2m11. 3058.3

Université de Montréal

ARCHITECTURE DE COMMUNICATION GÉNÉRIQUE POUR UN SYSTÈME DE BUS DU COMMERCE ÉLECTRONIQUE

par

Siham Tagmouti

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maîtrise (M.Sc.)
en informatique et recherche opérationnelle

Avril, 2003

©Siham Tagmouti, 2003

QA
76
.J54
2003
v.029

Université de Montréal

Direction des bibliothèques

AVIS

L'auteur a autorisé l'Université de Montréal à reproduire et diffuser, en totalité ou en partie, par quelque moyen que ce soit et sur quelque support que ce soit, et exclusivement à des fins non lucratives d'enseignement et de recherche, des copies de ce mémoire ou de cette thèse.

L'auteur et les coauteurs le cas échéant conservent la propriété du droit d'auteur et des droits moraux qui protègent ce document. Ni la thèse ou le mémoire, ni des extraits substantiels de ce document, ne doivent être imprimés ou autrement reproduits sans l'autorisation de l'auteur.

Afin de se conformer à la Loi canadienne sur la protection des renseignements personnels, quelques formulaires secondaires, coordonnées ou signatures intégrées au texte ont pu être enlevés de ce document. Bien que cela ait pu affecter la pagination, il n'y a aucun contenu manquant.

NOTICE

The author of this thesis or dissertation has granted a nonexclusive license allowing Université de Montréal to reproduce and publish the document, in part or in whole, and in any format, solely for noncommercial educational and research purposes.

The author and co-authors if applicable retain copyright ownership and moral rights in this document. Neither the whole thesis or dissertation, nor substantial extracts from it, may be printed or otherwise reproduced without the author's permission.

In compliance with the Canadian Privacy Act some supporting forms, contact information or signatures may have been removed from the document. While this may affect the document page count, it does not represent any loss of content from the document.

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé :

ARCHITECTURE DE COMMUNICATION GÉNÉRIQUE POUR UN SYSTÈME DE BUS DU COMMERCE ÉLECTRONIQUE

présenté par :

Siham Tagmouti

a été évalué par un jury composé des personnes suivantes :

Guy Lapalme	président-rapporteur
Peter Kropf	directeur de recherche
Gibert Babin	codirecteur de recherche
Esma Aïmeur	membre du jury

Mémoire accepté le : 28 juillet 2003

Sommaire

Dans ce mémoire, nous présentons une architecture de communication générique pour un système de bus du commerce électronique.

Nous nous proposons d'étudier WOS (Web Operating System), un système virtuel qui supporte et gère les processus distribués sur Internet.

Nous analysons en détail l'implémentation courante de la couche de communication de WOS, et en particulier les deux protocoles de communication WOSP (WOS Protocol) et WOSRP (WOS Request Protocol).

L'objectif de ce travail est de rendre l'infrastructure de WOS plus fléxible et meilleur pour la gestion des problèmes de sécurité.

Afin de réaliser cela, nous changeons l'implémentation actuelle de WOSP et WOSRP par une nouvelle implémentation conforme aux standards actuels d'Internet (HTTP, MIME et XML). Pour se faire nous avons suggéré une nouvelle architecture de ces deux protocoles afin qu'ils puissent supporter les deux versions d'implémentation de la couche communication de WOS, à savoir l'originale et celle utilisant HTTP, MIME et XML. Par la suite nous avons effectué des tests de communications entre deux nœuds WOS.

Mots-clés : Web Operating System, Protocoles de communication, WOS Protocol, WOS Request Protocol, Hyper Transfer Text Protocol, Multipurpose Internet Mail Extention, Extensible Markup Language, systèmes distribués.

Abstract

The Web Operating System (WOS) is a virtual system which supports and manages distributed processes over the Internet. WOS relies on a communication layer based on a service discovery protocol (WOSRP) and a generic service protocol (WOSP) to achieve communications between nodes.

The main objective of the work presented in this thesis is to investigate and propose a generic communication architecture for WOS in order to enhance the WOS infrastructure for more flexibility and better handling and management of security issues.

We first proceed by a thorough and comprehensive analysis of the current implementation of the communication layer, and more particularly the two communication protocols WOS Protocol (WOSP) and WOS Request Protocol (WOSRP). The main contribution of this work consists of a proposal for a new communication layer that supports Internet standards. In this proposal, the original implementations of WOSP and WOSRP are changed to new ones that make use of the standards HTTP, MIME and XML. The newly proposed architecture supports both the original implementation of WOSP and WOSRP and the new implementations that are based on HTTP, MIME and XML. The two implementations were used, for test purposes, to communicate between two WOS nodes and the results of the tests are reported in the thesis.

Keywords : Web Operating System, Communication Protocols, WOS Protocol, WOS Request Protocol, Hyper Transfer Text Protocol, Multipurpose Internet Mail Extention, Extensible Markup Language, distributed systems.

Table des matières

Table des figures

Liste des tableaux

À mes parents.

À mon mari.

À mes sœurs.

À mon frère.

À ma belle famille.

À toute ma famille.

Remerciements

Je tiens tout d'abord à exprimer mes remerciements à mes directeurs de recherche le professeur Peter Kropf et le professeur Gilbert Babin pour leur aide, leur patience, leur disponibilité et leur support constant tout au long de ce travail.

C'est avec un grand plaisir que j'exprime toute ma gratitude à mes parents, Assia et Khalid, à mes sœurs, Mariam, Soloua, Bouchra et Youssra et à mon frère Tahair qui m'ont encouragé tout au long de mes études et auxquels je dois chaque réussite.

J'aimerais remercier mon mari Ali pour sa présence, sa compréhension et son soutien permanents qui m'ont permis de mener à bien mes travaux.

Je remercie aussi mesdames et messieurs les membres du jury d'avoir bien accepté d'évaluer ce travail.

Enfin que tous celles et ceux qui m'ont apporté leur appui trouve ici l'expression de ma gratitude.

Chapitre 1

Introduction

1.1 Motivation du travail

Le développement rapide des réseaux ainsi que l'hétérogénéité du Web, à savoir la diversité des structures de communication, la nature des réseaux de transmission et les différentes architectures d'applications rendent de plus en plus complexes l'utilisation et le partage des ressources.

En effet, les applications réparties constituent un domaine en expansion, qui touche tous les secteurs d'activités : communication et prise de décision, gestion industrielle et financière, publication, diffusion de l'information, santé, transport et loisir. De plus, les utilisateurs de systèmes d'informations ont de plus en plus besoin d'échanger des données qui peuvent être de nature différente et sous des formats qui le sont tout autant. Toutes ces informations peuvent être dans des endroits différents, autrement dit, sur des postes distants, sur des réseaux distincts ou encore sur l'Internet.

Afin de répondre à ces besoins, plusieurs applications réparties ont vu le jour. Parmis ces projets on peut citer : JXTA, Napster, Gnutella, SOAP et WOS (Web Operating System) qui constitue notre sujet de recherche [21]. L'approche de WOS a été conçue pour supporter et gérer les processus distribués sur Internet. WOS fournit aux utilisateurs la possibilité de formuler des requêtes dans le but d'exploiter un certain nombre de ressources sur le réseau. WOS est constitué d'un ensemble de nœuds qui communiquent à l'aide de protocoles de communication. Chaque nœud WOS inclut les fonctionnalités d'un serveur et d'un client. En effet, si un nœud WOS est capable de répondre à une requête formulée par un client ,il joue dans ce cas le

rôle d'un serveur ; sinon il interroge les nœuds WOS voisins, dans ce cas il joue le rôle d'un client, pour avoir une réponse à la requête originale.

Chaque nœud WOS utilise son propre entrepôt (base d'informations) pour stocker et mettre à jour les informations sur le nœud ainsi que sur les ressources et services disponibles.

La communication entre les différents nœuds de WOS se fait à l'aide de deux protocoles WOSP (WOS Protocol) et WOSRP (WOS Request Protocol) [11]. Le protocole WOSP fournit un support de communication pour les classes de services tandis que le protocole WOSRP permet de localiser les classes de services et d'encapsuler les messages WOSP.

1.2 Objectifs de recherche

WOS utilise l'infrastructure actuelle du Web pour communiquer. Afin de le rendre plus compatible et plus facile d'utilisation à travers le Web, nous avons proposé d'utiliser des standards d'Internet tel que HTTP, MIME et XML dans la couche de communication de WOS. En effet le but de ce travail est de remplacer la structure actuelle des protocoles WOSP et WOSRP, qui utilise présentement une implémentation propre à WOS, par une autre conforme aux standards actuels d'Internet.

Pour atteindre cet objectif une étude approfondie de la structure et du fonctionnement de la couche communication, en particulier des deux protocoles WOSP et WOSRP, est nécessaire. À partir de cette étude nous allons identifier en premier lieu les changements qu'il faut apporter aux deux protocoles pour qu'ils puissent supporter et gérer les deux implémentations de WOS, autrement dit, l'ancienne et la nouvelle qui utilise les langages standards du Web.

Après avoir identifié ces changement, nous allons proposer une nouvelle architecture de WOSP/WOSRP afin d'assurer la communication entre deux nœuds WOS distants. Ensuite nous allons réaliser sa mise en œuvre en utilisant HTTP, MIME et XML. Finalement nous allons expérimenter son fonctionnement ainsi que sa compatibilité avec l'ancienne implémentation de WOS.

1.3 Plan du mémoire

Ce mémoire est constitué de sept chapitres. Dans le chapitre 2 nous présentons les technologies pair-à-pair (P2P) et les services Web. Par la suite nous décrirons des exemples de systèmes utilisant ces technologies de communication et de partage d'informations et de ressources. Nous abordons plus particulièrement dans ce chapitre les approches Napster, Gnutella, JXTA, Corba, Sun ONE, .NET et SOAP.

Dans le chapitre 3 nous expliquerons le fonctionnement de WOS, en particulier nous donnerons une description de la couche de communication de ce dernier. Nous présentons ensuite la structure actuelle des deux protocoles WOSP et WOSRP. Finalement nous détaillerons les différents scénarios de communication entre deux nœuds WOS. Ceci nous permettra de mieux comprendre les protocoles WOSP et WOSRP.

Dans le chapitre 4 nous proposons la nouvelle structure de WOSP et WOSRP en utilisant les standards HTTP, MIME et XML. Cette nouvelle structure permettra à WOS de supporter les implémentations multiples de WOSP et WOSRP.

Dans le chapitre 5, nous réalisons une analysé détaillée de l'implémentation courante de la couche de communication de WOS. Cette analyse nous permettra de définir les interfaces nécessaires au développement de plusieurs implémentations de WOSP et WOSRP.

Dans le chapitre 6 nous présentons la méthode d'implémentation de cette nouvelle version, ainsi que les différents tests de communication effectués. Enfin, une conclusion est présentée au chapitre 7, faisant une synthèse des résultats obtenus et des recherches futures possibles dans ce domaine.

Chapitre 2

État de l'art

Avec l'extension du Web plusieurs applications indépendantes et développées avec des langages différents sont apparues. Afin de permettre à ces composantes Web hétérogènes de communiquer et de partager des ressources, de nouveaux systèmes ont vu le jour.

Ces types de systèmes permettent aux utilisateurs de localiser et partager les services et les ressources à travers le Web, et cela indépendamment des langages et des systèmes d'exploitation utilisés. Quelques exemples de ces systèmes seront présentés plus tard dans ce chapitre.

Dans un premier lieu nous présenterons les technologies P2P [2] ainsi que quelques modèles qui les supportent, en particulier JXTA [4] [16], Napster [1] et Gnutella [18] [15].

Ensuite nous présenterons CORBA [5] [8] et les Services Web en l'occurrence SunOne [26], .NET [17] [13], et SOAP [27]. Finalement nous exposerons une comparaison entre l'approche WOS et ces modèles.

2.1 Les technologies Pair-à-Pair

La plupart des services Internet utilisent le modèle traditionnel client/serveur. Dans cette architecture le client envoie une requête au serveur auquel il est connecté afin d'obtenir une ressource spécifique. Une fois les ressources disponibles, le serveur envoie une réponse au client. Dans cette architecture le client à un rôle passif, il est seulement capable de demander un service, il ne peut ni répondre ni faire de

traitement.

Avec le modèle P2P [2] la notion de client et serveur disparaît. Chaque nœud est capable d'initialiser un dialogue et de répondre à une requête. L'avantage majeur de cette architecture est le partage de responsabilité entre les nœuds.

Dans le modèle P2P deux types d'architecture existent, l'architecture hybride qui est centralisée et l'architecture native qui est décentralisée.

2.1.1 Architecture hybride

Afin d'échanger les ressources, les utilisateurs doivent s'enregistrer dans un annuaire central, se trouvant dans un serveur, pour déclarer les éléments qu'ils désirent partager, ainsi que l'endroit du transfert. La figure 2.1 montre un exemple de communication P2P pour une architecture hybride [2].

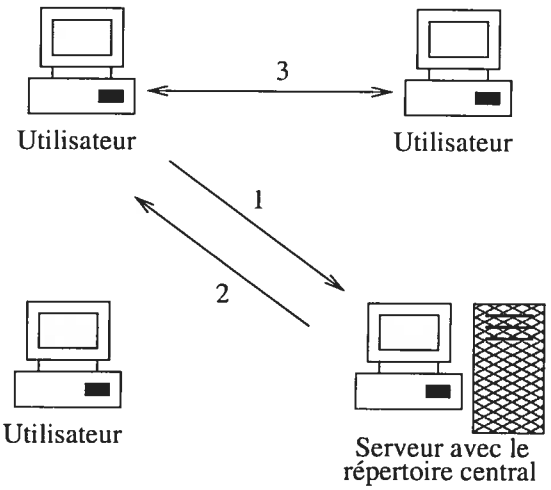


FIG. 2.1 – Scénario de communication dans le cas de l'architecture hybride

Les étapes de cette communication (figure 2.1) se présentent comme suit :

1. L'utilisateur envoie une requête au serveur central pour demander une ressource.
2. Le serveur central envoie sa réponse qui comporte la liste des ordinateurs qui peuvent offrir cette ressource.
3. L'utilisateur utilise la ressource directement à partir d'un des ordinateurs indiqués par le serveur.

2.1.2 Architecture native

Contrairement à l'architecture hybride, l'architecture native [2] n'a pas besoin d'ordinateur central. Tous les ordinateurs qui veulent appartenir à un groupe de communication doivent fournir leurs adresses IP au logiciel P2P afin de devenir visible par rapport au groupe de communication. Tous les ordinateurs doivent utiliser le même logiciel P2P.

Un ordinateur qui désir obtenir un fichier, envoie une requête à tous les autres ordinateurs connus par son logiciel. Si aucun de ces ordinateurs ne contient le fichier, alors ces derniers transmettent la requête à d'autres ordinateurs qui leurs sont connus. Une fois le fichier trouvé, son emplacement sera envoyé au demandeur qui pourra alors effectuer directement le transfert.

La figure 2.2 montre un exemple de communication P2P pour une architecture native.

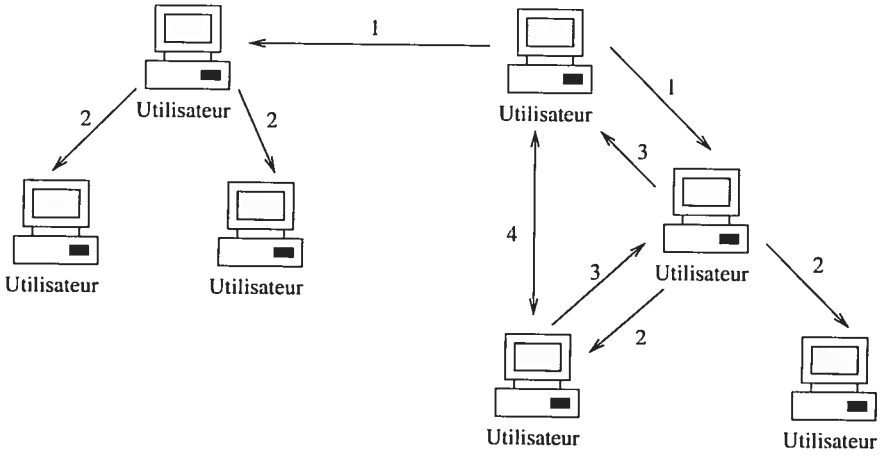


FIG. 2.2 – Scénario de communication dans le cas de l'architecture native

Les étapes de cette communication (figure 2.2)se présentent comme suit :

1. L'utilisateur qui cherche un fichier ressource transmet une requête aux différents utilisateurs connus.
2. Les utilisateurs étant incapables de répondre à cette requête, la renvoient à d'autres utilisateurs qu'ils connaissent.
3. L'utilisateur qui dispose du fichier ressource répond en suivant le chemin inverse.
4. Le transfert du fichier ressource se fait directement entre l'ordinateur qui dispose du fichier et celui qui le réclame.

Plusieurs nouvelles technologies se basant sur le modèle P2P sont apparues sur le marché. En particulier nous présentons JXTA, Napster et Gnutella.

2.1.3 JXTA

Le projet JXTA [4] [16] a été lancé par Sun Microsystèmes. Il se compose d'un ensemble de protocoles. Chaque protocole est défini par un ou plusieurs messages échangés entre les participants dans la communication. Chaque message à un format défini et peut contenir des données différentes.

JXTA v1.0 se constitue de six protocoles de base qui définissent les éléments primaires pour une communication P2P :

- **Le protocole de découverte PDP (Peer Discovery Protocol)** : permet aux nœuds faire connaître leurs ressources et de découvrir les services ainsi que les autres nœuds du réseau.
- **Le protocole d'invocation de service PRP (Peer Resolve Protocol)** : permet aux nœuds d'envoyer et de traiter des requêtes.
- **Le protocole de Rendez-vous RVP (Rendezvous Protocol)** : produit le détail de la propagation des messages entre les nœuds.
- **Le protocole d'information PIP (Peer Information Protocol)** : permet aux nœuds d'obtenir des informations sur les autres nœuds sur le réseau.
- **Le protocole de communication par canaux PBP (Pipe Binding Protocol)** : fournit les mécanismes nécessaires pour créer un canal de communication virtuel entre un ou plusieurs pairs.
- **Le protocole de routage ERP (Endpoint Routing Protocol)** : fournit les mécanismes nécessaires pour déterminer le chemin d'accès à une destination finale.

Les protocoles JXTA sont indépendants des systèmes d'exploitations, des langages de développement et du réseau de transport utilisé. Chaque protocole est indépendant des autres. De ce fait, chaque nœud peut implémenter un sous-ensemble de protocoles selon le besoin.

La figure 2.3 montre l'architecture de JXTA :

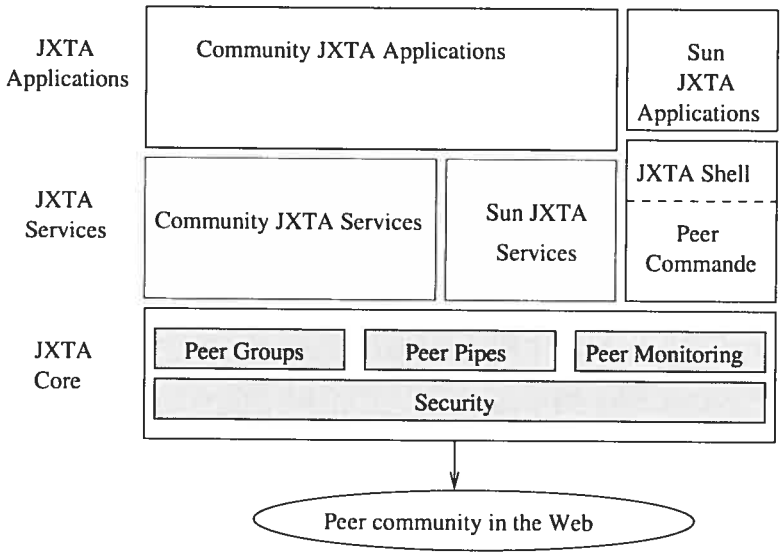


FIG. 2.3 – Le modèle JXTA

JXTA est constitué de trois niveaux :

- **Le noyau de JXTA (JXTA Core)** : il contient les six principaux protocoles qui sont désignés comme des services du noyau JXTA. Il comporte les éléments suivants :
  - Peer Groups : permet la création et la gestion des groupes de nœuds.
  - Peer Pipes : initialise les canaux de communication entre les nœuds afin de partager les informations sur le réseau de manière distribuée.
  - Peer Monitoring : permet le contrôle du comportement et de l'activité des nœuds à l'intérieur d'un groupe.
  - Sécurité (Security) : garantit la confidentialité, l'identité et l'accès contrôlé aux services.
- **Les services de JXTA (JXTA Services)** : cette couche fournit les services réseau désirables mais qui ne sont pas nécessaires pour la solution P2P. Ces services implémentent les fonctionnalités qui peuvent être incorporées dans les applications P2P. Par exemple : les mécanismes de recherche, de partage et d'indexage. Cette couche permet de développer les capacités du noyau et de faciliter le développement des applications.
- **Les applications de JXTA (JXTA Applications)** : Cette couche permet

aux utilisateurs d'accéder aux services P2P de la couche services et du noyau. Ces applications sont construites en utilisant les services JXTA ainsi que le noyau. Le «JXTA Shell» est une application qui permet l'utilisation d'une interface usager minimale, c'est la raison pour laquelle il est séparé des applications et des services.

