Intégration de l'Internet 3G au sein d'une plate-forme active

Maroun Chamoun

École Doctorale
d'Informatique,
Télécommunications
et Électronique de Paris

Thèse

présentée pour obtenir le grade de docteur
de l'Ecole Nationale Supérieure des Télécommunications

Spécialité : **Informatique et Réseaux**

Maroun Chamoun

Intégration de l'Internet 3G au sein d'une plateforme active

Soutenue le 15 mai 2006 devant le jury composé de :

Dominique Gaiti	Président
Ken Chen	Rapporteurs
Omar Abou Khaled	
Pascal Urien	Examinateurs
Mikael Salaun	
Ahmed Serhrouchni	Directeur de Thèse

A Mon Epouse

REMERCIEMENTS

Je tiens à remercier toutes les personnes qui ont rendu cette thèse possible par leurs aides et leurs contributions.

Mes premiers remerciements sont adressés à Ahmed Serhrouchni, Maître de conférences à l'ENST et Directeur de cette thèse, qui m'a encadré avec enthousiasme, et a su me conseiller efficacement tout en me laissant travailler très librement. Pour cela, je lui suis très reconnaissant.

Mes profonds remerciements à Ken Chen, Professeur à l'université de Paris 13 et à Omar Abou Khaled, Professeur à l'école d'ingénieurs et d'architectes de Fribourg pour avoir accepté la lourde tâche d'être rapporteurs de cette thèse.

Je remercie Dominique Gaiti, Professeur à l'Université de Technologie de Troyes, Pascal Urien, Professeur à l'ENST et Mikael Salaun, Chercheur à France Telecom R&D, pour avoir accepté d'être membres du jury de cette thèse.

Sincères remerciements au personnel de l'ESIB (Ecole Supérieure d'Ingénieur à Beyrouth) pour leurs encouragements; je tiens à nommer le Doyen Professeur Wajdi Najem et l'ex-Doyen Professeur Maroun Asmar.

Pour conclure, je garde une place toute particulière pour toute ma famille. Je les remercie pour leur soutien de tous les instants.

Mes derniers remerciements vont à mon épouse Rima, la moitié qui fait chaque jour de moi un tout. Comment pourrais-je suffisamment la remercier pour son amour sans limites, pour sa grande confiance, pour son constant encouragement, pour ses idées remarquables, pour ses précieux conseils, pour ses remarques pertinentes, pour ses différentes lectures avisées et commentées de mes rapports et publications, pour sa complète et consciencieuse immersion dans mes travaux lors de la validation exhaustive de mes prototypes.
Enfin je tiens à dire combien son soutien quotidien a été important tout au long de ces quelques années, je lui dois beaucoup.

RÉSUMÉ

Les services de gestion des réseaux d'informations deviennent de plus en plus difficiles à implémenter du fait de l'émergence de nouvelles technologies des réseaux, d'applications de plus en plus distribuées sur des systèmes hétérogènes. Les réseaux à base de politiques (PBN) définissent un paradigme des plus prometteurs pour la gestion et le contrôle des ressources réseaux. COPS et son extension COPS-PR tendent à être les protocoles de facto pour l'utilisation de ce type d'applications. La question se pose de savoir comment déployer ce type de services pour permettre les échanges entre les différentes entités protocolaires du réseau (PEP et PDP). C'est à ce niveau que nous proposons de faire intervenir la notion de réseaux actifs, vu que ce paradigme permet d'intégrer du code exécutable aux paquets transférés et/ou nœuds en vue de distribuer et de configurer dynamiquement les services du réseau. D'autre part, une grande effervescence existe actuellement autour du déploiement des Web services sur Internet à l'aide du protocole SOAP. Toujours dans le cadre de l'Internet mais selon une perspective différente, la représentation sémantique des données, et grâce à la définition d'ontologies, permet aux logiciels d'interpréter intelligemment les données qu'elles gèrent et de ne plus jouer le rôle de simple équipement de stockage passif.

