = currentNode.findByPrimaryKey(noeudIdCur);
currentObject=lm.locateMsid(msid);
nextNode=findNode(currentObject);
if(nextNode !=null && nextNode!=currentNode){
String agreementId=agrement.findAgreementByServiceId(serviceId).getAgreementId();
agrementNextNode=agrementNoeud.findByPrimaryKey(agreementId,nextNode.getNodeId());
if (agrementNextNode != null){
    flag=true;
}
}
}
}
} catch(Exception e){
    e.printStackTrace();
}
return flag;
}

public String notifyExit(String msid, String urlAppAnc, String urlAppNew) {
try {
    Service service = new Service();
    Call call = (Call) service.createCall();
    call.setTargetEndpointAddress( new java.net.URL(urlAppAnc) );
    call.setOperationName(new QName("MigrationNotification", "migrateTo"));
    String rep= (String) call.invoke( new Object[] {msid,urlAppNew} );
} catch (Exception e) {
    System.err.println(e.toString());
}
return rep;
}
}

```

Figure A.4 Extrait de l'implémentation de MigrationManager

```

public void ifEntry(String subject, ArrayList zone,String urlApp,LocationRequestContext
requestContext)
{
    LocationResult result=null;
    ArrayList mobileFound=null;
    boolean flag=false;
    try{
        result=locateSubject(subject,requestContext);
        ArrayList locations=(ArrayList) result.getLocations();
        for (int i=0; i<locations.size(); i++){ if(((LocationData)locations.get(i)).getLocationStatus() ==
LocationData.SUCCESSFUL_LOCATION){
            LocatedMobileStation lms =(LocatedMobileStation)locations.get(i);
            Spatial spatial = getSpatialData(lms);
            flag=isInsideCoveredArea(zone, spatial);

```

```

        if(flag==true){            notifyEntry(((LocationData)locations.get(i)).getMobileStationId(),urlApp);
        }
    }
}
} catch (Exception e){
    e.printStackTrace();
}
}

public void notifyEntry(String msid, String urlApp) {
    try {
        Service service = new Service();
        Call call = (Call) service.createCall();
        call.setTargetEndpointAddress( new java.net.URL(urlApp) );
        call.setOperationName(new QName("MigrationNotification", "pushAnnounceEntry"));
        String rep= (String) call.invoke( new Object[] {msid} );
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}

public LocationResult locateSubject(String subject, LocationRequestContext requestContext) {
    Theme theme=null;
    ThemeMsid themeMsid =null;
    com.ericsson.snf.mps.Connection conn=null;
    LocationResult result=null;
    CommunicationException communicationexception = null;
    LocationProtocol locationprotocol = null;
    java.net.URL urlList[] = factory.getURLs();
    java.net.URL urlLCS=null;
    try{
        themeMsid = new ThemeMsid();
        theme = new Theme();
        ArrayList msidList=(ArrayList)
        themeMsid.findAllMsid(theme.findThemeBySubject(subject).getThemeId());
        if (msidList.size()!=0){
            LocationTarget[] target=new LocationTarget[msidList.size()];
            for (int i=0; i<msidList.size(); i++){
                target[i]=new MobileStation(msidList.get(i).toString());
            }
            if (urlList[0] !=null) urlLCS=urlList[0];
            if (urlList[1] !=null) urlLCS=urlList[1];
            HttpURLConnection httpurlconnection = locationprotocol.openConnection(urlLCS);
            result = locationprotocol.getLocation(httpurlconnection, myUser, myPwd, target, requestContext);
            httpurlconnection.disconnect();
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally{
        try{
            conn.close();
        } catch (LocationException lex){
            System.out.println(lex.getMessage());
        }
    }
}

```