2.1.4 Napster

Napster [1] utilise le modèle hybride du P2P. Il s'appuie sur des serveurs centralisés pour répondre aux fonctionnalités de recherche. Les clients se connectent aux serveurs centraux et rapportent les fichiers qu'ils partagent. Les serveurs ajoutent ces fichiers à leurs listes de recherche. Une fois qu'un client émet une requête de recherche à un serveur, ce dernier envoie une réponse constituée seulement des noms de fichiers trouvés et leurs expéditeurs. Le transfert des fichiers se fait directement entre les deux nœuds impliqués. Autrement dit le serveur ne se charge pas de l'enregistrement ou du transfert des fichiers, il ne fait qu'informer les clients de l'endroit où ils peuvent trouver les fichiers recherchés. Le protocole de transport utilisé dans ce modèle est TCP.

Le protocole Napster est bien défini, et il existe plusieurs implémentations aussi bien pour le client que pour le serveur.

L'avantage de ce type de système est l'efficacité des méthodes de recherche et la facilité d'utilisation. Par contre son inconvénient majeur est sa dépendance totale au serveur central. Si celui-ci tombe en panne, c'est tout le service qui tombe en panne. De plus, aucun anonymat n'est garanti, étant donné que l'on doit s'enregistrer. Le gestionnaire du serveur central a la possibilité de construire des fichiers clients ou des profils d'utilisateurs.

2.1.5 Gnutella

Gnutella [18] [15] utilise l'architecture native du P2P. Dans chaque nœud du réseau Gnutella, un programme s'exécute jouant le rôle de serveur et de client en même temps. Il permet donc, d'envoyer des requêtes et des réponses simultanément. Les nœuds du réseau Gnutella sont connectés en utilisant les protocoles TCP et IP, et

les transferts de fichiers se font en utilisant HTTP. Gnutella fonctionne de la façon suivante : Chaque ordinateur voulant se connecter au réseau doit d'abord trouver un autre ordinateur sur le réseau et à travers lequel il peut se connecter. La liste des machines connectées au réseau se trouve dans des nœuds spécifiques appelés «Host-Cache». Une fois connecté au réseau un message de type «Ping/Pong» est envoyé pour connaître la taille du réseau ainsi que les autres machines qui s'y sont connectées. Les messages de type «Ping» représentent des requêtes envoyées à tous les éléments du réseau sauf l'expéditeur. Les messages de type «Pong» représentent des réponses qui empruntent le même chemin.

Au début de Gnutella tous les nœuds étaient pareils et jouaient le même rôle. Ceci a produit une saturation causée par la grande quantité des messages circulant sur le réseau, ainsi qu'un ralentissement du réseau causé par le manque de ressources de quelques nœuds pour faire transiter rapidement les données à travers le réseau.

Pour remédier à ce problème et améliorer la fluidité du réseau, le nouveau concept d'«ULTRAPEERS» a été utilisé. Ce concept consiste à diviser les nœuds du réseau en deux catégories, ceux qui ont une grande capacité de calcul et de transfert appelés «ULTRAPEERS» et ceux qui sont normaux (qui ont une petite capacité) appelés «Les Feuilles».

Pour qu'un ordinateur soit déclaré comme «ULTRAPEERS», il doit avoir une grande bande passante et un processeur puissant, il ne doit pas être derrière un mur coupe-feu. De plus il doit utiliser un système d'exploitation récent et être sur le réseau Gnutella depuis assez longtemps.

Les avantages de ce système sont : sa grande souplesse, sa grande robustesse et l'anonymat (relatif) qu'il assure, car il n'y a pas de serveur qui stocke des informations relatives aux utilisateurs. Par contre ses inconvénients sont : la gestion de la bande passante n'est pas toujours optimale. De plus, l'anonymat peut permettre de contourner les problèmes des droits d'auteur, voir d'échanger des données illégales. Les moyens pour contrôler ce qui transite par le réseau sont presque inexistants.

2.2 CORBA

CORBA (Common Object Request Broker Architecture) [5] [8] fournit une infrastructure logicielle de communication pour les applications distribuées. Elle permet aux applications de communiquer et d'utiliser des composantes hétérogènes de façon transparente.

Chaque application peut exporter des services sous la forme d'objets CORBA. Les interactions entre les applications se font par des invocations à distance des méthodes et des objets. Les éléments intervenant dans une invocation d'objets entre une application client et une application serveur (figure 2.4) sont :

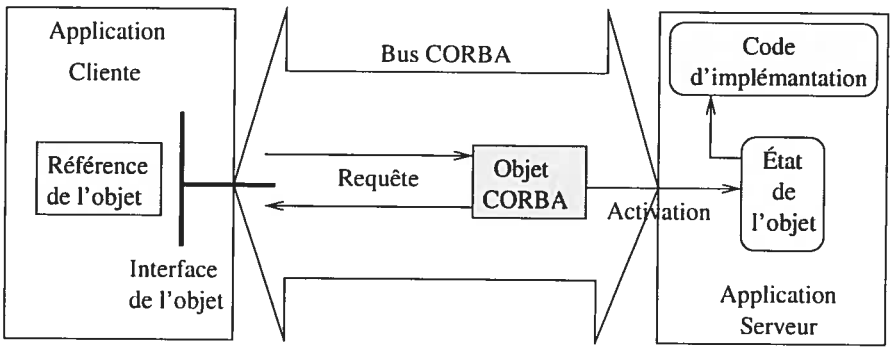


FIG. 2.4 – Les composants intervenant dans une invocation d'objet à travers le bus CORBA

- **Application client** : c'est un programme capable d'invoquer les méthodes et les objets à travers le bus CORBA.
- **Référence d'objet** : c'est la structure désignant l'objet CORBA et contenant l'information nécessaire pour le localiser sur un bus CORBA donné. La référence d'objet est utilisée pour optimiser la quantité de données à transférer.
- **Interface de l'objet** : c'est le type abstrait de l'objet CORBA qui définit ses attributs et les opérations possibles sur cet objet.
- **Requête** : c'est le mécanisme qui permet d'invoquer une opération ou d'accéder à un attribut de l'objet.
- **Bus CORBA** : c'est ce qu'on appelle aussi l'ORB (Object Request Broker). Il permet d'assurer le lien entre l'application client et l'application serveur d'une façon transparente.
- **Objet CORBA** : c'est une entité virtuelle gérée par le bus CORBA.

- **Activation** : c'est le processus d'association d'un objet d'implantation quelconque à un objet CORBA.
- **Code d'implantation** : c'est un code qui regroupe les traitements associés à l'implantation des opérations de l'objet CORBA.
- **Application serveur** : c'est la structure qui contient les objets et les exécutions des opérations.

Pour invoquer un service, le client doit connaître seulement le nom de l'objet serveur, son type et l'opération à invoquer. L'ensemble des problèmes liés à la localisation et à la disponibilité des ressources est géré par l'ORB. En effet l'ORB est un intermédiaire qui permet aux objets de dialoguer d'une manière transparante. Il a les caractéristiques suivantes :

- Il permet la liaison avec la plupart des langages de programmation, tel que C, C++, SmallTalk, Ada, COBOL et Java.
- Il permet une invocation de façon à ce que les requêtes aux objets semblent toujours être locales. En effet le bus CORBA se charge d'acheminer ces requêtes en utilisant le canal de communication le plus approprié. Il utilise aussi une invocation statique et dynamique, dans le premier cas (statique), les invocations sont contrôlées à la compilation. Tandis que dans le second cas (dynamique), les invocations sont contrôlées à l'exécution.
- Les interfaces des objets sont connues de l'ORB et sont accessibles par les programmes.
- Il permet une activation automatique et transparente des objets. Ces derniers sont stockés en mémoire uniquement s'ils sont utilisés par des applications clientes.
- Il permet l'interopérabilité entre bus. À partir de la norme CORBA 2.0, un protocole générique de transport des requêtes (GIOP ou General Inter-ORB Protocol) a été défini permettant l'interconnexion de différents bus CORBA.

2.3 Les Services Web

Les services Web fournissent des moyens de communication standards pour des logiciels et applications fonctionnant sur des plates-formes différentes. Ils permettent

ainsi l'interactivité transparente entre différents systèmes complètement hétérogènes.

Les services Web permettent non seulement le développement rapide de nouvelles applications réparties, mais aussi la réingénierie «re-engineering» des applications existantes. En effet, les services Web offrent le moyen de réutiliser les différents composants déjà programmés, en faisant abstraction du langage de programmation utilisé. Ce qui réduit les efforts de développement.

Dans ce qui suit nous allons présenter plusieurs types d'architectures des services Web qui sont présentement bien définies et utilisés dans différents domaines de l'industrie.

2.3.1 Sun ONE

Sun ONE (Sun Open Net Environment) [26] a été défini par Sun. Il représente une architecture logicielle ouverte supportant les services à la demande, et permettant ainsi aux utilisateurs de créer et d'utiliser ce type de services. L'architecture Sun ONE répond aux problèmes de confidentialité, de sécurité et d'identification. L'objectif de cette plate-forme est de fournir un environnement afin de mieux développer, rechercher et exploiter les ressources. La figure 2.5 montre les éléments de base de l'infrastructure de Sun ONE :

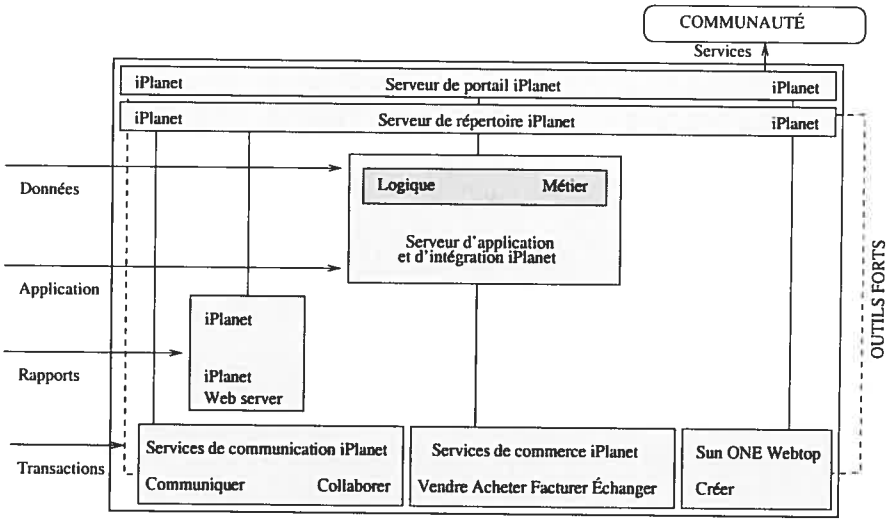


FIG. 2.5 – Les éléments de base de l'infrastructure Sun ONE

La plate-forme Sun One se constitue des éléments suivants :

- **Serveur de répertoire iPlanet** : il contient toute l'information nécessaire sur les ressources et la communauté. Il permet de faire l'intermédiaire dans les connexions allant des ressources à la communauté.
- **Serveur de portail iPlanet** : il permet aux utilisateurs appartenant à une communauté bien définie de se connecter à l'entreprise. Le «serveur de portail iPlanet» est intégré avec l'infrastructure du «serveur de répertoire iPlanet» pour faciliter l'authentification des utilisateurs et rassembler les informations relatives aux membres du groupe.
- **Serveur d'application et d'intégration** : il permet d'obtenir un meilleur niveau de performance, de fiabilité et d'évolution des produits basés sur les standards ouverts.
- **Serveur Web iPlanet** : il offre un support pour les services Web publics tout en permettant d'augmenter leur niveau de performance et de fiabilité.
- **Service de communication iPlanet** : il assure la communication interpersonnelle, et comprend «le serveur de messagerie iPlanet» et «le serveur de calendrier iPlanet». Le premier offre un support pour la messagerie électronique ainsi qu'une intégration transparente avec le client de messagerie «iPlanet Portal Server». Le deuxième fournit des agendas électroniques personnels et professionnels. Il permet aussi de programmer des événements à l'échelle de l'entreprise et des groupes afin de mieux gérer le partage des ressources.
- **Service de commerce iPlanet** : il permet les échanges entre machines. Il regroupe les opérations d'achat, de vente, d'information et de facturation réalisées au sein de l'architecture de Sun ONE.
- **Sun One WebTop** : il fournit un environnement intégré pour les langages de programmation tel que Java, C et C++. Il permet aux développeurs d'accéder de manière transparente aux différents modules, accélérant ainsi le développement.

2.3.2 .NET

L'environnement .NET [17] [13] est conçu pour les serveurs afin de pouvoir mieux communiquer avec de nombreux terminaux grâce à SOAP et XML. En s'appuyant sur le «.NET Framework», des composants d'un logiciel (par exemple Microsoft Word) pourrait être localisés dans différents serveurs et manipulés par un grand nombre de

terminaux. Pour mieux définir .NET nous allons voir quelques services offerts par cette plate-forme :

- Il offre une personnalisation du réseau Internet. Chaque utilisateur aura un espace d'hébergement pour ses données, ses applications et ses préférences. Les utilisateurs ne posséderont plus de logiciels, mais ils les loueront et en auront accès en ligne. Du fait que toutes les données seront dans un espace privé sur le Web, alors l'utilisateur aura un accès en permanence, où qu'il soit, à ses données et à ses applications.
- Il offre à l'utilisateur la possibilité d'interagir avec un ensemble d'équipements et de services. Ces derniers seront capables d'échanger et de combiner les objets et les données pour fournir les informations désirées.

Parmis les éléments essentiels qui composent l'environnement .NET, on peut citer :

- C#, un nouveau langage orienté objet qui intègre des éléments de C, C++ et JAVA. Il est destiné à faciliter la programmation dans .NET et apporte quelques innovations telles les métas-données.
- Un environnement d'exécution commun (Common Language Runtime - CLR) qui exécute du «bytecode» écrit dans un langage intermédiaire (Microsoft Intermediate Language MSIL ou IL). Ainsi du code et des objets écrits dans un langage quelconque peuvent être compilés en IL et exécutés par le CLR (si un compilateur IL existe pour ces langages).
- Une grande bibliothèque de composants qui fournit une grande variété de fonctions pour écrire des programmes accessibles par le CLR.
- ASP.NET, une nouvelle version d'ASP (Active Server Pages) qui supporte une compilation en IL. On peut également écrire les pages ASP dans n'importe quel langage disposant d'un compilateur IL.
- Visual Studio.NET, qui constitue une réadaptation de l'environnement Visual Studio et de Visual InterDev permettant ainsi le développement d'applications, de composants classiques et de composants WEB.
- WinForms et WebForms, un ensemble de composants graphiques accessibles dans Visual Studio .NET.
- ADO.NET, une nouvelle génération de composants d'accès aux bases de données

ADO qui utilise XML et SOAP pour l'échange de données.

- Un support pour les terminaux mobiles avec une version compacte de l'environnement .NET.

2.3.3 SOAP

SOAP [27] est un protocole permettant des appels de procédure à distance (RPC) et s'appuyant principalement sur le protocole HTTP et sur XML. SOAP simplifie l'accès aux objets en donnant aux applications la possibilité d'invoquer des méthodes ou des fonctions, résidant sur les serveurs distants. Une application SOAP crée un bloc de requêtes XML qui fournit à la fois les données nécessaires à la fonction distante et la localisation de l'objet distant lui-même. Le message SOAP est basé sur XML et contient les parties suivantes 2.6 :

- **L'enveloppe** : c'est le conteneur du plus haut niveau représentant le message.
- **L'en-tête** : c'est le conteneur générique pour ajouter des dispositifs à un message SOAP d'une façon générique. SOAP définit des attributs pour indiquer qui devrait traiter le dispositif et si la compréhension est facultative ou obligatoire.
- **Le corps** : c'est le conteneur des informations du message.

Parmis les avantages de SOAP ont peut distinguer :

- SOAP permet une interopérabilité avec divers environnements quelle que soit la plate-forme d'exécution ;
- SOAP est un protocole basé sur les standards d'Internet HTTP et XML ;
- SOAP offre une grande simplicité technologique.

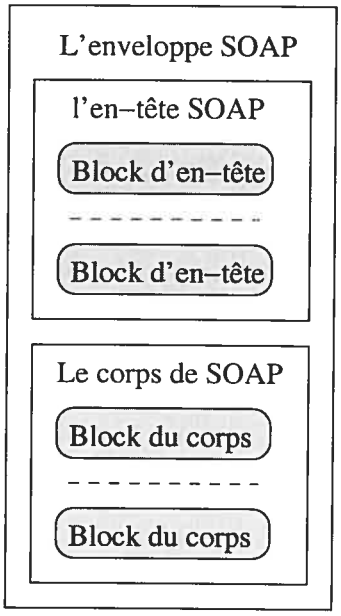


FIG. 2.6 – La structure du message SOAP

2.4 Classement des différents systèmes décrits

Dans cette section nous allons classer les différents systèmes vus dans ce chapitre. Cette classification sera faite à l'aide d'un graphe (2.7). Ensuite nous allons comparer WOS aux deux approches : P2P et services Web.

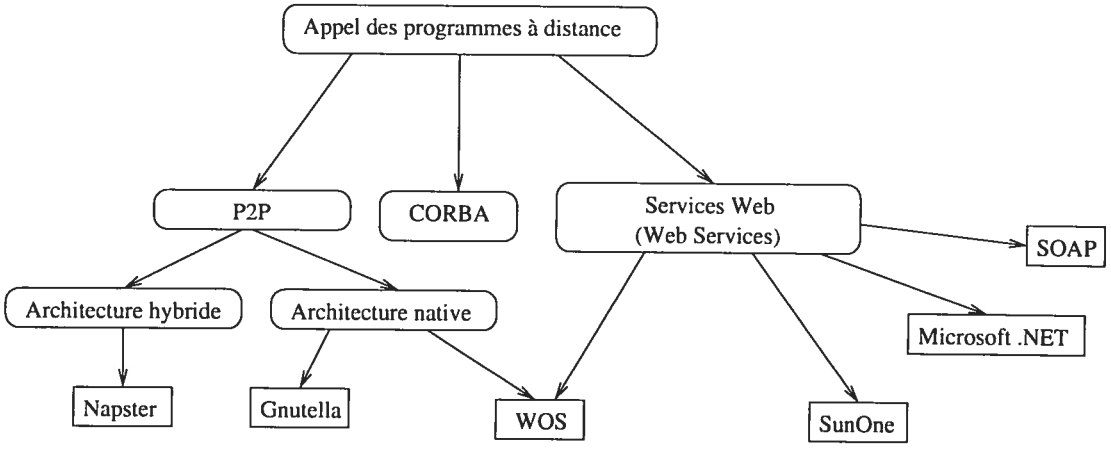


FIG. 2.7 – Classification des systèmes distribués présentés dans ce chapitre

Comme nous montre la figure 2.7, nous avons placé WOS avec les systèmes P2P et les services Web. En effet WOS peut être considéré comme un système P2P d'ar-

chitecture native, puisque chaque nœud WOS peut être client et serveur au même temps. Aussi WOS n'a pas de serveur central, chaque nœud WOS stocke lui-même les informations nécessaires sur les autres nœuds ainsi que les services disponibles.

WOS peut aussi être considéré comme un système de services Web, puisqu'il donne aux utilisateurs les moyens nécessaires pour invoquer des services de façon transparente.

2.5 Conclusion

Dans ce chapitre, nous avons donné un large éventail des différents types de systèmes qui ont été réalisés auparavant permettant aux utilisateurs de localiser et partager des services et des ressources à travers le Web. Nous avons présenté en détail le fonctionnement de quelques exemples de ces systèmes.

Dans un premier lieu nous avons présenté les technologies P2P ainsi que quelques modèles qui les supportent. En particulier JXTA, Napster et Gnutella. Ensuite nous avons présenté CORBA ainsi que les Services Web, en l'occurrence SunOne, .NET et SOAP. Finalement nous avons fait une comparaison entre l'approche WOS et ces modèles.

L'étude de ces différents systèmes qui sont maintenant disponibles et bien définis nous permettra d'avoir une meilleure compréhension des standards Web utilisés pour représenter les messages et les envoyer. Cette étude nous permettra ainsi de prendre une décision quand à la nouvelle implémentation de la couche de communication de WOS.

Chapitre 3

Description de WOSP et WOSRP

Dans ce chapitre, nous donnons une brève présentation du fonctionnement de WOS ainsi qu'une description détaillée de la couche de communication de WOS. Ensuite nous analysons la structure actuelle des deux protocoles WOSP et WOSRP. Finalement nous expliquerons les différents scénarios de communication possibles entre deux nœuds WOS.

3.1 Présentation du fonctionnement de WOS

WOS est un système virtuel qui supporte et gère les processus distribués et parallèles sur Internet. Ce système est constitué de plusieurs versions qui peuvent échanger des informations avec des requêtes particulières. WOS fonctionne de la manière suivante [11] :

- Une requête est émise par un utilisateur pour exécuter un programme particulier ou pour initialiser des services qui peuvent être localisés aux différents sites du réseau ;
- Le nœud qui reçoit la requête peut décider s'il a les ressources nécessaires pour répondre, sinon il l'envoie à d'autres nœuds qu'il connaît, et ainsi de suite jusqu'à ce qu'un nœud puisse répondre.