La synergie entre les 4 paradigmes: PBN, Réseaux actifs, Web services, et représentation sémantique des données, présente une solution intégrée et portable des plus intéressantes pour la représentation des nœuds de l'architecture active, la conception, l'implémentation et le déploiement de services, et plus spécifiquement un service de gestion dynamique contrôlable et intelligent, dont les données (politiques et règles) sont représentés par une même ontologie.

Le travail effectué dans le cadre de cette thèse s'est basé sur cette synergie et s'est concrétisé par l'adoption d'une architecture à trois plans dans laquelle on trouve:

- Au niveau le plus bas, un plan de déploiement dynamique de services qui est assuré par une plateforme que nous avons développée pour cette fin : ASWA (Architecture à base de Services Web Actifs) qui permet la conception et le déploiement de services réseaux à base de composantes s'appuyant sur des Web Services.
- Au niveau suivant, un plan de services qui supporte dans notre cas les deux services suivants : Un pare-feu distribué pour le filtrage ainsi que le protocole COPS-PR pour la gestion des réseaux. D'autres services actifs peuvent être de même implémentés et intégrés à ASWA.
- Et finalement, au niveau le plus haut, un plan de signalisation dont le rôle est de décrire des méta-données sémantiques en vue de contrôler intelligemment les données du plan de service. Dans notre cas, les données à contrôler ne sont que les politiques à instaurer sur les nœuds et c'est leur conformité aux règles de politiques définies au niveau du plan de signalisation qui doit être vérifiée. Pour cela nous avons d'une part structuré la SPPI en utilisant une ontologie OWL en vue d'utiliser ses définitions sémantiques dans la construction d'un modèle de la base de données de politiques: la PIB. Ce modèle a été validé par la définition de deux PIBs : une PIB DiffServ et une PIB de filtrage. D'autre part, nous avons utilisé le langage SWRL (une extension de OWL) pour définir les règles de politiques. Cette solution marque une avancée très importante au niveau de l'homogénéité de l'infrastructure, permettant ainsi la suppression de la traduction nécessaire lors de l'étape de vérification entre un langage traditionnel de spécification de règles et les langages de représentation de politiques.

Mots clés: Réseaux Actifs, Web Services, déploiement dynamique, routage, sécurité, pare feu, PBN, COPS, PIB, Ontology, OWL, SWRL.

ABSTRACT

This thesis presents a novel active architecture for building and deploying network services: ASWA, Web Services based Active network Architecture. At the architectural level, ASWA defines an active node whose functionalities are divided into the Node Operating System, the Execution Environment, and the Active Applications. At the implementation level, ASWA is a Web Services based platform where new components can be added and deployed, in order to dynamically modify network nodes behavior. Applications can be developed with any language and communicate across heterogeneous environments, and across Internet and Intranet structures. At the deployment level ASWA uses an active node approach, and offers a controlled deployment mode. In terms of security, Authentication of deployed code and protection of the nodes is achieved by the use of HTTPS and the header extensions of the SOAP envelope.

At another level, the task of managing information technology resources becomes increasingly complex as managers must take heterogeneous systems, different networking technologies, and distributed applications into consideration. Policy-based networking (PBN) has emerged as a promising paradigm for configuration management and service provisioning. The Common Open Policy Service (COPS) and its extension for policy provisioning (COPS-PR) are currently being developed as the protocols to implement PBN.

This thesis proposes a new Active Policy-based Network Management architecture which defines, deploys and integrates a policy-based management protocol, in conformity to the COPS protocol, into the active ASWA framework. The objective of this integration is mainly to take advantage of:

- the active network facilities in general, in order to propose a flexible and adaptable management solution to services and network monitoring and control.
- the active web services framework that ASWA offers, in order to propose an interoperable and transparent management solution.

This thesis also proposes a new approach for the definition of network policy management information, in order to enhance its semantic expressiveness and facilitate its maintenance, by using OWL the Web Ontology Language and its extension SWRL.

The combination of the 4 paradigms: PBN, Active Networks, Web services and Semantic representation of information proves to be a promising integrated and portable solution for representing active nodes, conceiving and deploying active services, and more specifically intelligent management services.