```

    }
    }
    return result;
}

public ArrayList locateSubjectInZone(String subject, ArrayList zone, LocationRequestContext
requestContext){
    LocationResult result=null;
    ArrayList mobileFound=null;
    boolean flag=false;
    try{
        result=locateSubject(subject,requestContext);
        ArrayList locations=(ArrayList) result.getLocations();
        for (int i=0; i<locations.size(); i++){ if(((LocationData)locations.get(i)).getLocationStatus()
==LocationData.SUCCESSFUL_LOCATION){
            LocatedMobileStation lms =(LocatedMobileStation)locations.get(i);
            Spatial spatial = getSpatialData(lms);
            flag=isInsideCoveredArea(zone, spatial);
        }
        if(flag==true){ mobileFound.add(((LocationData)locations.get(i)).getMobileStationId());
        }
    }
    } catch(Exception e){
        e.printStackTrace();
    }
    return mobileFound;
}

private boolean isInsideCoveredArea(ArrayList lineSegments, Spatial spatial){
    boolean flag =false;
    int j=0;
    try{
        if (lineSegments.size()>=3) flag =true;
        while (j<lineSegments.size() && flag==true) {
            PolygonLineSegment lineSegment = (PolygonLineSegment) lineSegments.get(i);
            double xCoeff=(lineSegment.getY2()-lineSegment.getY1())/(lineSegment.getX2()-
lineSegment.getX1());
            double leftValue=spatial.getY()-xCoeff*spatial.getX();
            double rightValue=lineSegment.getY1()-xCoeff*lineSegment.getX1();
            if (lineSegment.getSign()=="<=") {
                if (leftValue<=rightValue) j=j+1;
            }
            else {
                flag=false;
                break;
            }
        }
        if (lineSegment.getSign()==">=") {
            if (leftValue>=rightValue) j=j+1;
        }
        else {
            flag=false;
            break;
        }
    }
}

```

```

    }
    }catch(Exception e){
        e.printStackTrace();
    }
    return flag;
}

public java.util.ArrayList locateSubjectNearOf (String subject, Spatial position, float
precision,LocationRequestContext requestContext) {
    LocationResult result=null;
    ArrayList mobileTrouves=null;
    try{
        result=locateSubject(subject,requestContext);
        ArrayList locations=(ArrayList) result.getLocations();
        for (int i=0; i<locations.size(); i++){
            if(((LocationData)locations.get(i)).getLocationStatus()
==LocationData.SUCCESSFUL_LOCATION){
                LocatedMobileStation lms =(LocatedMobileStation)locations.get(i);
                Spatial spatial = getSpatialData(lms);
                if(distance(spatial.getX(),spatial.getY(),spatial.getZ(), position)<=precision){
                    mobileTrouves.add(((LocationData)locations.get(i)).getMobileStationId());
                }
            }
        }
    }catch(Exception e){
        e.printStackTrace();
    }
    return mobileTrouves;
}

public ArrayList collocate(String msid, String subject, float precision,LocationRequestContext
requestContext) {
    LocationResult result=null;
    Spatial spatial=null;
    ArrayList mobileTrouves=null;
    try{
        MobileStation ms=new MobileStation(msid);
        LocationTarget[] target=new MobileStation[]{new MobileStation(msid)};
        result=getLocation(target,requestContext);
        LocationData location =result.getLocationData(ms);
        if(location.getLocationStatus() ==LocationData.SUCCESSFUL_LOCATION){
            LocatedMobileStation lms =(LocatedMobileStation)location;
            spatial = getSpatialData(lms);
        }
        result=locateSubject(subject,requestContext);
        ArrayList locations=(ArrayList) result.getLocations();
        for (int i=0; i<locations.size(); i++){
            if(((LocationData)locations.get(i)).getLocationStatus()
==LocationData.SUCCESSFUL_LOCATION && spatial != null){
                LocatedMobileStation lms =(LocatedMobileStation)locations.get(i);
                Spatial pos = getSpatialData(lms);
                if(distance(pos.getX(),pos.getY(),pos.getZ(),
spatial.getX(),spatial.getY(),spatial.getZ())<=precision){

```

```
        mobileTrouves.add(((LocationData)locations.get(i)).getMobileStationId());  
    }  
    }  
    }  
    }catch(Exception e){  
        e.printStackTrace();  
    }  
    return mobileTrouves;  
}
```

Figure A.5 Extrait des méthodes de localisation thématique proposées