Pour la recherche d'une ressource, les nœuds liés directement au nœud émetteur sont contactés. S'ils ne possèdent pas l'information, ils transmettent la requête à d'autres nœuds qui leur sont connus. Cette recherche est appliquée à deux niveaux [11] :

- Au niveau du serveur : les serveurs WOS sont versionnés. Ainsi, le processus de

recherche doit identifier en premier les serveurs qui supportent la même version de WOS que le client.

- Au niveau des ressources : une fois les serveurs identifiés, le processus de recherche doit localiser les ressources dépendamment des besoins des clients.

En effet, la version du serveur WOS doit être identifiée avant que les ressources soient localisées, ce qui implique l'utilisation de deux protocoles différents qui sont WOSRP et WOSP, présentés dans la section 2.2 de ce chapitre.

L'architecture d'un nœud WOS est illustrée dans la figure 3.1 [9].

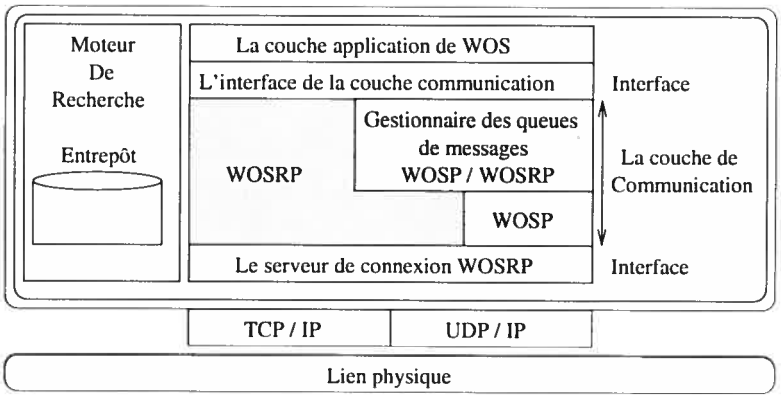


FIG. 3.1 – l'architecture d'un nœud WOS.

Le nœud WOS est constitué des éléments suivants [9] [21] :

- **La couche application de WOS** : permet à l'usager d'utiliser les services offerts par WOS.
- **L'interface de la couche communication** : elle assure le lien entre la couche application de WOS et la couche de communication, permettant ainsi d'invoquer toutes les méthodes de la couche de communication.
- **Le gestionnaire des queues de messages WOSP/WOSRP** : il permet de créer et de gérer les queues de messages. Ces dernières contiennent tous les types de messages WOSP et WOSRP, elles sont présentées en détail dans la section 3.3.1 de ce chapitre.
- **La couche de communication** : elle contient toutes les fonctionnalités pour recevoir et envoyer les messages.
- **Le serveur de connexion WOSRP** : il permet de faire le pont entre la couche de communication de WOS et les services de la famille TCP/IP.

- **Entrepôt** : il apporte au nœud WOS les informations et les composantes nécessaires pour formuler des requêtes de services. Le nœud utilise son entrepôt pour stocker et mettre à jour les informations sur les autres nœuds ainsi que sur les ressources et les services disponibles.

3.2 Présentation de WOSP et de WOSRP

3.2.1 WOSP

Le protocole WOSP [9] [11] permet de localiser et d'utiliser les ressources distribuées à travers WOS. Il définit des mécanismes ainsi qu'un langage de syntaxe générique pour deux nœuds WOS afin qu'ils puissent échanger des messages.

Les mécanismes génériques supportent deux modes de communication. Le mode sans connexion, qui ne nécessite pas une connexion WOSP préalable pour envoyer des messages. Et le mode avec connexion, où un canal de communication est établi entre les deux nœuds WOS avant d'envoyer les messages.

Chaque nœud WOS connaît une ou plusieurs versions de WOSP. Chaque version de WOSP doit être identifiée de façon unique. Pour cela, une chaîne de 448 octets est utilisée.

Un message WOSP a une structure simple (figure 3.2 [9]). Il peut être de type réponse ou commande. Un message WOSP est construit des éléments suivants :

- **Commande (command)** : chaque commande est identifiée par un nom et un identificateur. L'identificateur se compose de l'identifiant du message WOSP, la position de la commande dans ce message et le nombre de données (data) et de méta-données (metadata) décrivant le message. Il existe trois types de commande :
  - **La commande d'exécution (execute)** : elle permet au client WOS d'utiliser les ressources et services des autres nœuds.
  - **La commande d'initialisation (setup)** : elle est utilisée pour échanger les paramètres d'exécution d'un service ou d'utilisation d'une ressource disponible sur le WOS.
  - **La commande de requête (query)** : elle est utilisée par le nœud client WOS pour interroger les autres nœuds WOS.

- **réponse (reply)** : elle constitue une réponse à une commande reçue, et elle contient une référence à cette commande.
- **donnée (data)** : elle représente les données qui peuvent êtres attachées à une commande ou à une réponse.
- **méta-donnée (metadata)** : elle est utilisée pour décrire les commandes et les données.

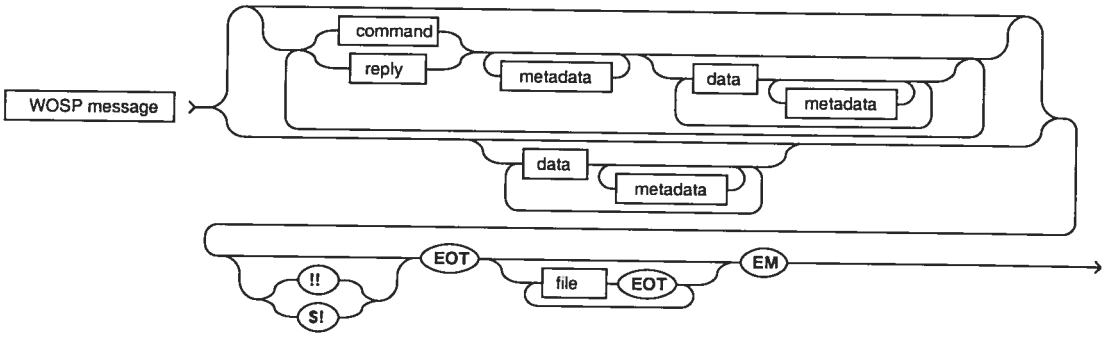


FIG. 3.2 – Syntaxe générique de WOSP

Pour communiquer, les applications composent les messages WOSP à l'aide d'une entité appelée «triplet». Un triplet est constitué de trois champs : Le type du triplet, le nom du triplet et la valeur du triplet.

Le tableau 3.1 contient les informations des messages WOSP qui peuvent être stockées dans un triplet.

TAB. 3.1 – Structure des triplets WOS [9]

Type	**Nom**	**Valeur**
EXECUTE	Le nom de la commande	L'identificateur de la commande
QUERY	Le nom de la commande	L'identificateur de la commande
SETUP	Le nom de la commande	L'identificateur de la commande
REPLY	Le nom de la commande à laquelle est associée cette réponse	L'identificateur de la commande
DATA	Le nom du champ data	La valeur du champ data
METADATA	Le nom du champ metadata	La valeur du champ metadata
FILE	Le nom local du fichier	Le nom utilisé pour le fichier

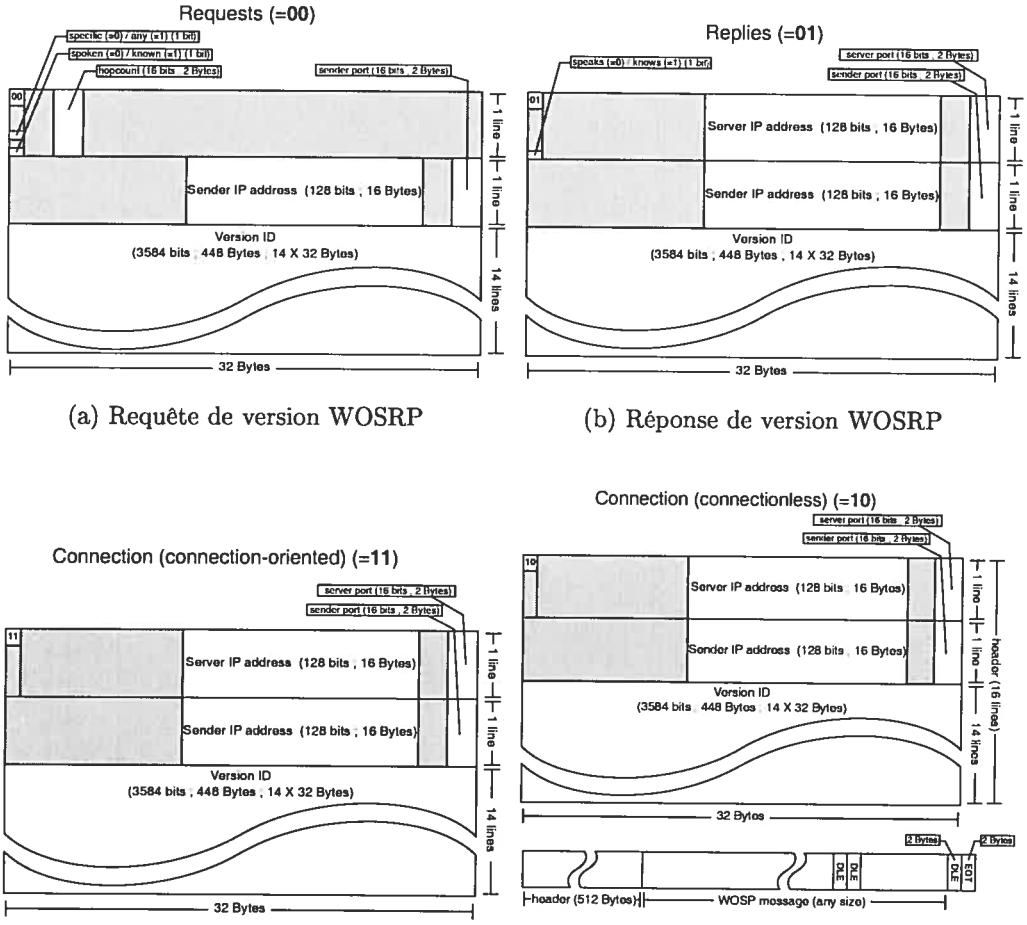
À l'envoi d'un message, le module qui implémente les classes de services transforme le message sous forme de liste de triplets. Il fournit cette liste à l'interface de la couche de communication. À la réception, le message reçu aura la même syntaxe, Ceci permet

d'utiliser un seul module pour la composition et la décomposition d'un message WOS. Pour ce qui est de l'interprétation du message, cela dépend de la version WOSP utilisée.

3.2.2 WOSRP

Le protocole WOSRP [11] apporte aux nœuds WOS les mécanismes nécessaires pour échanger les informations sur les versions WOSP qu'ils supportent, et d'obtenir des informations sur d'autres nœuds WOS compatibles avec ces versions WOSP. WOSRP permet aussi de transmettre les messages WOSP aux serveurs appropriés.

FIG. 3.3 – En-têtes WOSRP [11]



(a) Requête de version WOSRP

(b) Réponse de version WOSRP

(c) Demande de connexion WOSRP

(d) Message WOSP en mode hors connexion

Afin de localiser les classes de services deux types de messages sont utilisés, une requête WOSRP (WOSRP Request) et une réponse WOSRP (WOSRP Reply).

Une requête WOSRP (figure 3.4(a)) permet aux nœuds clients d'interroger d'autres nœuds sur leur niveau de connaissance d'une version WOSP spécifique. En particulier, nous pourrons déterminer si un nœud peut communiquer avec cette version (spoken) ou connaît d'autres nœuds capables de le faire (known).

La réponse (figure 3.4(b)) comprend le niveau de connaissance (speaks ou knows), l'adresse IP et le numéro du port du nœud WOS qui a ce niveau de connaissance ainsi que l'identificateur de la version du protocole WOSP concerné. Chaque message contient un seul numéro de version.

Pour établir une connexion WOSP entre client et serveur, le message de demande de connexion 3.4(c) est utilisé, tandis que pour envoyer un message en mode sans connexion, le message WOSP dans le mode hors connexion (figure 3.4(d)) est utilisé pour encapsuler les messages WOSP.

Les entêtes WOSRP [11] [19] illustrés dans la figure 3.3 contiennent les champs suivants :

- **specific/any** : détermine si la requête est envoyée pour identifier un nœud WOS qui peut communiquer avec une version spécifique WOSP (dans ce cas la version WOSP doit être déterminée). Dans le cas contraire (any), n'importe quelle version WOSP peut être utilisée.
- **speakes/knows** : détermine le niveau de connaissance de la version WOSP. Le niveau «speakes» indique que le nœud pour communiquer en utilisant cette vesion WOSP tandis que le niveau «knows» indique que le nœud connaît seulement d'autres nœuds qui peuvent le faire.
- **hopcount** : représente le nombre maximum de nœuds auxquels le message peut être transféré.
- **server IP address** : l'adresse IP du serveur; elle est utilisée par le nœud émetteur pour envoyer une réponse WOSRP.
- **sender IP address** : l'adresse IP de l'émetteur; elle est utilisée par le serveur et l'émetteur pour échanger des messages.
- **server port** : le port de communication du serveur; il est utilisé par l'émetteur

pour communiquer avec le serveur.

- **sender port** : le port de communication de l'émetteur ; il est utilisé par le serveur et l'émetteur pour communiquer.
- **Version ID** : l'identificateur de la version WOSP ; il permet de spécifier la version WOSP du nœud serveur WOS.

3.3 La couche de communication de WOS

La couche de communication de WOS a la responsabilité de recevoir tous les messages WOSP et WOSRP adressés à un nœud WOS spécifique et de transmettre les messages sortants.

3.3.1 Les queues de messages

Afin de permettre une communication asynchrone, les messages WOSP et WOSRP doivent être stockés à leur réception. Pour cela des queues de messages [9] sont utilisées. En effet, quand les serveurs de communication de WOS reçoivent les messages WOSP et WOSRP, ils les placent dans la queue appropriée. Une fois que l'application est prête pour traiter le prochain message, elle le demande à la queue désignée. Chaque queue est identifiée, d'une façon unique, par un identificateur de queue de message («Message Queue Identifier» ou MQID).

Les types de queues utilisées par la couche de communication sont :

- **WOSRP Request Queue :** elle contient les requêtes WOSRP reçues par les nœuds de WOS. Un nœud WOS ne possède qu'une seule queue de ce type.
- **WOSRP Reply Queue :** elle contient les réponses aux requêtes WOSRP reçues par les nœuds WOS. Un noeud WOS ne possède qu'une seule queue de ce type.
- **WOSP Connectionless Queue :** elle contient les messages WOSP reçus en mode sans connexion par un nœud WOS pour une version WOSP spécifique. Un nœud peut avoir plusieurs queues de ce type, soit une pour chaque version.
- **WOSP Connection Request Queue :** elle contient les requêtes de connexion reçues par un nœud WOS pour une version spécifique de WOSP. Un nœud WOS peut avoir plusieurs de ces queues, mais une seule au plus, est attribuée à chaque

version.

- **WOSP Connection Oriented Queue :** elle contient les messages WOSP reçus par un nœud WOS en mode orienté connexion. Il y a une seule queue de ce type pour chaque connexion WOSP établie avec un autre nœud WOS.

À la réception d'un message, la couche de communication détermine son type et vérifie si une queue qui lui est appropriée existe. Si oui elle place le message dans cette queue, sinon elle la crée avant de le placer. Le type de la queue créée dépend du type de message reçu. Initialement, aucune queue n'existe.

La couche de communication ne traite pas les messages. Cette tâche est du ressort du serveur de queue [9]. Ce dernier est une application qui informe la couche de communication qu'elle traite les messages mis dans une queue spécifique.

Une queue peut avoir au plus un serveur de queue. Si une queue n'est pas vide mais qu'aucune application ne s'est enregistrée comme serveur de queue pour cette dernière, l'interface de la couche communication de WOS lance alors une application qui jouera ce rôle.

3.3.2 L'enregistrement des serveurs de queues

La procédure d'enregistrement dépend du type de la queue [9] :

- **Pour «WOSRP Request Queue» et «WOSRP Reply Queue»** : l'application donne seulement le type de la queue. Si une application est déjà enregistrée pour cette queue, l'enregistrement échoue sinon l'application reçoit le MQID.
- **Pour «WOSP Connectionless Queue»** : l'application doit identifier le type de la queue ainsi que la version de WOSP utilisée. Une seule application peut s'enregistrer pour une version WOSP donnée.
- **Pour «WOSP Connection Request Queue»** : l'application doit identifier le type de la queue et la version de WOSP. Au plus une application peut s'enregistrer afin d'agir comme serveur de queues en mode connexion pour une version WOSP spécifique.

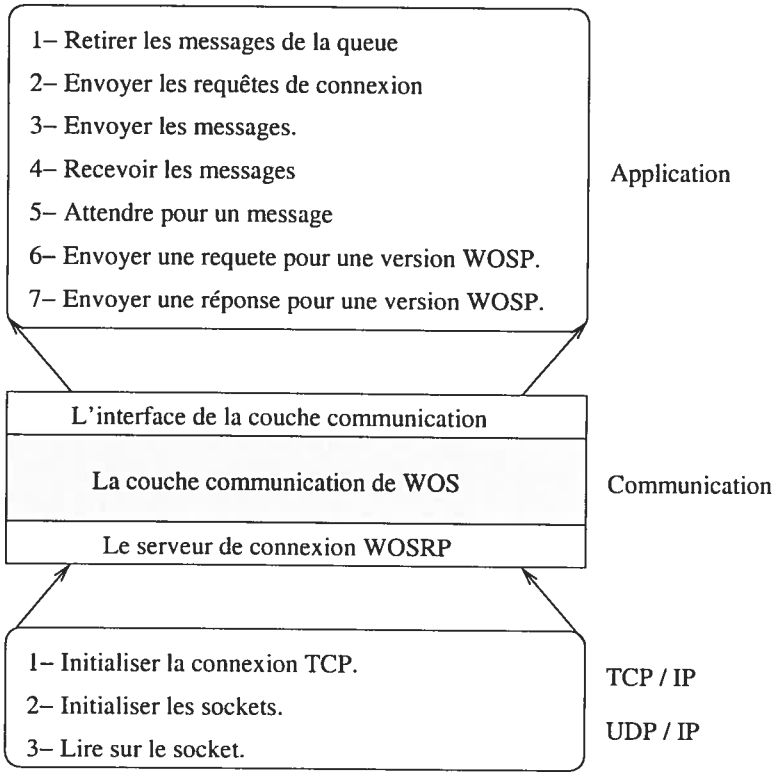


FIG. 3.4 – Services offerts et utilisés par les interfaces.

3.3.3 Les services offerts et utilisés par les interfaces

Dans la figure 3.4 nous trouvons les différents services offerts par l'interface de la couche communication à la couche application. Elle décrit aussi les services de la famille TCP/IP utilisées par le serveur de connexion WOSRP.

3.4 Scénarios de communication entre deux noeuds WOS

WOSRP a pour but de localiser des versions des familles de services et de transmettre des messages WOSP aux serveurs appropriés.

Avant d'initier une communication entre les nœuds WOS, il faut d'abord accéder à la couche communication pour pouvoir utiliser ces services. La méthode «RMIRegister.lookup("WOS_Interface")» [9] nous permettra cela.

3.4.1 Scénario de localisation des services entre deux noeuds WOS

Afin de localiser les services, deux types de messages sont utilisés : **WOSRP request** et **WOSRP reply**. La figure 3.5 représente le scénario de l'envoi d'une requête pour une version WOSP donnée.

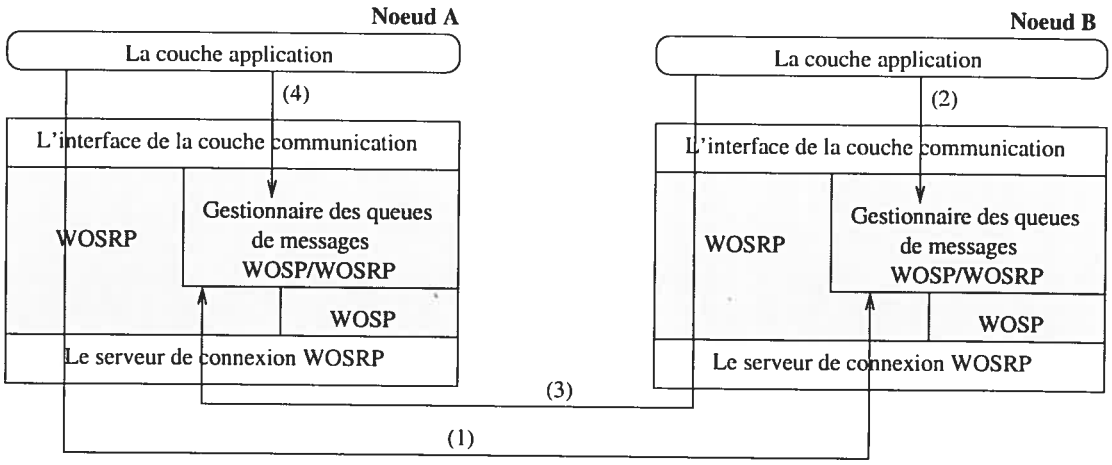


FIG. 3.5 – scénario de recherche des services entre deux noeuds WOS.