Keywords:

Active networks, Web Services, Dynamic deployment, Routing, Security, Policy-based Management, COPS, PIB, Ontology, OWL, SWRL.

TABLE DES MATIERES

CHAPITRE 1 INTRODUCTION 1

1.1. MOTIVATIONS ET ENJEUX 1
1.2. CONTRIBUTION 3
1.3. ORGANISATION DE LA THESE 4

CHAPITRE 2 ETAT DE L'ART 6

2.1. EMERGENCE DES RESEAUX ACTIFS 6
2.2. DIFFERENTES APPROCHES DES RESEAUX ACTIFS 7
2.3. MODELE DE REFERENCE DES RESEAUX ACTIFS 8
2.4. PANORAMA DE QUELQUES PLATEFORMES DE RESEAUX ACTIFS 9
2.4.1. Capsules: ANTS and BEES 9
2.4.2. Extensions Actives: Switchware 10
2.4.3. Réseaux Actifs Virtuels: Netscript 11
2.4.4. Services actifs : AS1 (Active Service Version 1) 12
2.5. RESEAUX ACTIFS A BASE DE MIDDLWARE 13
2.5.1. FAIN 13
2.5.2. COMAN 15
2.5.3. HABA 16
2.6. RESEAUX ACTIFS ET GESTION DES RESEAUX 17
2.6.1. Architecture des réseaux à base des politiques 17
2.6.2. Motivation pour utiliser les Réseaux Actifs dans la Gestion des Réseaux 18
2.6.3. Smart Packets de BBN 19
2.6.4. L'approche FAIN pour la gestion de réseau 21
2.6.5. Gestion active à base des politiques (A-PBM) des réseaux multi-domaines 22
2.7. UTILISATION DES ONTOLOGIES DANS LA GESTION DES RESEAUX 24
2.7.1. Web sémantique et Ontologie 24
2.7.2. Ontologie pour les spécifications de l'information de la gestion 24
2.7.3. Gestion des politiques à base de DAML dans l'environnement KAoS 26
2.7.4. Intégration sémantique des modèles d'information de gestion 28
2.7.5. Application du Langage Web d'Ontologie aux définitions des informations de la gestion 30
2.8. CONCLUSION DU CHAPITRE 32

CHAPITRE 3 PLATEFORME DE RESEAU ACTIF A BASE DE WEB SERVICES 35

3.1. INTRODUCTION 35
3.2. POSITIONEMENT DES PLATEFORMES ACTIFS 37
3.3. APPORT DES WEB SERVICES POUR LES RESEAUX ACTIFS 37
3.4. ASWA : LES CONCEPTS 38
3.5. ARCHITECTURE DE ASWA 40
3.5.1. Format du paquet dans ASWA 40
3.5.2. Nœud d'administration 41
3.5.3. Nœud de contrôle 42
3.5.4. Nœud Actif 43
3.5.5. Fonctionnement du service de routage dynamique d'un nœud actif 43
3.5.5.1. Entête de routage du paquet actif 43
3.5.5.2. Modes de routage 44
3.6. LA SECURITE DANS ASWA 46
3.6.1. Sécurité d'un réseau actif 46

3.6.2. Sécurité des Web Services *47*
3.6.3. Solution de sécurité dans ASWA *47*
3.7. INSTALLATION ANTICIPE DU CODE DANS ASWA 49
3.8. UNE APPLICATION ACTIVE: UN PARE-FEU DISTRIBUE 50
3.8.1. Définition d'un pare-feu distribué *50*
3.8.2. Conception d'un pare-feu distribué actif *51*
3.8.3. « XML SCHEMA » pour la validation des Listes de Contrôle d'Accès *52*
3.8.4. Mise en œuvre du pare-feu actif sous ASWA *54*
3.9. CONCLUSION DU CHAPITRE 56

CHAPITRE 4 IMPLEMENTATION DE ASWA 57

4.1. INTRODUCTION 57
4.2. NODEOS 58
4.3. ENVIRONNEMENT D'EXECUTION D'UN NŒUD ACTIF 59
4.3.1. Le gestionnaire de requête ASWAHandler *61*
4.3.2. Le module d'interception de paquets et le service de routage : ASWA Transport *65*
4.3.3. Implémentation des modes de routage *67*
4.3.3.1. Le mode intégré 67
4.3.3.2. Le mode progressif 67
4.3.3.3. Le mode séquentiel 68
4.4. GESTION GRAPHIQUE D'UN NŒUD ACTIF 69
4.5. NŒUD DE CONTROLE 72
4.6. NŒUD D'ADMINISTRATION 73
4.7. SECURITE DANS ASWA 73
4.7.1. Implémentation de la sécurité dans ASWA *73*
4.7.2. Haute disponibilité des services actifs *75*
4.8. IMPLEMENTATION DU PARE-FEU DISTRIBUE SOUS ASWA 76
4.8.1.1. La classe d'analyse de paquets : FirewallInputFilter 77
4.8.1.2. La classe de construction du paquet 77
4.8.1.3. Exemple d'exécution 78
4.8.2. Administration du pare-feu *78*
4.9. CONCLUSION DU CHAPITRE 80

CHAPITRE 5 APPROCHE ACTIVE ET SEMANTIQUE POUR LA GESTION DES RESEAUX PAR POLITIQUE 81

5.1. INTRODUCTION 81
5.2. MODELE CONCEPTUEL 82
5.3. IMPLEMENTATION DE COPS DANS ASWA 83
5.3.1. L'implémentation des Web services PEP et PDP *83*
5.3.1.1. Entête du message COPS 83
5.3.1.2. Entête des objets COPS 84
5.3.1.3. Les objets COPS 84
5.3.2. Implémentation de la communication entre PEP et PDP *85*
5.3.2.1. L'établissement du contact 85
5.3.2.2. Traitement au niveau du PEP et du PDP 85
5.3.3. Exemple d'un dialogue COPS *86*
5.3.4. Exemple de messages SOAP échangés *87*
5.4. REPRÉSENTATION SÉMANTIQUE DES INFORMATIONS DE GESTION 89
5.4.1. Définition sémantique de la SPPI *89*
5.4.2. Définition sémantique de la PIB *92*
5.4.2.1. Définition d'un modèle générique pour la PIB 92
5.4.2.2. Exemple 1 : La PIB DiffServ 93
5.4.2.3. Exemple 2: La PIB de Filtrage 95

5.4.3. Ajout d'un Plugin à Protégé : Pib Editor *97*
5.5. APPLICATION DE TEST : GESTION PAR POLITIQUE D'UN PARE-FEU 98
5.6. REPRESENTATION SEMANTIQUE DES REGLES DE POLITIQUE 101
5.6.1. Introduction *101*
5.6.2. Représentation des règles de politique en SWRL *101*
5.7. CONCLUSION DU CHAPITRE 103

CHAPITRE 6 CONCLUSION ET PERSPECTIVES 105

ANNEXE A GESTION DES RESEAUX PAR POLITIQUE 108

A.1. INTRODUCTION 108
A.2. L'ARCHITECTURE POLICY-BASED NETWORKING (PBN) 108
A.3. COPS (COMMON OPEN POLICY SERVICE) 110
A.3.1. Généralités *110*
A.3.2. Messages COPS *111*
A.3.3. Exemple d'échange protocolaire *113*
A.4. COPS-PR 114
A.4.1. Objets et messages COS-PR *115*
A.4.2. La PIB dans COPS-PR *116*
A.4.3. La SPPI (Structure of Policy Provisioning Information) *117*
A.4.4. Meta Politique du PIB *118*
A.5. LES REGLES DE POLITIQUE 118

ANNEXE B WEB SEMANTIQUE 121

B.1. LE WEB COMME OUTIL DE PARTAGE D'INFORMATIONS 121
B.2. FORMALISMES ET EVOLUTIONS DES LANGAGES WEB 121
B.2.1. Le langage HTML *122*
B.2.2. Le langage XML *122*
B.2.3. Le Web sémantique : Partage des connaissances sur le Web *123*
B.2.4. Les Ontologies *124*
B.2.5. RDF *125*
B.2.6. RDFS *125*
B.2.7. OWL (Ontology Web Langage) *127*
B.2.8. Editeurs d'ontologie *127*
B.2.9. Conclusion *128*
B.3. SWRL (SEMANTIC WEB RULE LANGUAGE) 128