Les étapes de la transmission des requêtes entre les nœuds WOS sont les suivantes :

1. Le nœud A envoie une requête au nœud B pour une version WOSP spécifique à l'aide d'un message **WOSRP request**. Cette requête sera stockée dans la queue **WOSRP Request Queue** du nœud B.
2. Le serveur de la queue **WOSRP Request Queue** du nœud B retire le message de la queue pour que le nœud B puisse le traiter.

3. Le nœud B traite la requête et envoie une réponse au nœud A. Cette réponse sera stockée dans la queue **WOSRP Reply Queue** du nœud A.
4. Le serveur de queue **WOSRP Reply Queue** du nœud A retire le message de la queue pour que le nœud A puisse le lire.

Pour envoyer ou recevoir un message WOSRP il faut s'enregistrer à la queue appropriée. Le code suivant montre les différentes étapes pour envoyer une requête WOSRP (Figure 3.5 (1)) [9].

Programme 1 *Envoi d'une requête WOSRP.*

```
String MQID = WOSInterface.WOS_RegisterRequestServer("HP-WOSP") ;
WOSInterface.WOSRP_Request(NœudB, 9671,true, true,1 , "VersionID")
WOSInterface.WOS_Unregister(MQID) ;
```

La méthode «WOSRP Request Queue» permet de s'enregistrer à la queue «WOSRP Request Queue».

Les champs de la requête WOSRP sont : l'adresse IP du nœud Receveur, le port du nœud receveur, une valeur boolean qui indique si le nœud A communique avec la version WOSP, une valeur boolean qui indique si l'identificateur de la version est inclus le «HopCount» et l'identificateur de la version WOSP

Pour la lecture d'un message, deux fonctions peuvent être utilisées :

Programme 2 *Fonction de lecture d'un message.*

```
msg = WOSInterface.WOS_WaitForMessage(MQID) ;
msg = WOSInterface.WOS_WaitForMessage(MQID, Timeout) ;
```

La méthode «WOS_WaitForMessage(MQID)» permet d'attendre indéfiniment pour un message, tandis que la méthode «WaitForMessage(MQID, Timeout)» permet d'attendre un temps déterminé (Timeout). une fois le message reçu, elles le retirent de la queue appropriée.

À la réception du message, une réponse est produite et envoyée au nœud demandeur. Le code [9] suivant montre cette étape :

Programme 3 *Envoi d'une réponse WOSRP.*

```
WOSInterface.WOSRP_Reply(NœudA, 9671, NœudB, 9671, true, "VersionID") ;
```

La réponse WOSRP est constituée des éléments suivants : l'adresse IP du nœud receveur, le port du nœud receveur, l'adresse IP du nœud expéditeur, le port du nœud expéditeur, une valeure booléenne qui indique si le nœud B communique avec la version WOSP, l'identificateur de la version WOSP.

3.4.2 Scénario de communication entre deux nœuds WOS avec WOSP en mode sans connexion

La transmission des messages WOSP peut se faire selon deux modes : le mode sans connexion et le mode orienté connexion.

En mode sans connexion, une application envoie un seul message WOSP à une autre application qui agit comme serveur, sans établir une connexion au préalable. Dans ce cas, un message WOSRP est utilisé afin d'identifier la version WOSP appropriée du serveur et d'encapsuler le message WOSP.

La figure 3.6 illustre le scénario d'envoi de messages entre le nœud A et le nœud B en mode sans connexion.

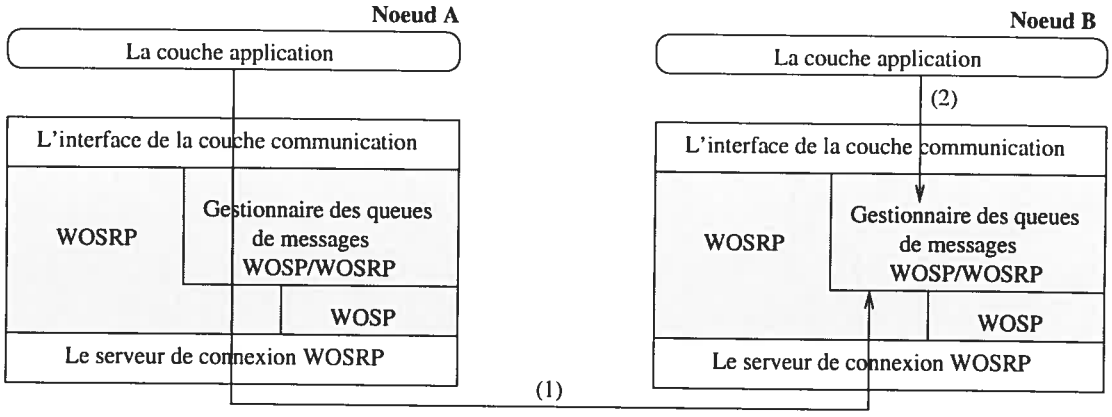


FIG. 3.6 – scénario de communication entre deux nœuds WOS en mode sans connexion.

Les étapes de transmission de la requête du nœud A au nœud B sont :

1. Le nœud A veut envoyer une requête au nœud B. Le serveur de connexion WOSRP du nœud A initialise une connexion TCP et envoie le message. Le message WOSP est encapsulé dans un message WOSRP. Il est placé dans la

queue de message **WOSP Connectionless Queue** de la version correspondante.

2. Une fois le message envoyé, la connexion TCP est fermée et le serveur de la queue **WOSP Connection Queue** de la version correspondante du nœud B récupère le message.

Avant d'envoyer un message WOSP en mode sans connexion, il faut s'enregistrer à la queue **WOSP Connectionless Queue**. Le code suivant décrit les différentes étapes pour envoyer un message WOSP. Une étape préalable pour la constitution de la liste des triplets n'est pas incluse dans ce code.

Programme 4 *Envoi de message WOSRP en mode sans connexion.*

```
String MQID = WOSInterface.WOS_RegisterConnectionlessServer("HP-WOSP") ;
WOSInterface.WOSP_SendMessage(NœudB, "VersionID", list) ;
msg = WOSInterface.WOS_WaitForMessage(MQID) ;
while (msg != null) {
    msg = WOSInterface.WOS_WaitForMessage(MQID) ;

    //traitement du message

}
WOSInterface.WOS_Unregister(MQID) ;
```

La méthode «WOS_RegisterConnectionlessServer("HP-WOSP")» permet de s'enregistrer à la queue «WOSRP Connectionless Queue».

Le message envoyé contient : l'adresse IP du nœud Receveur et l'identificateur de la version et la liste de triplets.

3.4.3 Scénario de communication entre deux nœuds WOS avec WOSP en mode connexion

La figure 3.7 présente le scénario d'échange d'informations entre deux nœuds WOS en mode connexion.

Les étapes de la transmission des requêtes entre les nœuds WOS sont :

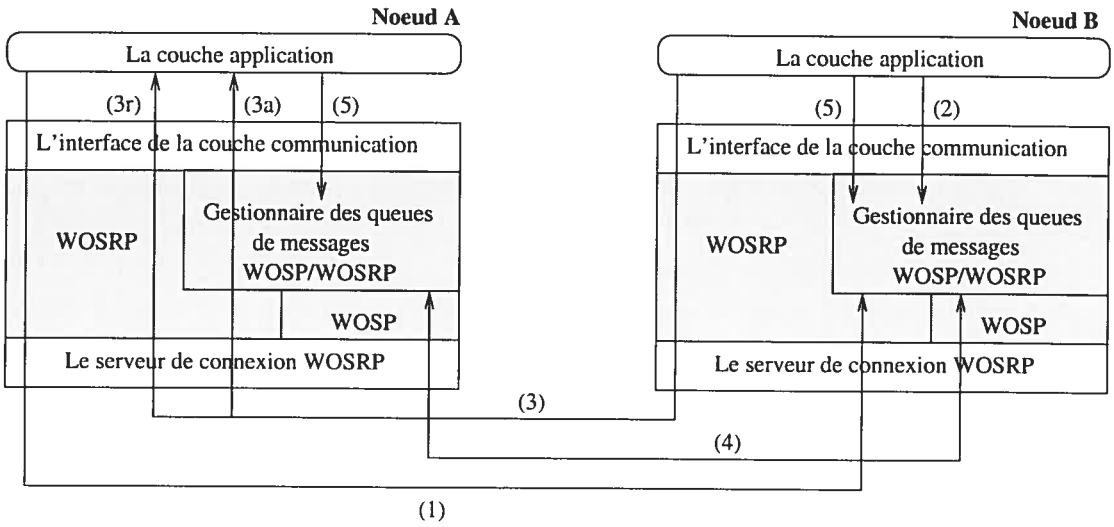


FIG. 3.7 – scénario de communication entre deux nœuds WOS en mode connexion.

1. Le nœud A envoie au nœud B une demande de connexion qui sera stockée dans la queue **WOSP Connection Request queue** de la version correspondante sur le nœud B. La demande est faite avec un message **WOSRP Connection Request**.

2. À la réception de la requête, le serveur de queue **WOSP Connection Request Queue** du nœud B récupère le message de la queue, crée une queue de type **WOSP Connection Oriented Queue** et lance un serveur de queue pour la gérer.

3. Le nœud B envoie une réponse au nœud A. Deux cas peuvent se présenter, la connexion est refusée ou bien le temps est écoulé. Dans les deux cas, le nœud A reçoit la valeur «null» (3r), la queue **WOSP Connection Oriented Queue** du nœud B ainsi que la connexion TCP seront détruites. Si la connexion est acceptée, alors le nœud A reçoit un acquittement et crée une queue de type **WOSP Connection Oriented Queue** (3a).

4. une fois la connexion établie, les deux nœuds A et B peuvent échanger les messages WOSP de manière asynchrone. Ainsi, leur serveur de connexion WOSP respectif écoute sur le «socket» et les deux applications récupèrent les messages des queues.

Dans une communication en mode connexion, il y a deux grandes étapes : l'établise-

ment de la connexion et l'envoi de message. Les codes ci-dessous décrivent ces étapes :

Programme 5 *Établissement de la connexion entre deux nœuds WOS..*

```
String MQID = WOSInterface.WOS_RegisterConnectionServer("HP-WOSP") ;

//Pour s'enregistrer à la queue «WOSRP Cennection Request»


//Envoi de la demande de connexion au nœud B

WOSInterface.WOSP_SetupConnection(NœudB, 9671, "VersionID") ;

//La rêquete de connection contient : l'adresse IP du nœud Receveur, Le
port du nœud receveur et l'identificateur de la version WOSP


//Le nœud B peut accepter ou rejeter la connexion. Le premier cas est
représenté par le code suivant :

WOSInterface.WOSP_AcceptConnection(MQID) ;

//Pour le cas de rejet de connexion le code est le suivant :

WOSInterface.WOSP_RejectConnection(MQID) ;

//Une fois la connexion acceptée l'envoi de message se fait avec le code
suivant :

WOSInterface.WOSP_SendMessage(NœudB,list) ;

//Le message envoyé contient : l'adresse IP du nœud Receveur et la liste
des triplets

msg = WOSInterface.WOS_WaitForMessage(MQID) ;
while (msg!=null) {
   msg = WOSInterface.WOS_WaitForMessage(MQID) ;

   //traitement du message

}
WOSInterface.WOS_Unregister(MQID) ;
```

Dans le cas d'une connexion acceptée, deux problèmes se posent dans l'implémentation actuelle de la couche de communication de WOS : le cas où la connexion

n'est pas fermée et le cas où la connexion n'est pas utilisée. En effet, il n'y a pas de mécanisme qui détruit les queues et les «sockets» dans ces cas.

3.5 Conclusion

Dans ce chapitre, nous avons présenté le fonctionnement de WOS, en particulier sa couche de communication. Ensuite nous avons montré la structure actuelle des deux protocoles WOSP et WOSRP.

Nous avons aussi analysé l'utilisation des différents types de message de WOS selon le scénario de communication. Cette étude nous a permis de mieux comprendre la structure actuelle de la couche de communication de WOS et des deux protocoles WOSP et WOSRP afin de pouvoir identifier les changements à réaliser à ce niveau.

Chapitre 4

Présentation du model de WOSP/WOSRP en MIME, HTTP et XML

Dans le chapitre précédent, nous avons décrit le fonctionnement de WOS ainsi que la structure actuelle de la couche communication et des protocoles WOSP et WOSRP.

Dans ce chapitre, nous commencerons par présenter HTTP, MIME et XML. Ensuite, nous définirons le nouvel en-tête WOSRP qui tient compte de l'implémentation de WOSP et WOSRP en HTTP, MIME et XML. Finalement, nous présenterons les interfaces WOSP et WOSRP avec une version MIME, HTTP et XML.

HTTP, MIME et XML représentent les standards Internet les plus utilisés. Ces standards offrent un moyen de communication avec les serveurs distants proposant des services Web tout en contournant les problèmes de sécurité et les murs coupe-feu «firewalls». Ainsi, les efforts d'intégration sont réduits.

Puisque la communication dans WOS se fait à l'aide de l'infrastructure actuelle d'Internet, alors la standardisation de WOSP et WOSRP en utilisant HTTP, MIME et XML nous permet de faire transiter les données sans avoir à modifier les routeurs, les murs coupe-feu ou les serveurs Proxy.

4.1 Présentation de HTTP

HTTP (HyperText Transfer Protocol) [22] [3] est un protocole de la couche application qui permet l'échange des données entre clients et serveurs à travers le Web.

Le protocole HTTP utilise la normalisation des identificateurs de ressources uniformes «Uniform Resource Identifier» (URI), soit sous forme de liens «Uniform Resource Link» (URL) ou de noms «Uniform Resource Name» (URN), pour indiquer l'objet sur lequel porte la méthode. Les messages sont transmis sous le même format que les messages électroniques (E-mail). D'ailleurs, HTTP réutilise les extensions «Multipurpose Internet Mail Extensions» (MIME) définies par la messagerie électronique.

Le protocole HTTP peut être implémenté au-dessus de plusieurs types de protocoles de la couche communication pourvu que ce protocole garantisse la sécurité de transmission. Par exemple sur Internet, les communications HTTP s'appuient principalement sur le protocole de connexion TCP/IP. Le port utilisé par défaut est le port TCP 80, d'autres ports peuvent être utilisés.

4.1.1 Modèle de Fonctionnement

Le protocole HTTP utilise un mécanisme de requête et réponse. Le client initialise la communication avec le serveur en lui envoyant une requête avec un entête HTTP, suivie d'un message MIME. Une fois la requête reçue, le serveur envoie une réponse de structure similaire, c.-à-d. un entête HTTP et un message de type MIME.

Dans le cas le plus simple, la communication HTTP est initiée par l'utilisateur qui envoie une requête au serveur d'origine. Ceci peut être réalisé par une connexion simple entre un utilisateur et un serveur origine (figure 4.1).

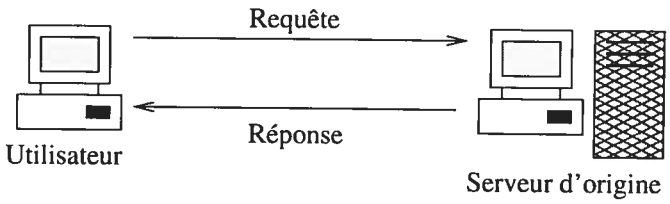


FIG. 4.1 – Communication entre client et serveur HTTP

Dans une situation plus complexe, un ou plusieurs intermédiaires peuvent apparaître dans la communication. Ces intermédiaires peuvent être des Proxy, des routeurs ou des tunnels. Un Proxy reçoit une requête, recompose une partie ou la totalité de la requête et la réémet à sa destination. Un routeur agit en tant qu'entité qui peut

(au besoin) encapsuler le message avant de l'envoyer à sa destination. Un tunnel sert de relais entre deux parties d'une même connexion.

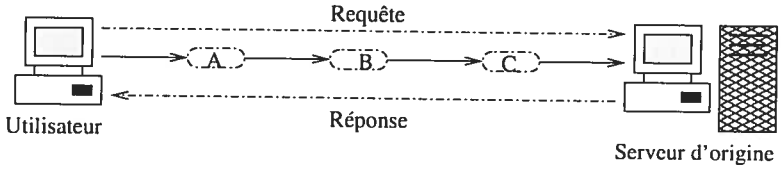


FIG. 4.2 – Communication entre client et serveur HTTP

La figure 4.2 schématise le cas de trois intermédiaires A, B, et C entre un utilisateur et le serveur d'origine. Une requête ou une réponse passant d'un bout à l'autre doit transiter par les quatre connexions impliquées dans la chaîne. Chaque participant à la communication peut avoir d'autres communications avec d'autres nœuds.

Tous les éléments actifs de la communication doivent avoir une mémoire cache pour stocker des informations. Le but de cette technique est de réduire le chemin de la requête. En effet si un des intermédiaires possède la réponse, le processus n'aura pas besoin de remonter jusqu'au serveur d'origine pour l'avoir. La figure 4.3 illustre le cas où B dispose dans sa mémoire cache d'une information relative à la requête envoyée par l'utilisateur au serveur.

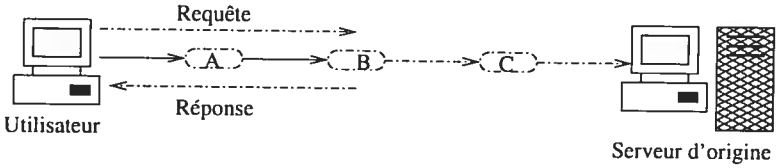


FIG. 4.3 – Communication entre client et serveur HTTP

4.1.2 HTTP/1.1

La version HTTP/1.1 introduit de nouveaux concepts par rapport aux versions qui l'ont précédé, parmi ces concepts nous pouvons citer :

- Le mécanisme «hop-by-hop» ;
- Codage de transfert «Transfer encoding» ;
- Accueil virtuel «Virtual hosting» ;
- Assurance de la transparence sémantique ;
- Support des variantes de ressources.

4.1.3 Le format d'un message HTTP/1.1

Un message HTTP peut être une requête envoyée du client au serveur ou une réponse envoyée du serveur au client. La requête ainsi que la réponse peuvent avoir zéro ou plusieurs en-têtes séparés du corps du message, optionnel, par les deux caractères **CR** (Carriage return) et **LF** (Linefeed).

4.2 Présentation de MIME

MIME (Multi-purpose Internet Mail Extensions) [6] [7] est une spécification décrivant les formats de messages multimédias sur l'Internet. MIME permet en particulier l'échange de textes écrits dans des jeux de caractères différents, ainsi que l'utilisation de courrier électronique multimédia entre les systèmes informatiques.

MIME donne la possibilité de composer et de consulter des messages contenant d'autres caractères que l'ASCII, en l'occurrence, du texte contenant des caractères spéciaux, des images, des sons et des séquences vidéo.

Les extensions MIME introduisent deux nouveaux codages pour les données autres que l'ASCII, appelés «Quoted-Printable» (QP) et «base64». Le premier permet de coder tout alphabet nécessitant plus que 7 bits. Le second est préféré pour les fichiers binaires à transmettre sous forme de pièces jointes incluses dans les messages.

Les deux codages définis par MIME permettent de résoudre les problèmes suivants lors du transport du message sur l'Internet :

- La conversion des séquences CRLF ;
- La transmission des caractères NULs ;
- La mauvaise interprétation des tabulations ;
- L'élimination des lignes au-delà du 76ème caractère ;
- Les espaces blancs excédentaires.

Pour que MIME soit utile, il doit être implanté dans les logiciels de communication utilisés par les deux parties. Ainsi, si deux correspondants, échangeant des messages électroniques veulent éliminer les problèmes de transfert ou de lecture de données, ils doivent tous les deux utiliser un logiciel conforme à MIME.

4.2.1 En-tête de MIME

Les en-têtes de MIME sont les suivants :

- **MIME-Version** : permet de spécifier la version de MIME utilisée. Cet en-tête est le seul qui s'applique au message en entier ;
- **Content-Type** : permet de décrire la nature des données contenues dans le message ;
- **Content-Transfer-Encoding** : permet de décrire l'encodage utilisé pour le transport du message ;
- **Content-ID** : Permet d'identifier une entité d'une façon unique ;
- **Content-Description** : permet de donner la description de l'entité.

4.3 Présentation de XML

XML (Extensible Markup Language) [25] [12] peut être considéré comme une généralisation de HTML (Hyper Text Makup Language) où au lieu d'utiliser un jeu de balises prédéfinies, c.-à-d. des commandes HTML constituées d'une directive sous forme de mot-clé encadré par les caractères $<$ et $>$, qui permet de mettre en forme un texte et qui indique au navigateur Web comment devrait être affiché un document, l'auteur peut écrire ces propres balises. Le but principal de XML est de faciliter le traitement automatisé des données.

Le langage XML décrit une classe d'objets de données appelés «documents XML» et décrit partiellement le comportement des programmes qui les traitent. XML est une forme restreinte de SGML (Standard Generalized Makup Language), le langage normalisé de balisage généralisé. Les documents XML sont des documents conformes à SGML.

Pour lire des documents XML ou pour accéder à leurs contenus et à leur structure, un module logiciel appelé «processeur XML» est utilisé. Le processeur XML lit les données XML et identifie les informations qu'il doit fournir à l'application. Les objectifs de conception de XML sont les suivants :

- XML doit pouvoir être utilisé sans difficulté sur Internet ;
- XML doit soutenir une grande variété d'applications ;
- XML doit être compatible avec SGML ;

- il doit être facile d'écrire des programmes traitant les documents XML ;
- le nombre d'options dans XML doit être réduit au minimum ;
- les documents XML doivent être lisibles et clairs ;
- la conception de XML doit être préparée rapidement ;
- il doit être facile de créer des documents XML.

Cette spécification, ainsi que certaines normes fournissent toutes les informations nécessaires pour comprendre la version 1.0 de XML et pour construire des programmes pour la traiter.

Afin d'écrire un document XML conforme, un certain nombre de règles de base doivent être respectées :

- toutes les balises portant sur un contenu non vide doivent être fermées ;
- les balises n'ayant pas de contenu doivent se terminer par /> ;
- les noms d'attributs sont en minuscules ;
- les valeurs d'attributs doivent être entre guillemets ;
- les majuscules et les minuscules doivent être respectés pour toutes les occurrences des noms de balises.

4.4 Présentation de l'en-tête générique WOSRP

Dans cette partie nous proposons des extensions à l'en-tête WOSRP (présenté au deuxième chapitre) pour lui permettre de supporter les deux implémentations de WOS, à savoir, l'ancienne implémentation utilisant les langages WOSP et WOSRP, et la nouvelle utilisant les standards HTTP, MIME et XML.

La figure 4.4 montre l'architecture du nouvel en-tête générique WOSRP :

Dans la nouvelle structure de l'en-tête WOSRP (figure 4.4) nous avons ajouté les champs suivants :

- **«WOSP protocol ID»** : ce champ permet d'identifier le type de message WOSP transmis. Il prend la valeur 0 pour l'ancienne implémentation de WOSP et 1 pour la nouvelle version écrite en XML. À la réception d'un message, le nœud WOS teste la valeur du champ «WOSP Protocol ID» et selon le type de message il effectue le traitement approprié.
- **«Message type»** : ce champ représente le type de message envoyé. Il prend

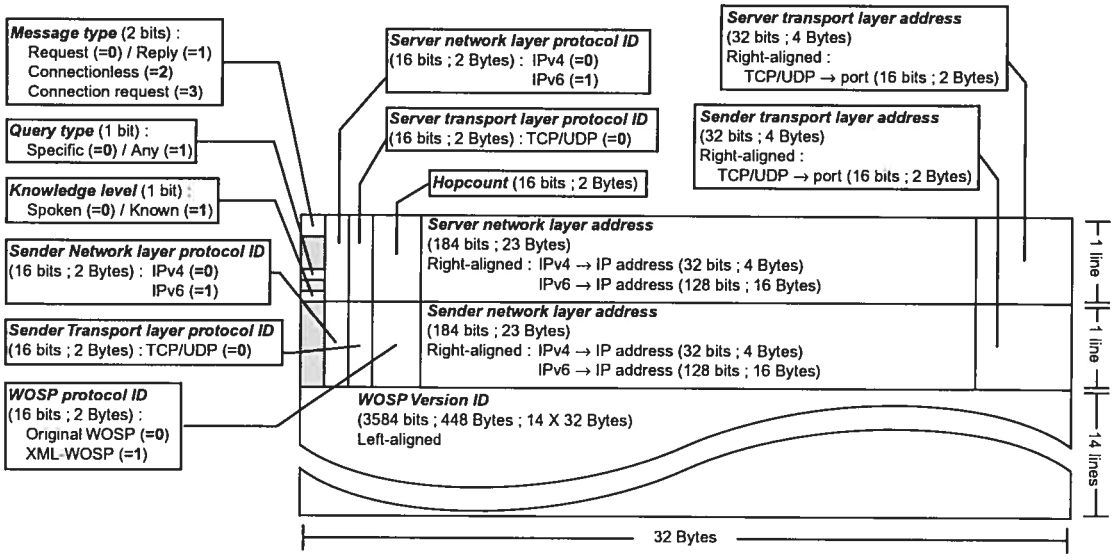


FIG. 4.4 – En-tête générique WOSRP

la valeur 0 pour une requête, la valeur 1 pour une réponse, la valeur 2 pour un message en mode sans connexion et la valeur 3 pour une demande de connexion. Le champ «Message Type» a été ajouté afin de pouvoir regrouper les quatre types des anciens en-têtes WOSRP, représentant les quatre messages WOSRP, dans un seul en-tête.

- **«Sender network layer protocol ID» et «Server network layer protocol ID»** : ces champs représentent les identificateurs du protocole de la couche réseau de l'expéditeur et du serveur, ils prennent la valeur 0 pour IPv4 et la valeur 1 pour IPv6. Ces champs ont été ajoutés pour permettre à WOS d'utiliser d'autres protocoles que IPv4, utilisé actuellement, entre autres IPv6.
- **«Sender transport layer protocol ID» et «Server transport layer protocol ID»** : ces champs représentent les identificateurs du protocole de la couche transport de l'expéditeur et du serveur, ils prennent la valeur 0 pour les deux protocoles TCP et UDP. Ces champs ont été ajoutés pour permettre éventuellement à WOS d'utiliser d'autres protocoles que TCP et UDP.

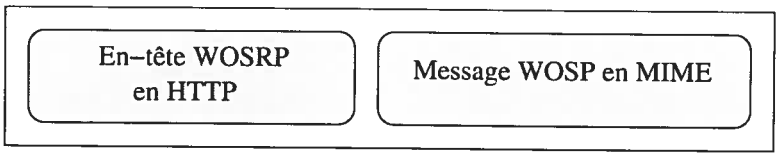
4.5 Présentation de l'interface WOSRP avec une version HTTP/MIME

Dans cette partie nous présentons l'implémentation de l'en-tête générique WOSRP en HTTP et MIME. Pour cela deux scénarios sont proposés.

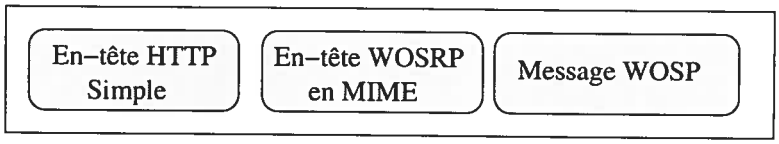
Dans le premier scénario nous définirons les différents champs de l'en-tête WOSRP à l'aide des en-têtes HTTP. Dans ce cas, MIME sera utilisé afin d'identifier le type de message WOSP, qui peut prendre une des deux formes d'implémentation : originale ou en XML.

Dans le deuxième scénario, l'en-tête WOSRP est représenté en MIME et encapsulé avec le message WOSP dans un en-tête simple HTTP.

La figure 4.5 illustre ces deux scénarios :



Premier scénario



Deuxième scénario

FIG. 4.5 – Les deux scénarios possibles pour écrire l'en-tête WOSRP en HTTP/MIME

4.5.1 Présentation de l'en-tête WOSRP

Le message WOSRP peut être une requête, une réponse, un message en mode hors connexion ou une demande de connexion. Chacun de ces messages à un en-tête spécifique. En effet dans chaque type de message WOSRP nous allons utiliser un emsemble de champs de l'en-tête générique WOSRP (figure 4.4). Chaque champ correspond à un libellé conforme à la nomenclature HTTP. Le tableau 4.1 représente la correspondance entre les champs de l'en-tête WOSRP et les noms utilisés pour la représentation de cet en-tête en format HTTP.

TAB. 4.1 – Correspondance entre les champs de l'en-tête WOSRP et leurs équivalents selon HTTP

Les champs de l'en-tête WOSRP	**Leurs correspondants en HTTP**
Message type	WOS-MessageType
Query type	WOS-QueryType
Knowledge level	WOS-KnowledgeLevel
Sender network layer protocol ID	WOS-SenderNLPID
Sender transport layer protocol ID	WOS-SenderTLPID
WOSP protocol ID	WOSP-ProtocolID
Server network layer protocol ID	WOS-ServerNLPID
Server transport layer protocol ID	WOS-ServerTLPID
Server transport layer address	WOS-ServerTLAdd
Sender transport layer address	WOS-SenderTLAdd
HopCount	WOS-HopCount
Server network layer address	WOS-ServerNLAdd
Sender network layer address	WOS-SenderNLAdd
WOSP version ID	WOSP-VersionID

Nous allons ensuite présenter des exemples pour chaque type d'en-tête (requête WOSR, réponse WOSRP, message WOSP en mode sans connexion et demande de connexion WOSRP) en utilisant la syntaxe HTTP.

Une requête WOSRP

Ceci est un exemple d'en-tête HTTP pour une requête WOSRP :

Programme 6 *Exemple de requête WOSRP.*

```
Post * HTTP/1.1
WOS-MessageType : 0
WOS-QueryType : 1
WOS-Knowledgelevel : 1
WOS-SenderNLPID : 0
WOS-SenderTLPID : 0
WOSP-ProtocolID : 1
WOS-HopCount : -1
WOS-SenderTLAdd : 9671
WOS-SenderNLAdd : 132.204.27.62
WOSP-VersionID : "HP-WOSP"
```

Une réponse WOSRP

Ceci est un exemple d'en-tête HTTP pour une réponse WOSRP :

Programme 7 *Exemple de réponse WOSRP.*

```
HTTP/1.1 200 ok
WOS-MessageType : 1
WOS-Knowledgelevel : 1
WOS-SenderNLPID : 0
WOS-SenderTLPID : 0
WOSP-ProtocolID : 1
WOS-HopCount : -1
WOS-ServerNLPID : 0
WOS-ServerTLPID : 0
WOS-SenderTLAdd : 9671
WOS-SenderNLAdd : 132.204.27.62
WOS-ServerTLAdd : 9671
WOS-ServerNLAdd : 132.204.27.157
WOSP-VersionID : "HP-WOSP"
```

Un message WOSRP en mode sans connexion

Ceci est un exemple d'en-tête HTTP pour un message WOSRP en mode sans connexion :

Programme 8 *Exemple de message WOSRP en mode sans connexion.*

```
post * HTTP/1.1
WOS-MessageType : 2
WOS-SenderNLPID : 0
WOS-SenderTLPID : 0
WOSP-ProtocolID : 1
WOS-ServerNLPID : 0
WOS-ServerTLPID : 0
WOS-SenderTLAdd : 9671
WOS-SenderNLAdd : 132.204.27.62
WOS-ServerTLAdd : 9671
WOS-ServerNLAdd : 132.204.27.157
WOSP-VersionID : "HP-WOSP"
```

Une requête de connexion WOSRP

Ceci est un exemple d'en-tête HTTP pour une demande de connexion WOSRP.

Programme 9 *Exemple de demande de connexion WOSRP.*

```
post * HTTP/1.1
WOS-MessageType : 3
WOS-SenderNLPID : 0
WOS-SenderTLPID : 0
WOSP-ProtocolID : 1
WOS-ServerNLPID : 0
WOS-ServerTLPID : 0
WOS-SenderTLAdd : 9671
WOS-SenderNLAdd : 132.204.27.62
WOS-ServerTLAdd : 9671
WOS-ServerNLAdd : 132.204.27.157
WOSP-VersionID : "HP-WOSP"
```

4.6 Présentation du message WOSP avec la version XML

Dans cette partie nous allons présenter la structure du message WOSP en XML (figure 4.6).

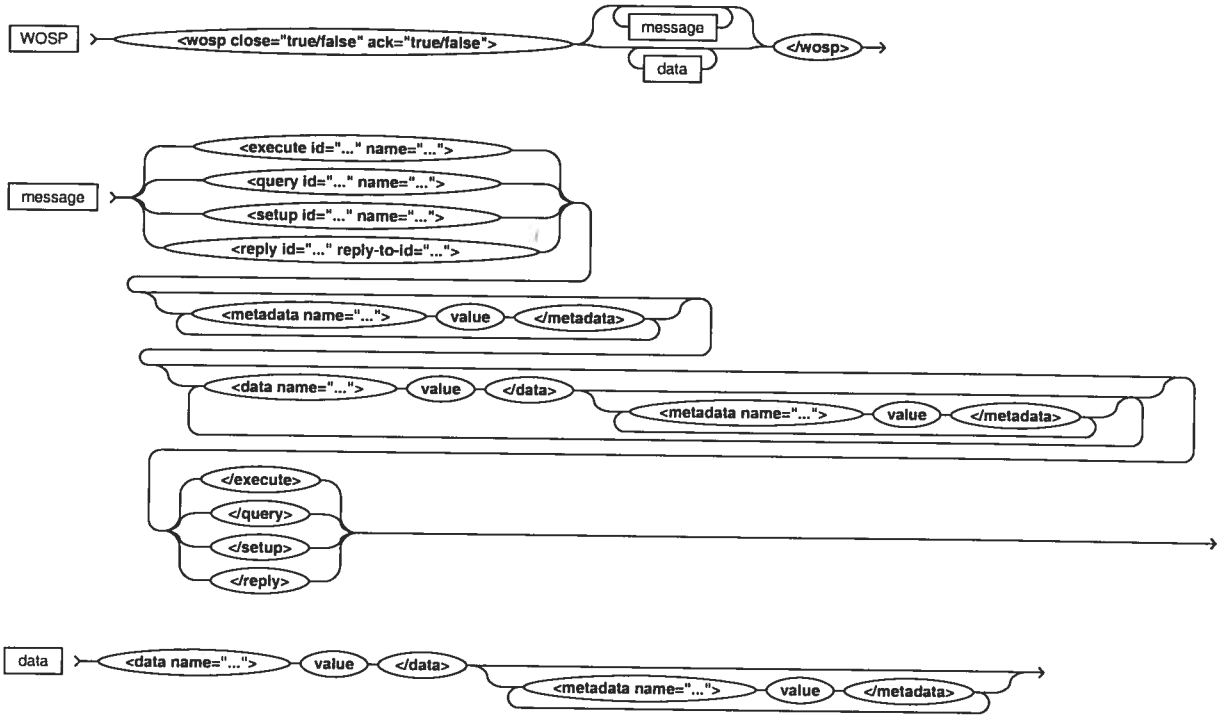


FIG. 4.6 – Structure du message WOSP

4.7 Conclusion

Dans ce chapitre, nous avons présenté les standards d'Internet HTTP, MIME et XML qui seront utilisés par la suite pour implémenter les deux protocoles WOSP et WOSRP. Ensuite, nous avons défini une structure de l'en-tête WOSRP qui tient compte de la nouvelle implémentation de WOSP et WOSRP. Finalement, nous avons présenté la structure de WOSP et WOSRP avec une version MIME, HTTP et XML.

Chapitre 5

Classification des fonctions WOSP/WOSRP

Ce chapitre est divisé en trois sections. Dans la première, nous regroupons les principales classes de l'implémentation Java de WOS selon les couches du nœud WOS qui sont : L'interface de la couche de communication, le gestionnaire des queues de messages WOSP/WOSRP et finalement le serveur de connexion WOSRP. Dans la deuxième section, nous identifions les relations entre ces différentes classes et dans le troisième module, nous définissons les interfaces qui permettront de réduire les dépendances entre les classes de WOSP/WOSRP.

L'objectif est d'identifier les relations entre les classes dépendantes de WOSP et WOSRP et celles qui ne le sont pas, afin de les éliminer.

5.1 Regroupement des classes de WOS selon les couches du nœud WOS

Dans cette partie nous regroupons les principales classes de WOS selon leur appartenance à une des couches du nœud WOS (figure 5.1). Pour plus de détails sur les différentes méthodes de ces classes ainsi que leurs fonctions, référez vous à l'annexe A.

Cette classification nous permet d'identifier trois types de classes WOSP/WOSRP : les classes qui dépendent des langages WOSP et WOSRP, les classes qui sont indépendantes et les classes qui sont à la frontière des langages WOSP et WOSRP (c.-à-d., partiellement dépendantes des langages WOSP et WOSRP).

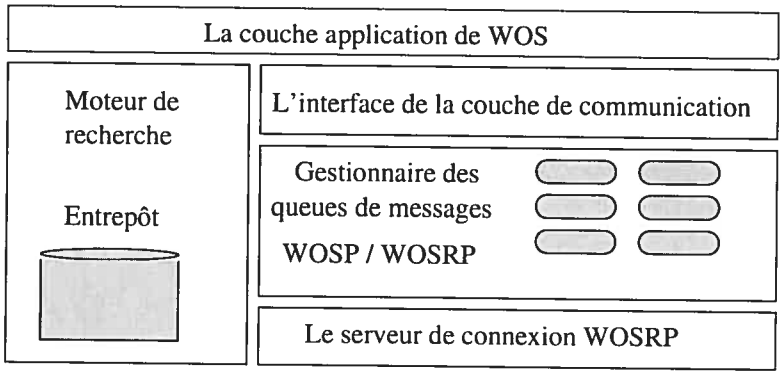


FIG. 5.1 – L'architecture du nœud WOS [9]

5.1.1 L'interface de la couche de communication

L'interface de la couche communication fait le lien entre la couche application de WOS et la couche communication, elle contient la classe «**WOS_ClientImpl**». Cette classe assure la gestion des informations dans les queues de messages, et gère les différentes applications qui s'enregistrent en tant que gestionnaire de queues de messages. Elle contient des méthodes pour envoyer les requêtes et les réponses WOSRP, initialiser les connexions WOSP et accepter ou rejeter une connexion.

5.1.2 Le gestionnaire des queues de messages WOSP/WOSRP

Le gestionnaire des queues de messages WOSP/WOSRP définit le comportement des différentes queues de message .Pour cela différentes classes sont utilisées :

1. **WOSRP_ReplyQueue** : cette classe définit le comportement de la queue des réponses WOSRP «**WOSRP_ReplyQueue**». Elle détermine l'existence d'une queue de ce type, vérifie si elle est servie par une application et enregistre ou annule l'enregistrement d'une application en tant que gestionnaire de cette queue.
2. **WOSRP_RequestQueue** : cette classe définit le comportement de la queue de requête WOSRP «**WOSRP_RequestQueue**». Elle a la même fonction que la classe «**WOSRP_ReplyQueue**» mais pour une queue de type «**WOSRP_RequestQueue**».
3. **WOSP_ConnectionOrientedQueue** : cette classe définit le comportement

de la queue **WOSP_ConnectionOrientedQueue**. Elle permet d'accepter ou de rejeter une connexion, d'envoyer une requête de connexion et de retourner les différentes valeurs utilisées dans cette connexion.

4. **WOSP_ConnectionRequestQueue** : cette classe définit le comportement de la queue «**WOSRP_ConnectionRequestQueue**». Elle permet d'envoyer une requête de connexion WOSP, elle lance le serveur spécifique capable de servir la version appropriée «**WOSP connection request manager**» et enregistre ou supprime une application en tant que serveur de requête de connexion WOSP (WOSP connection request server).
5. **WOSP_ConnectionlessQueue** : cette classe définit le comportement de la queue «**WOSP_ConnectionlessQueue**». Elle permet d'envoyer une requête de connexion WOSP et de retourner les différentes valeurs utilisées dans cette connexion. Elle enregistre ou annule l'enregistrement d'une application en tant que **WOSP «Connectionless message manager»**.

Le gestionnaire des queues de messages WOSP/WOSRP permet aussi de gérer le canal de communication en mode connexion, ainsi que les messages WOSP. Pour cela les classes suivantes sont utilisées :

1. **WOSP_Connection** : cette classe est utilisée pour créer un flux (stream) pour la communication en mode connexion. Elle permet d'obtenir une référence aux flots d'entrée et de sortie, et une référence au socket d'origine et à celui qui répond à la connexion.
2. **WOSP_Parser** : Cette classe contient toutes les méthodes nécessaires au programme d'analyseur syntaxique WOSP (WOSP Parser), qui convertit les messages WOSP sous un format générique pour les envoyer ou les recevoir.

5.1.3 Serveur de connexion WOSRP

Le serveur de connexion WOSRP fait le lien entre la couche de communication et les services de la famille TCP/IP ou UDP/IP. Il peut créer un serveur et gérer ses messages.

Pour accomplir ses tâches, le serveur de connexion utilise les classes suivantes :

1. **WOSRP_Client** : Cette classe gère les communications TCP et UDP du serveur WOSRP. Elle permet de construire l'entête WOSRP, d'encapsuler le message WOSP et d'initialiser la connexion afin d'envoyer l'entête WOSRP. Elle permet aussi d'envoyer les requêtes de connexion WOSP ainsi que les requêtes et les réponses WOSP.
2. **WOSRP_ServerTcpSocket** : Cette classe traite l'information reçue pour différents services (connexion et sans connexion) en mode de transmission TCP de WOSRP. Elle traite l'en-tête reçu de WOSRP, distingue l'option de service de l'en-tête pour déterminer si elle utilise le service en mode connexion ou sans connexion.
3. **WOSP_FillConnectionOrientedQueue** : Cette classe lit sur le socket et stocke les messages dans une queue dès qu'ils arrivent.
4. **WOSRP_Pipe** : Cette classe lance des processus en parallèle pour filtrer les informations qui arrivent de WOSRP Client vers WOSRP Serveur dans le mode sans connexion.
5. **WOSRP_Server** : Cette classe crée un nouveau serveur **«WOSRP_Server»** avec le protocole spécifique (TCP ou UDP), puis elle le lance. Elle permet aussi de traiter l'information de l'en-tête WOSRP reçu, et d'initialiser les connexions TCP et UDP avec WOSRP client dans un port spécifique
6. **WOSRP_TcpServerException** : Cette classe définit une exception qui sera lancée à l'arrêt de **«WOSRP-Server»** pour une connexion TCP ou quand une erreur se produit.
7. **WOSRP_UdpServerException** : Cette classe définit une exception qui sera lancée à l'arrêt de **«WOSRP-Server»** pour une connexion UDP ou quand une erreur se produit.