ANNEXE C TECHNOLOGIE .NET 130

C.1. INTRODUCTION 130
C.2. LA PLATEFORME .NET 130
C.3. ASP.NET 131
C.3.1. LE SERVEUR WEB IIS *131*
C.3.2. ASP.NET *131*
C.3.3. Le pipeline HTTP *133*
C.3.4. La notion de handler HTTP *133*
C.4. LES WEB SERVICES 134
C.4.1. Introduction *134*
C.4.2. Les objets répartis *134*
C.4.3. Les intergiciels (Middleware) *135*
C.4.4. Evolution du Web *135*
C.4.5. Le modèle SOA (Service Oriented Architecture) *136*
C.4.6. WSDL (Web Services Description Language) *137*
C.4.7. UDDI (Universal Discovery Description and Integration) *138*
C.4.8. Le protocole SOAP (Simple Object Access Protocol) *139*
C.4.8.1. Généralités 139

C.4.8.2. Avantages 139
C.4.8.3. Description 140
C.5. WSE (WEB SERVICE ENHANCEMENTS) 142
C.5.1. Introduction au Web Service Enhancements *142*
C.5.2. Architecture de WSE *142*
C.5.3. WS-Routing *143*
C.5.4. WS-Security *144*
C.5.5. Signature numérique *145*
C.5.6. Chiffrement *146*
C.5.7. Exemple d'un message chiffré et signé *146*
ANNEXE D EXEMPLES DE REPRESENTATION DES PIBS 149
D.1. LA PIB DIFFSERV 149
D.1.1. La représentation en ASN.1 ..... *149*
D.1.2. Représentation en OWL *150*
D.2. LA PIB DE FILTRAGE 152
D.2.1. Représentation en ASN.1 ..... *152*
D.2.2. Représentation en OWL *154*
REFERENCES 161
PUBLICATIONS 168
GLOSSAIRE 169

LISTE DES FIGURES

Figure 2.1. Modèle des réseaux Programmables 7
Figure 2.2. Les composantes génériques d'un nœud actif 8
Figure 2.3. Architecture d'un nœud actif FAIN 14
Figure 2.4. Architecture de COMAN 15
Figure 2.5. Nœud de Contrôle dans HABA 17
Figure 2.6 - Le paquet Smart Packets 20
Figure 2.7. Architecture de gestion de FAIN 21
Figure 2.8. Gestion des éléments de réseaux actifs à base de politique 22
Figure 2.9 - Architecture de A-PBM inter-domaine 23
Figure 2.10. Correspondance entre les architectures de CIM et de l'Ontologie 25
Figure 2.11. Les trois types de conflit 27
Figure 2.12. L'application d'Ontologie dans la gestion de réseau 28
Figure 2.13 - Le processus de fusionnement et de mappage de l'information de gestion 28
Figure 2.14. Mappage d'Ontologie 29
Figure 3.1 - Pile réseau 36
Figure 3.2 - Architecture en 3 plans proposée par le projet RNRT Amarrage 38
Figure 3.3. Architecture de ASWA 39
Figure 3.4 - Architecture SOA 40
Figure 3.5 - Format d'un paquet 40
Figure 3.6 - Format XML du paquet actif encapsulé dans un message SOAP transporté via HTTP 41
Figure 3.7 - Déploiement inter domaine du code actif 41
Figure 3.8 - Architecture d'un noeud actif de contrôle 42
Figure 3.9 - Architecture d'un noeud actif 43
Figure 3.10 - Format d'un paquet avec entête de routage 43
Figure 3.11 - Format XML d'un message SOAP conforme à la spécification WS-Routing 44
Figure 3.12 - Routage intégré 45
Figure 3.13 - Routage progressif 45
Figure 3.14 - Routage séquentiel 46
Figure 3.15 - Certificat X509 dans un message SOAP 48
Figure 3.16 - Message SOAP chiffré conformément à la spécification WS-Security 48
Figure 3.17 - Routage WSE 49
Figure 3.18 - Page Web du déploiement dynamique d'un service actif et des fichiers de configuration 50
Figure 3.19 - Déploiement anticipé du pare-feu distribué 51
Figure 3.20 - Processus de filtrage 52
Figure 3.21 - "XML SCHEMA" des ACLs 53
Figure 3.22 - Fichier XML servant comme ACL 54
Figure 3.23 - Format d'un paquet du pare-feu actif 55
Figure 3.24 - Paquet ASWA encapsulant une entête spécifique au pare-feu distribué 55
Figure 4.1 - Les couches du NodeOS dans ASWA 59
Figure 4.2 - L'environnement d'exécution de ASWA 60
Figure 4.3 - Architecture de ASWA Handler 62
Figure 4.4 - Gestionnaire du serveur Web IIS 63
Figure 4.5 - Menu "properties" de l'application Web aswa 63
Figure 4.6 - Configuration d'un gestionnaire HTTP 64