5.2 Identification des dépendances entre les classes

Dans cette partie nous définissons les dépendances entres les différentes classes de WOSP et WOSRP. Ces dépendances peuvent êtres de trois types :

- Une relation d'héritage entre les classes, notée par «extends».

- Une classe qui implémente une interface, notée par «implements».
- Une relation d'utilisation entre les classes, notée par «agregates».

  Les termes extends, implements, agregates sont emprintés du langage de programmation JAVA.

Cette classification nous permettra d'identifier les relations entre les classes dépendantes de WOSP/WOSRP, les classes indépendantes et les classes à la frontière des langages WOSP/WOSRP afin de définir les interfaces qui nous permettront d'éliminer cette dépendance.

5.2.1 Les classes indépendantes

Les classes indépendantes sont les classes qui n'ont aucune relation avec d'autres classes, elles sont complètement indépendantes des autres classes.

1. WOS_DeadThread
2. WOS_TimeStampServerException
3. WOS_UdpServerException
4. WOSP_ListOfTripletsException
5. WOSP_NotStorableException
6. WOSRP_TCPServerException
7. Base64
8. WOSP_VersionWarehouse
9. WOSP_Triplet
10. WOSP_Connection

5.2.2 Les interfaces

Les interfaces sont des classes qui permettent au programmeur de définir la forme d'autres classes. Dans les interfaces, on déclare les différentes méthodes qui seront utilisées dans les classes qui implémenteront ces interfaces.

1. WOS_ConfigInstLexSymbols
2. WOS_Client

3. WOS_Constants
4. WOSP_LexSymbols
5. WOSP_InstLexSymbols

5.2.3 Les classes dépendantes

Les classes dépendantes sont des classes qui ont un lien avec d'autres classes ou interfaces.

TAB. 5.1 – Les relations de dépendance entre les classes WOSP/WOSRP

Nom de la classe	**Les classes dépendantes**	**Le type de dépendance**
WOS_ConfigInstlex	WOSP_Lex	extends
	WOS_ConfigInstlexSymbols	implements
WOS_MessageQueueNode	WOS_MessageQueue	agregates
	WOS_MessageQueueID	agregates
WOS_Config	WOSP_LexSymbols	implements
	WOS_ConfigInstLexSymbols	implements
	WOS_Constants	implements
	WOS_ConfigInstLex	agregates
WOS_Servers	WOS_MessageQueueNode	agregates
	WOS_MessageQueueNode	agregates
	WOS_Params	agregates
	WOS_Constants	implements
WOS_TimesStampServer	WOS_Constants	implements
	WOS_Params	agregates
WOS_MessageQueueID	WOS_TimeStamp	extends
WOS_TimeStamp	WOS_Constants	implements
	WOS_Params	agregates
WOS_ClientImpl	WOS_Client	implements
	WOS_Params	agregates
	WOS_MessageQueue	agregates
WOS_MessageQueue	WOS_Params	agregates
	WOS_MessageQueueID	agregates
WOS_Message	WOSP_ListOfTriplets	agregates
WOS_Params	WOS_Constants	implements
WOSP	WOSP_LexSymbols	implements
	WOS_InstLexSymbols	implements
	WOS_Constants	implements
	WOSP_InstLex	agregates

TAB. 5.2 – Les relations de dépendance entre les classes WOSP/WOSRP (suite)

Nom de la classe	**Les classes dépendantes**	**Le type de dépendance**
WOSP_Lex	WOSP_LexSymbols	implements
WOSP_InstLex	WOS_InstLexSymbols	implements
	WOSP_Lex	extends
WOSP_MsgId	WOS_MessageQueueID	agregates
	WOS_TimeStamp	agregates
	WOS_Params	agregates
WOSP_ConnectionRequestQueue	WOS_MessageQueue	extends
WOSP_CammandID	WOSP_MsgId	agregates
WOSPConnectionlessQueue	WOS_MessageQueue	extends
	WOSP_Parser	agregates
WOSP_Parser	WOS_Constants	implements
WOSP_FillConnectionOrientedQueue	WOSP	agregates
	WOSP_Parser	agregates
	WOS_Message	agregates
	WOSP_Connection	agregates
	WOS_MessageQueue	agregates
WOSP_ConnectionOrientedQueue	WOS_MessageQueue	extends
	WOSP_FillConnectionOrientedQueue	agregates
	WOSP	agregates
	WOSP_Parser	agregates
	WOS_Message	agregates
	WOSP_Connection	agregates
WOSP_ListOfTriplets	WOSP_ListOfTripletsNodes	agregates
	WOSP_ListOfTripletsNodes	agregates
WOSP_ListOfTripletsNode	WOSP_Triplet	agregates
	WOSP_ListOfTripletsNode	agregates
	WOSP_ListOfTripletsNode	agregates
WOSP_SetupConnectionTreatment	WOSP_Connestion resultat	agregates
WOSRP_Pipe	WOS_Constants	implements
WOSRP_Client	WOS_Params	agregates
WOSRP_Server	WOS_Params	agregates
	WOSP_VersionWarehouse	agregates
WOSRP_RequestQueue	WOS_MessageQueue	extends
WOSRP_ServerTcpSocket	WOSRP_Server	agregates
	WOS_Params	agregates
	WOSP	agregates
	WOSP_VersionWarhouse	agregates
	WOSP_Parser	agregates
WOSRP_ReplyQueue	WOS_MessageQueue	extends

La définition des relations entre les classes nous permettra d'identifier les liens entre les classes dépendantes du langage WOSP/WOSRP et les classes qui ne doivent pas l'être. Les liens entre ces deux types de classes doivent être éliminé afin de pouvoir créer plusieurs implémentations de la couche de communication de WOS.

Afin de mieux voir ces dépendances et pouvoir définir cette nouvelle architecture nous allons présenter ces dépendances sous forme de graphes dans la section suivante.

5.3 Description des relations entre les classes

La figure 5.2 montre les différentes classes qui implémentent les interfaces identifiées à la section 4.2.2.

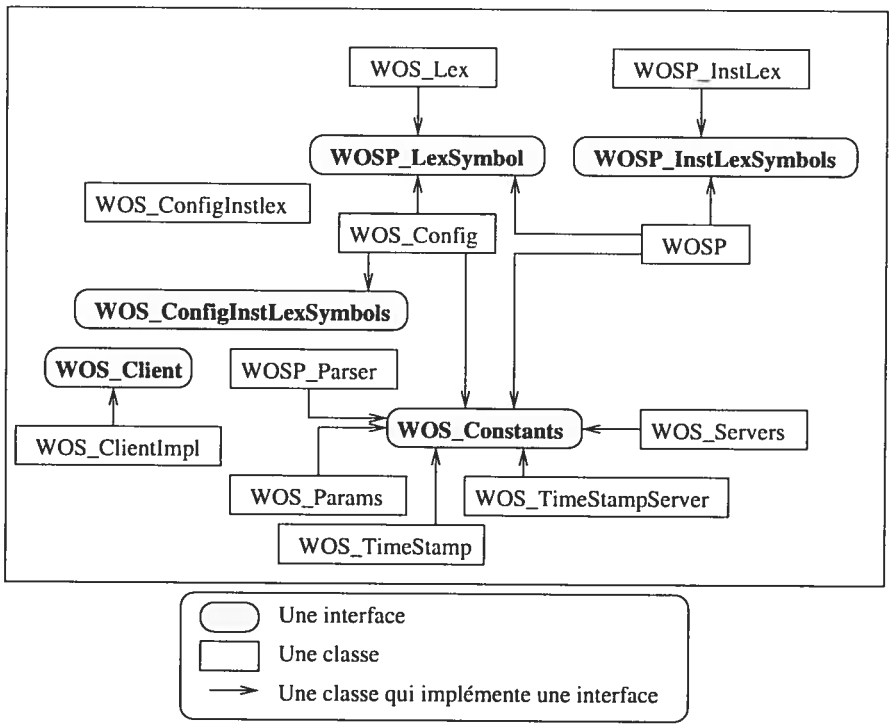


FIG. 5.2 – Schéma des relations d'implémentation entre les classes et les interfaces.

La figure 5.3 montre les relations d'héritage entre les classes. Par exemple une flèche qui va de la classe «**WOSP_InstLex**» vers la classe «**WOS_Lex**» indique que la première classe hérite de la deuxième.

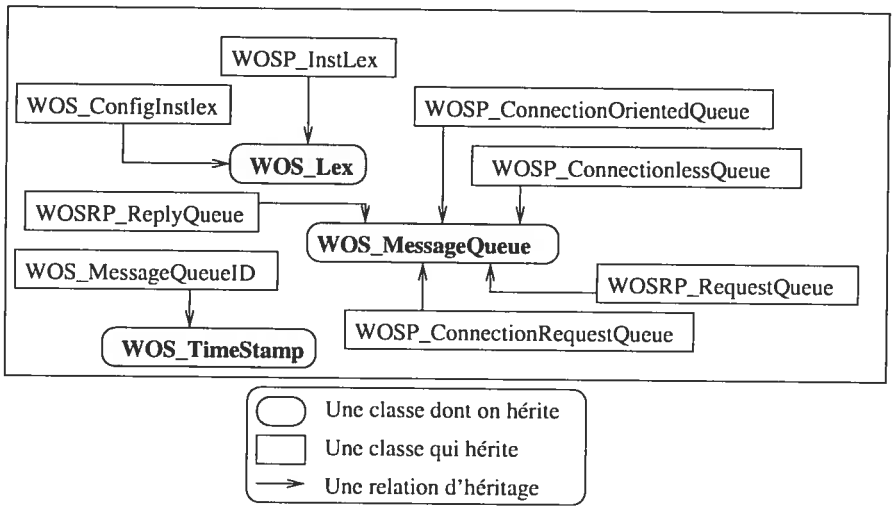


FIG. 5.3 – Schéma des relations d'héritage entre les classes.

La figure 5.4 illustre les relations d'utilisation entre les classes. Par exemple une flèche qui va de la classe «**WOS_Message**» vers la classe «**WOSP_ListOfTriplet**» indique que dans la classe «**WOS_Message**» on définit un objet de type «**WOSP-_ListOfTriplet**».

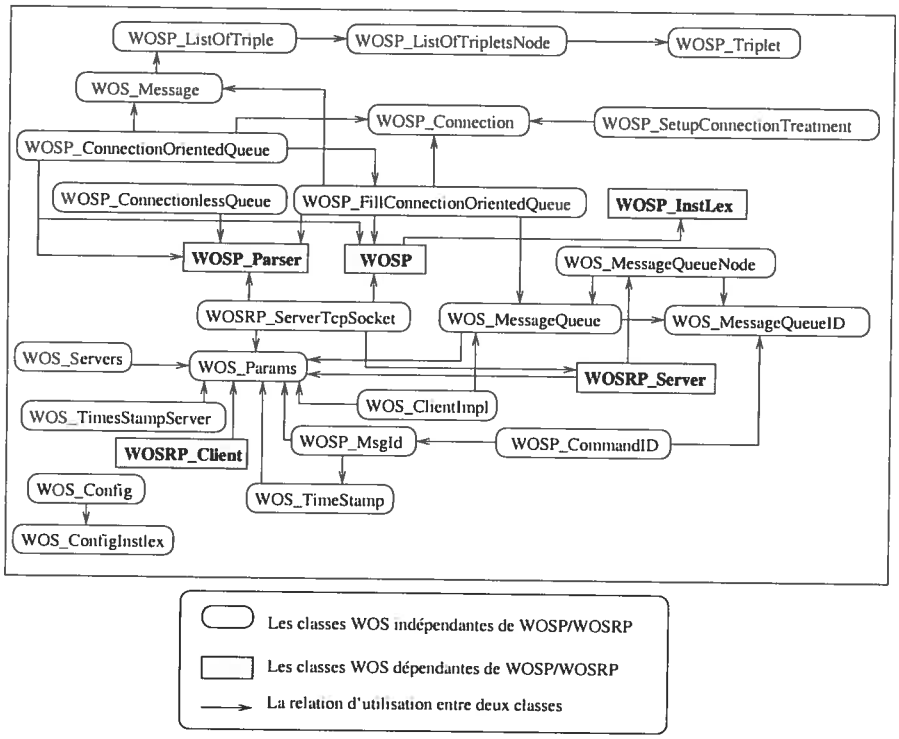


FIG. 5.4 – Schéma d'utilisation entre les classes

Les relations qui doivent être éliminées afin de permettre plusieurs implémentations de la couche de communication de WOS sont celles qui lient les classes indépendantes des langages WOSP et WOSRP avec les classes dépendantes de ces langages. Le graphe 5.5 Montre ces relations.

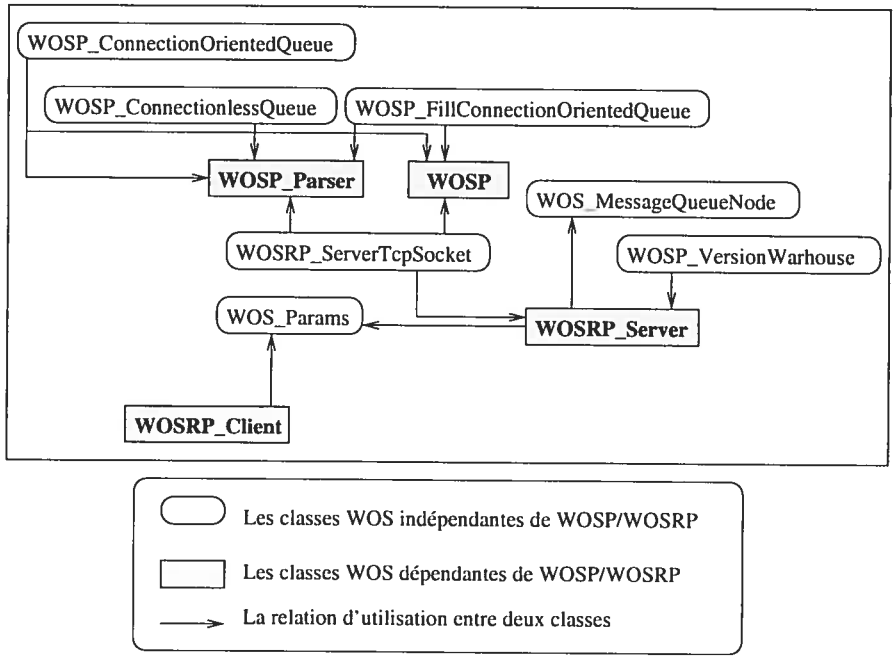


FIG. 5.5 – Schéma des relations entre les classes dépendantes et les classes indépendantes du langage WOSP/WOSRP.

Après avoir identifié les relations entre les classes qui implémentent les protocoles WOSP/WOSRP et celles qui sont indépendantes, nous allons proposer une nouvelle architecture pour éliminer ces dépendances. En effet de nouvelles classes abstraites seront créées comme l'indique la figure 5.6

Dans cette nouvelle architecture, deux classes abstraites sont définies. Soit **«WOS_Protocol»** et **«WOS_RequestProtocol»**.

La classe **«WOSP_WOSPImpl»** remplacera la classe **«WOSP_Parser»** et héritera de la classe **«WOS_Protocol»**. Elle servira de point d'accès à la classe **«WOSP»**.

Puisque le nœud WOS peut être client et serveur au même temps, nous n'avons pas besoin de deux classes **«WOSRP_Client»** et **«WOSRP_Server»**. La classe **«WOSRP_WORPImpl»** représente la fusion des deux. Cette nouvelle classe héritera de la classe abstraite **«WOS_RequestProtocol»**.

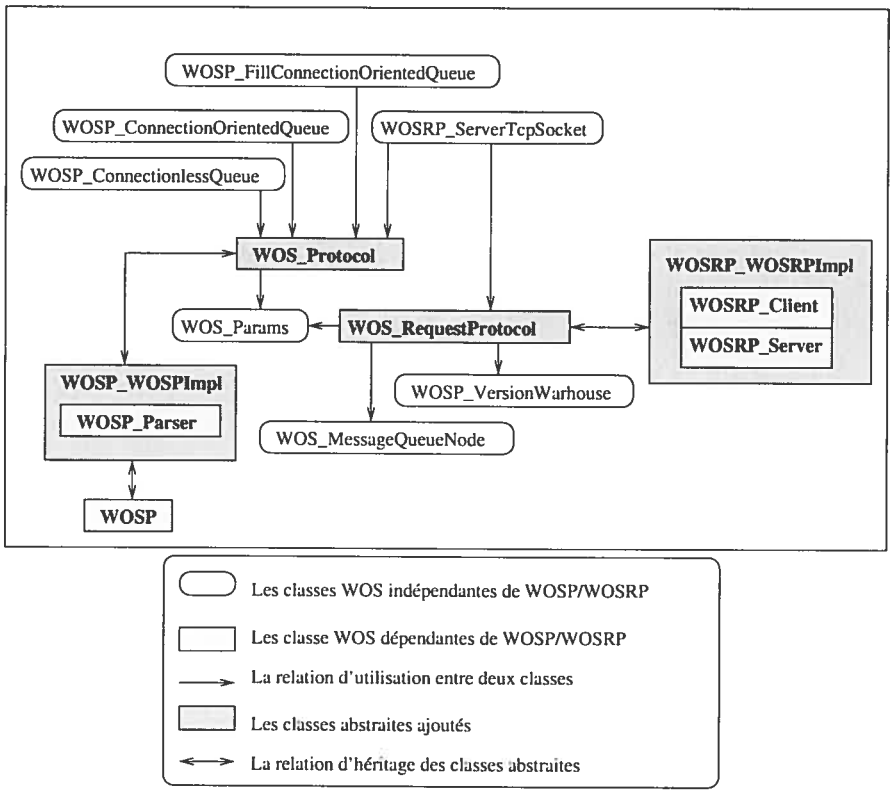


FIG. 5.6 – Proposition d'architecture pour les classes WOSP/WOSRP

5.4 Conclusion

Dans ce chapitre, nous avons effectué une analyse détaillée de l'ancienne implémentation de la couche de communication de WOS. Cette analyse nous a permis de d'identifier les changements à apporter afin de développer la nouvelle implémentation de WOS.

Chapitre 6

Implémentation

Dans ce chapitre nous expliquons la nouvelle implémentation en HTTP/XML. Nous commençons par montrer les méthodes de l'interface de la couche de communication, ainsi que leur utilisation dans les différents types de messages. Après nous expliciterons l'exemple d'envoi de messages dans le mode sans connexion. Finalement nous effectuons les scénarios de tests.

6.1 Implémentation des applications client et serveur

Afin de mieux comprendre l'utilité des différentes méthodes de l'interface selon le type de message utilisé et les traitements appliqués nous avons regroupé les informations nécessaires sous forme de tableaux. Chaque tableau décrit un type de message et les méthodes utilisées afin d'envoyer ou de recevoir ce message, de s'enregistrer ou d'annuler l'enregistrement dans une queue qui lui est spécifique.

6.1.1 Envoi de messages WOSP

Dans cette partie nous décrivons les applications client et serveur qui utilisent les méthodes de l'interface pour envoyer et recevoir des messages WOSP.

Pour l'envoi de message WOSP deux modes sont possibles, le mode avec connexion et le mode sans connexion.

Pour envoyer ou recevoir un message WOSP en mode hors connexion il faut passer par trois étapes, à savoir :

TAB. 6.1 – Les méthodes utilisées pour une requête de connexion WOSP

	Requête de connexion WOSP
Envoi de messages	WOSP_SetupConnection(IP, port, VersionID) WOSP_SetupConnection(IP, port, VersionID, timeout) WOSP_SetupConnection(IP, VersionID) WOSP_SetupConnection(IP, VersionID, timeout)
Enregistrement	WOS_RegisterConnectionServer(VersionID)
Annulation de l'enregistrement	WOS_Unregister(mqid)
Réception de messages	WOSP_AcceptConnection(mqid) WOSP_RejectConnection(mqid) WOS_GetMessage(mqid) WOS_WaitForMessage(mqid) WOS_WaitForMessage(mqid, timeout)

TAB. 6.2 – Les méthodes utilisées pour un message WOSP en mode sans connexion

	Message WOSP en mode sans connexion
Envoi de messages	WOSP_SendMessage(IP, port, versionID, message) WOSP_SendMessage(IP, versionID, message) WOSP_SendMessage(mqid, IP, port, message) WOSP_SendMessage(mqid, IP, message)
Enregistrement	WOS_RegisterConnectionlessServer(VersionID)
Annulation de l'enregistrement	WOS_Unregister(mqid)
Réception de messages	WOS_GetMessage(mqid) WOS_WaitForMessage(mqid) WOS_WaitForMessage(mqid, timeout)

TAB. 6.3 – Les méthodes utilisées pour un message WOSP en mode connexion

	Message WOSP en mode connexion
Envoi de messages	WOSP_SendMessage(String mqid, WOSP_ListOfTriplets message) WOSP_SendMessage(String mqid, WOSP_ListOfTriplets message, boolean doClose)
Réception de messages	WOS_GetMessage(String mqid) WOS_WaitForMessage(String mqid) WOS_WaitForMessage(String mqid, int timeout)

TAB. 6.4 – Les méthodes utilisées pour une requête WOSRP

	Requête WOSRP
Envoi de messages	WOSRP_Request(InetAddress RecipientIP, int RecipientPort, boolean SpokenOrKnown, boolean SpecificOrAny, short HopCount, String VersionID)
Enregistrement	WOS_RegisterReplyServer()
Annulation de l'enregistrement	WOS_Unregister(String mqid)
Réception de messages	WOS_GetMessage(String mqid) WOS_WaitForMessage(String mqid) WOS_WaitForMessage(String mqid, int timeout)

TAB. 6.5 – Les méthodes utilisées pour une réponse WOSRP

	Réponse WOSRP
Envoi de messages	WOSRP_Reply(InetAddress RecipientIP, int RecipientPor, InetAddress ServerIP, int ServerPort, boolean SpokenOrKnown, String VersionID)
Enregistrement	WOS_RegisterReplyServer()
Annulation de l'enregistrement	WOS_Unregister(String mqid)
Réception de messages	WOS_GetMessage(String mqid) WOS_WaitForMessage(String mqid) WOS_WaitForMessage(String mqid, int timeout)

1. Se connecter à l'interface pour pouvoir utiliser ses méthodes, et cela dans le cas du client et du serveur.
2. S'enregistrer à la queue qui contient les messages en mode hors connexion «WOSRP Connectionless Queue».
3. Le client envoie un message et se met en attente. Dès que le serveur reçoit ce message, il retourne une réponse au client.

L'envoi ou la réception de messages en mode avec connexion se fait selon ces étapes :

1. Se connecter à l'interface.
2. S'enregistrer à la queue «WOSRP Connection Request Queue».
3. Le client envoie une demmande de connexion et aatend la réponse du serveur.
4. Ils s'enregistre à la queue «WOSP Connection Oriented Queue» pour pouvoir envoyer et recevoir des messages.

6.1.2 Implémentation du modèle HTTP/XML

Tout d'abord nous allons définir les étapes d'envoi d'un message dans le modèle HTTP/XML. La figure 6.1 présente ces différentes étapes.

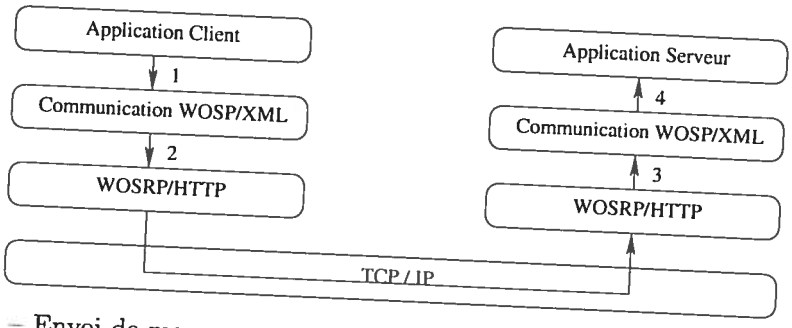


FIG. 6.1 – Envoi de messages entre client et serveur dans le modèle HTTP/XML

Les étapes d'envoi et de réception des messages dans le modèle HTTP/XML sont :

- **Etape 1** : le message est construit au niveau de la couche application du client sous forme de triplets est envoyé ensuite à la couche communication qui le transforme en un format XML.
- **Etape 2** : la couche communication envoie le message à la couche WOSRP/HTTP qui l'encapsule dans un en-tête WOSRP et l'envoie dans le support de communication.
- **Etape 3** : une fois le message WOSP reçu au niveau du serveur, il sera retiré et envoyé à la couche communication.
- **Etape 4** : la couche communication transforme le message du format XML au format triplets pour l'envoyer à son tour à la couche application du serveur.

Afin de réaliser ce scénario nous avons besoin de deux analyseurs syntaxiques «parser». Un premier qui va convertir le message du format triplets au format XML, et un deuxième qui va convertir le message du format XML au format triplets.

Après avoir défini ces différentes étapes, nous pouvons maintenant présenter les changements qui ont été faits au niveau des méthodes d'envoi et de réception des messages afin qu'elles puissent supporter les modèles HTTP/XML. Nous montrerons les changements et les scénarios de tests dans le mode hors connexion.

L'envoi de message en mode hors connexion se fait selon la figure 6.2.

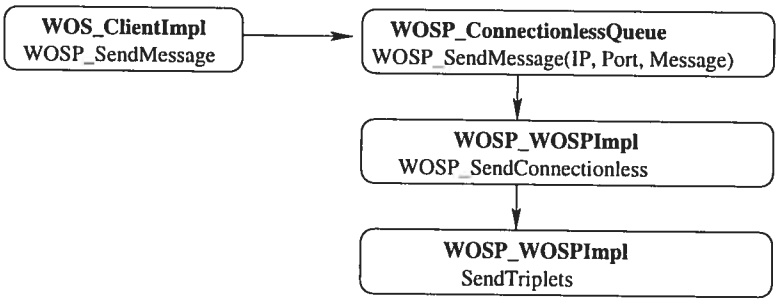


FIG. 6.2 – les appels de méthode pour l'envoi des messages en mode sans connexion

Les appels des méthodes dans la figure 6.2 se font comme suit :

1. L'interface client fait appel à la méthode **WOSP_SendMessage** pour envoyer un message au serveur.
2. La méthode **WOSP_SendMessage** de l'interface fait appel à son tour à la méthode **WOSP_SendMessage** de la classe **WOSP_ConnectionlessQueue** qui envoie le message à la queue appropriée.
3. La méthode **WOSP_SendMessage** fait appel à la méthode **WOSP_SendConnectionless** de la classe **WOSP_WOSPImpl** qui utilise la méthode **WOSP_SendTriplets** pour convertir le triplet dans le format approprié de WOS avant de l'envoyer.

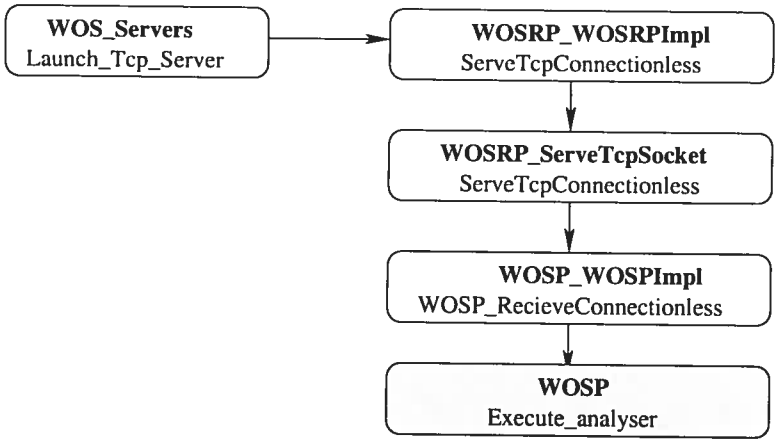


FIG. 6.3 – les appels des méthodes pour la réception des messages en mode sans connexion

La figure 6.3 illustre l'appel des méthodes pour la réception des messages en mode hors connexion.

1. Le serveur TCP est lancé à l'aide de la méthode **launch_Tcp_Server** de la classe **WOS_Servers**. Cette méthode fait appel à la méthode **ListenOnTcp-Port** de la classe **WOSRP_WOSRPImpl** qui initialise la connexion TCP avec le client WOSRP dans le port spécifique.
2. La méthode **ListenOnTcpProt** fait appel à la méthode **ServeTcpConnectionless** de la classe **WOSRP_ServeTcpSocket** qui fournit les services en mode sans connexion.
3. La méthode **ServeTcpConnectionless** fait appel à la méthode **WOSP_-RecieveConnectionless** de la classe **WOSP_WOSPImpl** qui traite le message reçu en mode hors connexion. Ce message est analysé à l'aide de la méthode **Execute_analyser** de la classe **WOSP**.

Afin d'implémenter le modèle WOSP/XML, des changements ont été apportés à la méthode **SendTriplet** de la classe **WOSP_WOSPImpl** pour l'envoie de message ainsi que la méthode **execute_analyser** de la classe **WOSP** pour la réception de message, et ce pour convertir un message WOSP du format triplets au format XML, et vice versa. Ces changements sont propagés dans toutes les autres méthodes dépendantes.

6.2 Les tests

Les tests sont effectués pour un modèle de communication en mode sans connexion à l'aide de quatre scénarios.

1. L'envoi de messages entre deux nœuds WOS qui utilisent l'implémentation originale de WOS (figure 6.4).
2. L'envoi de messages entre deux nœuds WOS qui utilisent la nouvelle l'implémentation de WOS en XML/HTTP (figure 6.5).
3. L'envoi de messages se fait entre un nœud client qui utilise l'implémentation de WOS en HTTP/XML et un nœud serveur qui utilise l'implémentation originale de WOS (figure 6.6).
4. L'envoi de messages se fait entre un nœud client qui utilise l'implémentation

originale de WOS et un nœud serveur qui utilise l'implémentation de WOS en HTTP/XML (figure 6.7).

À la réception d'un message par la couche de communication de WOS. Un test sera effectué sur le champ «WOSP protocol ID» afin de déterminer si le message est écrit en utilisant l'ancienne implémentation de WOS, ou bien il est en XML. Selon le type de message l'analyseur syntaxique approprié sera utilisé.

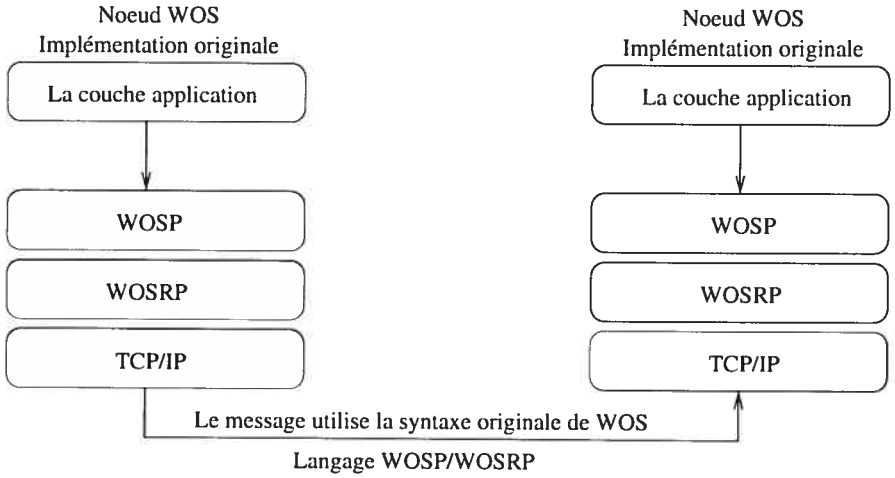


FIG. 6.4 – Envoi des messages entre deux nœuds avec l'implémentation originale de WOS

Dans le premier test (figure 6.4) nous avons envoyé un message WOSP en mode sans connexion entre deux machines avec l'implémentation originale de WOS. Le message est acheminé de la machine source à la machine destination en utilisant la syntaxe originale de WOS.

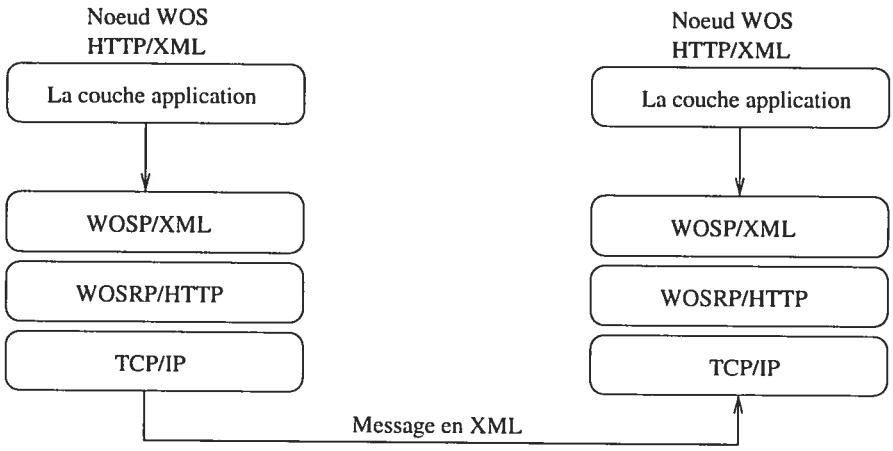


FIG. 6.5 – Envoi des messages entre deux nœuds avec la nouvelle implémentation de WOS

Dans le deuxième test (figure 6.5) nous avons envoyé un message WOSP en mode sans connexion entre deux machines avec la nouvelle implémentation de WOS. Les messages sont envoyés sous un format XML.

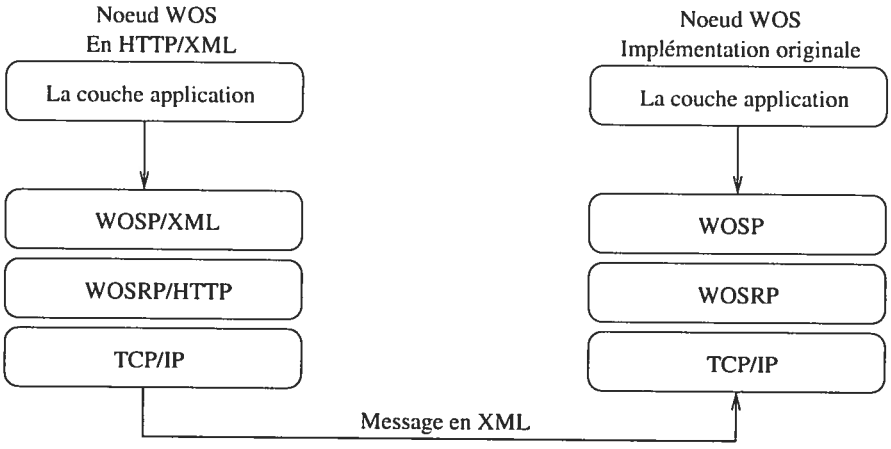


FIG. 6.6 – Envoi des messages entre deux nœuds d'implémentation différente

Dans le troisième test (figure 6.6) nous avons effectué une communication en mode sans connexion entre une machine source qui utilise la nouvelle implémentation de WOS vers une machine destination qui utilise l'ancienne implémentation. Les messages sont envoyés sous un format XML.

Dans le quatrième test (figure 6.7) nous avons effectué une communication en mode sans connexion entre une machine source qui utilise l'ancienne implémentation de WOS vers une machine destination qui utilise la nouvelle implémentation. Les messages sont envoyés en utilisant la syntaxe originale de WOS.

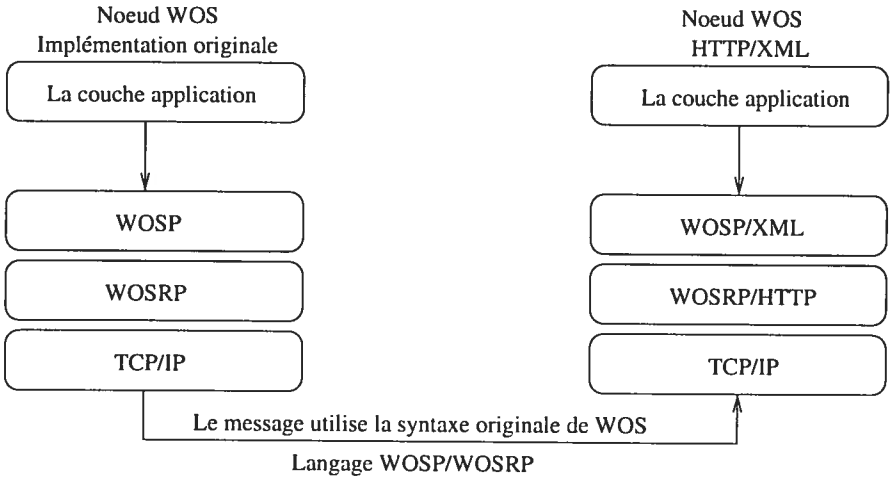


FIG. 6.7 – Envoi des messages entre deux nœuds d'implémentation différente

Après avoir présenté les tests de communication entre les deux implémentations de WOS, nous allons comparer le temps d'envoi des messages entre ces deux dernières. Un premier test consiste à illustrer l'évolution du temps en fonction du nombre de messages envoyés (figure 6.8). Le second test montre l'évolution du temps en fonction de la taille des messages envoyés (figure 6.9).

Dans les deux graphes (figure 6.8, 6.9) nous constatons que le temps d'envoi des messages dans la version utilisant la syntaxe originale de WOS et meilleur que celui de la version utilisant XML. Cela peut être justifier par l'utilisation d'une syntaxe XML qui augmante la taille du message ainsi que par l'utilisation de l'analyseur syntaxique XML qui prend plus de temps pour convertir les messages.

En effet dans cette nouvelle implémentation, WOS a gagné en terme d'utilisation des standards Internet au détriment de sa performance en terme de temps.

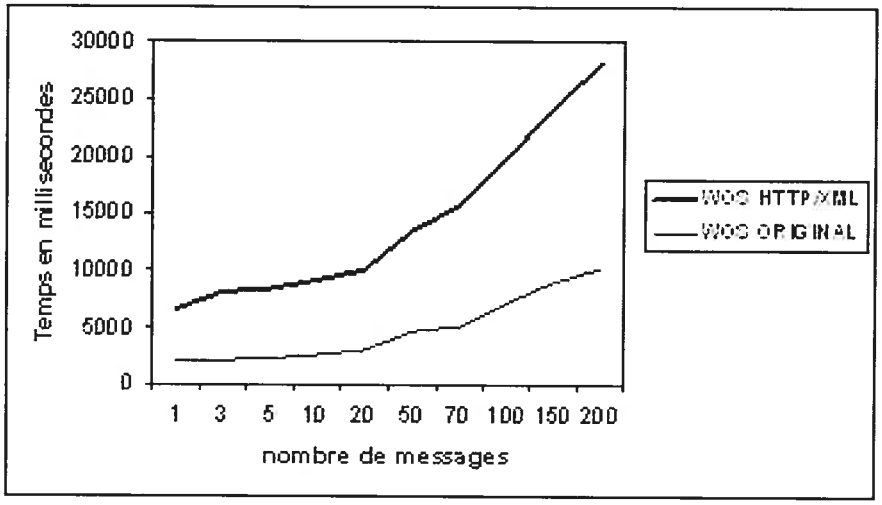


FIG. 6.8 – L'évolution du temps en fonction du nombre de messages envoyés

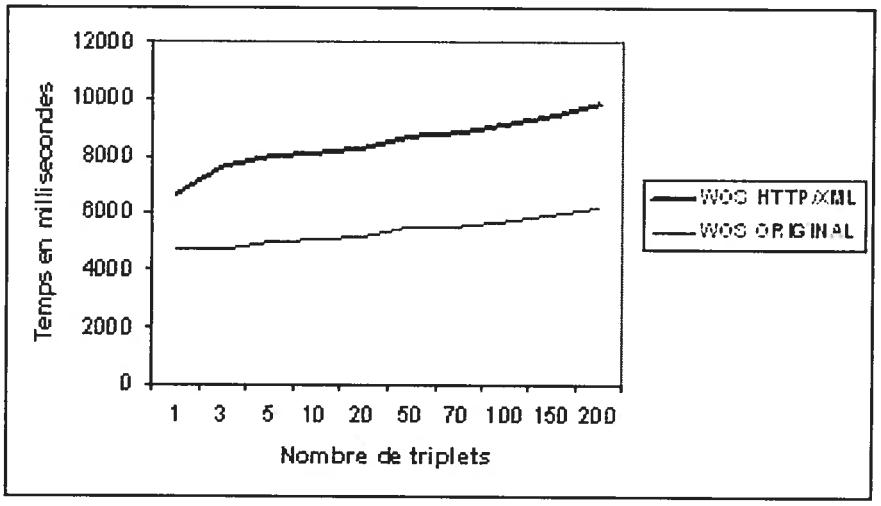


FIG. 6.9 – L'évolution du temps en fonction de la taille des messages envoyés

6.3 Conclusion

Dans ce chapitre nous avons expliqué la nouvelle implémentation. En particulier nous avons cité les classes et les méthodes impliquées pour l'envoi et la réception des messages dans le mode hors connexion. Ensuite nous avons montré les différents scénarios de test de communication entre deux nœuds WOS. Finalement nous avons effectué des tests de comparaison entre les performances des deux implémentations de WOS.

Conclusion

Dans ce mémoire, notre étude a porté sur des protocoles de communication appropriés afin d'échanger d'une manière plus facile les informations entre des partenaires du commerce électronique. Dans la plupart des cas, le contexte de tels échanges est fortement sensible. Ceci étant la couche de communication doit offrir des protocoles assez flexibles. Une façon prometteuse pour participer à ce dynamisme est l'utilisation de la communication versionnée. En appliquant ce concept, WOS (Web Operating System) a été développé afin de permettre aux différents utilisateurs d'accéder à des ressources ou d'échanger des informations sous forme de requêtes de services et cela sans avoir une connaissance préalable sur les services demandés à savoir, leur disponibilité, leurs coûts et sous quelles contraintes peut-on les avoir. L'objectif de ce travail était de changer l'infrastructure de la couche de communication de WOS afin de la rendre conforme aux standards actuels d'Internet, pour obtenir une meilleure compatibilité et une meilleure flexibilité.

Nous avons pu atteindre cet objectif lorsque nous avons changé la structure de l'implémentation des deux protocoles WOSP et WOSRP en utilisant les standards d'Internet en l'occurrence, HTTP, MIME et XML. Notre travail consistait à :

- Changer la structure de la couche de communication afin qu'elle puisse supporter plusieurs types d'implémentation, à savoir l'ancienne et la nouvelle. Pour cela nous avons séparé entre les classes qui dépendent du langage propre à WOSP/WOSRP et celles qui doivent être indépendantes.
- Nous avons changé la structure de l'en-tête WOSRP pour qu'elle puisse prendre en considération la nouvelle implémentation ainsi que des nouvelles extensions, tel que l'utilisation de IPv6 au lieu de IPv4.
- Finalement nous avons effectué des tests de communication ainsi que des tests de performance entre deux nœuds WOS utilisant ces deux implémentations.

les résultats ont montré que les performances en terme de temps d'envoi des messages dans la version utilisant la syntaxe originale de WOS sont meilleurs que ceux de la version utilisant XML. Cela peut être justifier par l'utilisation d'une syntaxe XML

qui augmante la taille du message ainsi que par l'utilisation de l'analyseur syntaxique XML qui prend plus de temps pour convertir les messages. Cependant, nous pourrions dir que WOS a gagné en terme d'utilisation des standards Internet au détriment de sa performance en terme de temps.

Après la réalisation de ce projet, nous pouvons maintenant confirmer que WOS peut utiliser la plateforme du Web pour communiquer avec d'autres systèmes distribués. Il reste encore plusieurs aspects à développer dans WOS en général et la couche communication en particulier. Nous pouvons citer :

- La gestion des problèmes de connexion
- La gestion des problèmes de débordement des queues de message.

Comme perspectives futures, nous pourrions par la suite envisager d'intégrer WOS au projet JXTA ou CORBA.

Bibliographie

[1] A.K.Lance (2001). *Internet Peer-to-Peer Technology Pricing Schemes as Seen Through the Lens of Principal Agent Theory, Using Napster as an Example Application.* http ://citeseer.nj.nec.com/468639.html.

[2] A.W.Télécommunication (2002). *Le modèle peer-to-peer.* http ://www.awt.be/tel/fic/t11.pdf

[3] B. Krishnamurthy et J. Rexford (2001), *Web Protocols and Practice, HTTP/1.1, networking Protocols, Caching, and Traffic Measurement.* Première édition, Addison Wesley.

[4] B. Wilson (2002), *JXTA Book*, http ://www.brendonwilson.com/projects/jxta-/pdf/jxta01.pdf.

[5] CORBA (2002). *CORBA.* www.corba.org & www.omg.org.

[6] D. H. Crocker (1996). *RFC 2045, MIME Part One : Format of Internet Messages Bodies.* http ://www.ietf.org/rfc.

[7] D. H. Crocker (1982). *RFC 822, Standard for the Format of ARPA Internet Text Messages.* http ://www.ietf.org/rfc.

[8] D. Serain (1999)*Middleware et Internet CORBA, COM/DCOM, JavaRMI, java Beans, ActiveX.* 2^e édition, InterEditions.

[9] G. Babin, H. Coltzau, M. Wulff et S. Ruel (2000). *Application Programming Interface for WOSP/WOSRP.*P.Kropf et al., editor, LNCN 1830, Distributed Communities on the Web, LNCN 1830, pp. 110-121. Springer 2000.

[10] G. Babin (1998). *Requirements for the implementation of WOS^{TM}.* Distributed Computing on the Web Workshop (DCW 98), Rostock, Germany, pp. 129-133.

[11] G. Babin, P. Kropf et H. Unger (1998). *A Two-Level Communication Protocol for a Web Operating System (WOS^{TM})*. IEEE Euromirco Workshop on Network Computing, Västerås, Sweden, pp. 939-944.

[12] I. S. Graham et L. Quin (1999). *XML Specification Guide.* Wiley computer publishing.

[13] J. F. Bobier (2001).*Microsoft.Net : architecture et services.* http ://www.developpez.biz/downloads/dotnet/MemoireDotNet.pdf.

[14] J. Plaice et P. Kropf (1999) *WOS Communities - interactions and relations between entities in distributed systems.* Distributed Computing on the Web (DCW99), Rostock, Germany, pp. 163-166.

[15] J. Vaucher, G.Babin, P.Kropf et T.Jouve (2002). *Experimenting with Gnutella Communities.* Distributed Communities on the Web (DCW2002), Sydney, Australie. LNCS 2468, Springer Berlin, pp. 85-99.

[16] L. Gong (2002). *Project JXTA : A Technology Overview.* http ://www.jxta.org/project/www/docs/jxtaview_01nov02.pdf.

[17] Microsoft.net (2001). *Microsoft.Net.* msdn.microsoft.com.

[18] M.Ripeanu, I.Foster (2002). *Mapping the Gnutella Network.* http ://www.cs.rice.edu/Conferences/IPTPS02/128.pdf.

[19] M. Wulff, G. Babin, P. Kropf et H. Zhong (1999). *Communication in the Web Operating System (WOS).* Research report diul-rr-9902, Université Laval, Sainte-Foy, Québec, Canada.

[20] P.Kropf, H.Unger et G.Babin (2000). *WOS : an Internet Computing Environment.* Workshop on Ubiquitous Computing, IEEE International Conference on parallel Architecture and Compilation Techniques. Philadelphia, PA, pp. 1422

[21] P. Kropf (1999). *Overview of the WOS project.* Advanced Simulation Technologies Conferences (ASTC 99), San Diego, CA, USA, pp. 939-944.

[22] R. Fielding, J. Gettys, J. Mogul, H. Feystuk, L. Masinter, P. Leach et T.Berners-Lee (1999). *RFC 2616 Hypertext Transfert Protocol - HTTP/1.1.* http ://www.ietf.org/rfc.

[23] R. Fielding, J. Gettys, J. Mogul, H. Frystyk et T.Berners-Lee (1997). *RFC 2068 Hypertext Transfert Protocol – HTTP/1.1.* http ://www.ietf.org/rfc.

[24] S. Ben lamine, J. Plaice et P. Kropf (1997). *Problems of Computing on the WEB.* High performance computing symposium, A.Tentner,ed., The Society for Computer Simulation International, Atlanta, GA, pp. 296-301.

[25] Microsystems (2002). *Java technology and XML.* Sun Microsystems, java.sun.com/xml

[26] Microsystems (2001).*SunOne.* www.sun.com/SunOne.

[27] SOAP (2002). *SOAP1.1.* www.w3.org/TR/SOAP et msdn.microsoft.com.

Annexe A

Principales méthodes de la couche de communication de WOS

A.1 WOS_ClientImpl

La classe WOS_ClientImpl se compose des méthodes suivantes :

1. WOS_GetMessage(MQID) : cette méthode permet d'avoir le prochain message valable sur la queue identifiée par son MQID.
2. WOS_WaitForMessage(MQID) : Cette méthode attend indéfiniment le prochain message valable dans la queue de messages.
3. WOS_WaitForMessage(MQID, timeout) : cette méthode attend pour le prochain message un temps donné (timeout). Elle retourne ou bien le message ou bien nul.
4. WOS_RegisterRequestServer() : cette méthode permet à une application de s'enregistrer en tant que «WOSRP Request Manager». Dans un nœud WOS, une seule application peut être enregistrée.
5. WOS_RegisterReplyServer() : cette méthode permet à une application de s'enregistrer en tant que «WOSRP Reply Manager». Dans un nœud WOS, une seule application peut être enregistrée.
6. WOS_RegisterCennectionlessServer(VersionID) : cette méthode permet à une application de s'enregistrer entant que «WOSP connectionless message manager ».

7. WOS_RegisterCennectionServer(VersionID) : cette méthode permet à une application de s'enregistrer en tant que «WOSP connectionless message manager».
8. WOS_Unregister(MQID) : cette méthode permet à une application d'annuler son enregistrement en tant que gestionnaire de queue de message.
9. WOSRP_Request(RecipientIP, RecipientPort, SpokenOrKnown, SpecificOrAny, HopCount, VersionID) : cette méthode est utilisée pour envoyer les requêtes WOSRP. Le message ne sera pas envoyé si un de ces paramètres est invalide.
10. WOSRP_Reply(RecipientIP, RecipientPort, ServerIP, ServerPort, SpokenOrKnown, VersionID) : cette méthode est utilisée pour envoyer les réponses WOSRP. Le message ne sera pas envoyé si un de ces paramètres est invalide.
11. WOSP_SetupConnection(IP, port, VersionID) : cette méthode est utilisée pour initialiser une connexion WOSP ; Elle attend indéfiniment pour une réponse.
12. WOSP_SetupConnection(IP, port, VersionID, timeout) : cette méthode est utilisée pour initialiser une connexion WOSP, elle attend un temps donné «timeout» pour une réponse.
13. WOSP_SetupConnection(IP, VersionID) : cette méthode est utilisée pour initialiser une connexion WOSP en utilisant le port WOSRP par défaut, elle attend indéfiniment pour une réponse.
14. WOSP_SetupConnection(InetAddress IP, String VersionID, int timeout) : Cette méthode est utilisée pour initialiser une connexion WOSP en utilisant le port WOSRP par défaut, elle attend un temps donné «timeout» pour une réponse.
15. WOSP_AcceptConnection(MQID) : cette méthode permet au gestionnaire de queue de requête de connexion WOSP de confirmer l'acceptation de la connexion WOSP.
16. WOSP_RejectConnection(MQID) : cette méthode permet au gestionnaire de queue de requête de connexion WOSP de rejeter une connexion WOSP.
17. WOSP_SendMessage(IP, port, versionID, message) : cette méthode est utilisée afin de transmettre des messages WOSP en mode sans connexion.
18. WOSP_SendMessage(IP, versionID, message) : cette méthode est utilisée afin

de transmettre des messages WOSP en mode sans connexion en utilisant le port par défaut du serveur WOSRP.

19. WOSP_SendMessage(MQID, IP, port, message) : cette méthode est utilisée afin de transmettre des messages WOSP en mode sans connexion pour une version WOSP donnée.
20. WOSP_SendMessage(MQID, IP, message) : cette méthode est utilisée afin de transmettre des messages WOSP en mode sans connexion pour une version WOSP donnée, et en utilisant le port par défaut du serveur WOSRP.
21. WOSP_SendMessage(mqid, message) : cette méthode est utilisée afin de transmettre des messages WOSP en mode connexion.
22. WOSP_SendMessage(mqid, message, doClose) : Cette méthode est utilisée afin de transmettre des messages WOSP en mode connexion. La variable «doClose »est mise à vrai pour confirmer ou demander l'arrêt de la connexion.

A.2 WOSRP_ReplyQueue

Les méthodes définies dans cette classe sont :

1. exists() : cette méthode détermine si une instance de cette queue est déjà créée. Puisqu'une seule queue de ce type peut exister pour un nœud donné, elle retourne oui si elle est déjà créée et non autrement.
2. register(params) : elle est utilisée pour enregistrer une application en tant que gestionnaire de queue de réponse WOSRP «WOSRP reply manager server».
3. unregister() : cette méthode est utilisée pour annuler l'enregistrement d'un serveur de queue de réponse WOSRP «WOSRP reply manager server».
4. launchServer() : cette méthode permet de lancer un «WOSRP reply manager».
5. bindQueue(params) : cette méthode permet d'assure que la queue recherchée existe et qu'elle est servie. Elle retourne une référence à cette queue.
6. SendMessage(RecipientIP, RecipientPort, InetAddr, ServerPort, SpokenOrKnown, VersionID) : cette méthode est utilisée par le serveur de queue pour envoyer une réponse WOSRP (WOSRP_Reply) à un autre nœud WOS.

A.3 WOSRP_RequestQueue

Les méthodes définies dans cette classe sont :

1. exists() : cette méthode détermine si une instance de cette queue est déjà créée. Puisqu'une seule queue de ce type peut exister pour un nœud donné, elle retourne oui si elle est déjà créée et non autrement.
2. register(params) : cette méthode est utilisée pour enregistrer un gestionnaire de queue de requêtes WOSRP «WOSRP request manager server».
3. unregister() : cette méthode est utilisée pour annuler l'enregistrement d'un gestionnaire de queue de requête WOSRP «WOSRP request manager server».
4. launchServer() : cette méthode permet de lancer un «WOSRP request manager».
5. bindQueue(params) : cette méthode assure que la queue recherchée existe et qu'elle est servie. Elle retourne une référence à cette queue.
6. SendMessage(RecipientIP, RecipientPort, InetAddr, ServerPort, SpokenOrKnown, VersionID) : cette méthode est utilisée par le serveur de queue pour envoyer une réponse WOSRP (WOSRP_Reply) à un autre nœud WOS.

A.4 WOSP_Connection

Les méthodes utilisées dans cette classe sont :

1. getIn() : cette méthode permet d'obtenir une référence au flot d'entrée.
2. getOut() : cette méthode permet d'obtenir une référence au flot de sortie.
3. getSocket() : cette méthode permet d'obtenir une référence au socket correspondant à la connexion .
4. SetSocket() : cette méthode permet d'assigner la référence au socket correspondant à la connexion.

A.5 WOSP_ConnectionOrientedQueue

Les méthodes utilisées dans cette classe sont :

1. sendMessage(list, doClose) : cette méthode est utilisée pour envoyer un message en mode sans connexion.
2. accept() : cette méthode est utilisée par le gestionnaire de la queue de requêtes de connexion WOSP pour accepter une requête de connexion. Cette méthode informe le serveur que la connexion peut être établie.
3. reject() : cette méthode est utilisée par le gestionnaire de requêtes de connexion WOSP pour rejette une requête de connexion. Cette méthode ferme le socket entre les deux serveurs.
4. fillQueue() : cette méthode lance le processus qui sera chargé de remplir la queue de message locale.
5. getConnection() : cette méthode retourne la valeur correspondante à la connexion.
6. getMsgAnalyser() : cette méthode retourne la valeur du «msgAnalyser».
7. getParser() : cette méthode retourne la valeur de l'analyseur syntaxique «parser».
8. getMessage() : cette méthode retourne la valeur du message.
9. getDoClose() : cette méthode retourne la valeur de la variable «DoClose».
10. setDoClose() : cette méthode assigne la valeur de la variable «DoClose».
11. setClose() : cette méthode assigne la valeur de la variable «Close».

A.6 WOSP_ConnectionRequestQueue

Les méthodes utilisées dans cette classe sont :

1. sendMessage(RecipientIP, RecipienPort, versionID, timeout) : cette méthode est utilisée pour envoyer une requête de connexion WOSP.
2. launcheServer(VersionID) : cette méthode lance le serveur spécifique capable de servir la queue de requêtes de connexion «WOSP connection request manager» pour une version WOSP donnée.
3. bindQueue(params, VersionID) : cette méthode assure que la queue recherchée existe et qu'elle est servie. Elle retourne la référence à cette queue.

4. findFirstQueue(versionID) : cette méthode cherche la première queue spécifique à une version WOSP donnée. Si elle la trouve, elle retourne la référence à cette queue, sinon retourne la valeur nul.
5. exists(versionID) : cette méthode détermine s'il y a au moins une queue crée pour une version WOSP spécifique, elle retourne une valeur booléenne.
6. getVersionID() : elle retourne la valeur de la version WOSP utilisée (versionID).
7. register(params, versionID) : cette méthode est utilisée pour enregistrer une application en tant que gestionnaire de queue de requêtes de connexion «WOSP connexion request server».
8. enregister() : cette méthode est utilisée pour supprimer l'enregistrement d'une application enregistrée en tant que gestionnaire de requête de connexion «WOSP connection request server».

A.7 WOSP_ConnectionlessQueue

Les méthodes utilisées dans cette classe sont :

1. sendMessage(RecipientIP, RecipienPort, versionID, message) : Cette méthode est utilisée pour envoyer une requête de connexion WOSP.
2. launcheServer(VersionID) : cette méthode lance le gestionnaire de queue de message en mode sans connexion «WOSP connectionless message manager» pour une version WOSP spécifique.
3. bindQueue(params, VersionID) : cette méthode assure que la queue recherchée existe et qu'elle est servie. Elle retourne la référence à cette queue.
4. findFirstQueue(versionID) : cette méthode cherche la première queue spécifique à une version WOSP donnée. Si elle la trouve elle retourne la référence à cette queue, sinon elle retourne la valeur nul.
5. exists(versionID) : cette méthode détermine s'il y a au moins une queue créée pour une version WOSP spécifique, elle retourne une valeur booléenne.
6. getVersionID() : cette méthode retourne la valeur de la version WOSP.

7. register(params, versionID) : cette méthode est utilisée pour enregistrer une application en tant que gestionnaire de messages en mode sans connexion «WOSP connexionless message manager».
8. unregister() : cette méthode est utilisée pour annuler l'enregistrement d'une application autant que gestionnaire de messages en mode sans connexion «WOSP connectionless message manager».

A.8 WOSP_Parser

Les méthodes utilisées dans cette classe sont :

1. assignTmpFilename(params) : cette méthode génère un nom de fichier unique pour stocker les fichiers reçus avec les messages. Le répertoire par défaut est " WOS-TMPDIR ", et il est défini dans la configuration des fichiers de WOS.
2. createMsgId(params, wosPid) : cette méthode permet de créer l'identifiant d'un message.
3. checkTripletOrder(list) : cette méthode permet de vérifier quelques règles de base qui doivent être valides pour les triplets. Elle retourne une valeur booléenne.
4. printDataline(tripeltValue, PrintWriter, PrintOut) : cette méthode permet d'imprimer le contenu des données des triplets en ajoutant des caractères d'échappement.
5. sendTriplets(list, out, msgId, closeCommand) : cette méthode transforme la liste des triplets pour qu'elle soit envoyé sur le socket.
6. readChar(br, prevChar, index) : cette méthode permet de lire un caractère.
7. writeFilesFromMessage(params, br, prevChar, receivedList) : cette méthode permet de lire les fichiers du message et les stockés localement avant de lire le massage.
8. WOSP_SendConnectioless(params, serverIP, serverPort, WOS-MessageQueueID, list) : cette méthode permet d'envoyer un message en mode sans connexion.
9. WOSP_ReceiveConnctionless(WOS-Params, msgAnalyser, SenderIP, SenderPort, message) : cette méthode est appelée par le serveur pour recevoir les

messages en mode sans connexion.

10. WOSP_SetupConnection(serverIP, serverPort, VersionID, timeout) : cette méthode initialise la connexion pour le nœud WOS demandeur.
11. WOSP_SendConnection(params, out, list, WOSPID, closeCommand) : cette méthode permet d'envoyer les messages WOSP en mode orienté connexion.
12. WOSP_ReceiveConnection(params, msgAnalyser, br) : cette méthode permet de recevoir les messages WOSP dans le mode orienté connexion.

A.9 WOSRP_Client

Les méthodes utilisées dans cette classe sont :

1. buildWOSRPHeader(header[], options, hopCount, ServerIP, ServerPort, SenderIP, SenderPort, VersionId) : cette méthode est utilisée pour construire l'en-tête WOSRP.
2. WOSRP_Connectionless(WOS-Params, ServerIP, ServerPort, VersionID, Message) : cette méthode encapsule le message WOSP, initialise la connexion TCP et envoie l'entête WOSRP correspondante. Elle envoie le message en filtrant les caractères DEL. Tout envoi de message fini par l'envoi des deux caractères : (DEL) et (EOT).
3. WOSRP_SetupConnection(resultat, ServerIP, ServerPort, VersionID) : cette méthode permet d'envoyer la requête de connexion WOSP : elle initialise la connexion TCP, envoi l'en-tête WOSRP et attend un acquittement positif.
4. WOSRP_Request(params, RecipientIP, RecipientPort, SpokenorKnown, SpecificOrAny, HopCount, CersionID) : cette méthode permet d'envoyer les requêtes WOSRP, d'initialiser la connexion WOSP et d'envoie l'entête WOSRP.
5. WOSRP_Reply(params, RecipientIP, RecipientPort,SenderIP, SenderPort, SpokenorKnown, CersionID) : cette méthode permet d'envoyer les réponses WOSRP, d'initialise la connexion WOSP et envoie l'entête WOSRP.

A.10 WOSRP_Server

Les méthodes utilisées dans cette classe sont :

1. ProcessHeader(buf[]) : cette méthode traite l'information d'en-tête WOSRP reçu.
2. ListenOnTcpPort(int port) : cette méthode initialise la connexion TCP avec un client WOSRP dans un port spécifique.
3. ListenOnUdpPort(int port) : cette méthode initialise la connexion WOSP avec un client WOSRP dans un port spécifique.
4. ServeUdpPacket(DatagramPacket packet) : cette méthode traite l'information de l'en-tête WOSRP reçu , et selon l'option du service elle détermine si elle fournit un service «WOSRP-VM Request» ou un service «WOSRP-VM Reply».

A.11 WOSRP_ServerTcpSocket

Les méthodes utilisées dans cette classe sont :

1. ServerTcpConnectionless(in, header) : cette méthode assure le service en mode sans connexion.
2. ServerTcpConnection(BufferedReader in, PrintWriter out, DataoutputStream output, WOSMessage header) : cette méthode assure le service en mode connexion, envoie l'acquittement et prépare le message de demande de connexion WOSP.