Figure 4.7 - Définition d'un nouveau gestionnaire HTTP 64
Figure 4.8 - Interception des paquets et routage 65
Figure 4.9 - Requête ASWA émise par un client et reçue par le module ASWATransport 66
Figure 4.10 Entête de routage dans un nœud intermédiaire en mode intégré 67
Figure 4.11 – Entête de routage en mode progressif à l'entrée du nœud « flute » 68
Figure 4.12 – Entête de routage en mode progressif à la sortie du nœud « flute » 68
Figure 4.13 – Entête de routage en mode séquentiel 69
Figure 4.14 – Interface de configuration du noeud de contrôle 69
Figure 4.15 - Définition d'un noeud de contrôle 70
Figure 4.16 - Générateur de paquets ASWA 70
Figure 4.17 - Liste des services accessibles par le noeud actif 71
Figure 4.18 - Ecran de paramétrage 71
Figure 4.19 - Ecran de surveillance 72
Figure 4.20 – Interface graphique pour la gestion de la table de routage 73
Figure 4.21 - Filtres de sécurité 74
Figure 4.22 - Exemple de définition des filtres dans le fichier de configuration Web.config 74
Figure 4.23 - Echange sécurisé des paquets SOAP 74
Figure 4.24 - Paquet chiffré et signé 75
Figure 4.25 - Pare-feu ASWA 77
Figure 4.26 - Trace d'un paquet rejeté par le pare-feu 78
Figure 4.27 - Consultation de l'état des paquets reçus 79
Figure 4.28 - Consultation et modification des règles de filtrage 79
Figure 5.1 – Modèle conceptuel de l'approche active et sémantique pour la gestion par politique 82
Figure 5.2 - Classe définissant l'entête du message COPS 83
Figure 5.3 - Classe définissant l'entête d'un objet COPS 84
Figure 5.4 - Classe définissant l'objet COPS "error" 84
Figure 5.5 - Classe définissant l'objet COPS "KeepAlive" 84
Figure 5.6 - Instanciation d'un objet de type KeepAlive 85
Figure 5.7 - Envoi d'un message au PDP 85
Figure 5.8 - Envoi d'un message au PEP 85
Figure 5.9 - Traitement des messages au niveau du PEP 86
Figure 5.10 - Traitement des messages au niveau du PDP 86
Figure 5.11 - Exemple d'un dialogue 87
Figure 5.12 - Message COPS de type "Request" (REQ), encapsulé dans un message SOAP 88
Figure 5.13 - Méta-Classes représentant les macros 90
Figure 5.14 - Méta-classes en protégé 90
Figure 5.15 - Restrictions dans OWL 91
Figure 5.16 - Attributs de SPPI 91
Figure 5.17 - Définition ASN.1 de OT_Entry_PibIndex 92
Figure 5.18 - Représentation OWL de la méta-classe OT_Entry_PibIndex 92
Figure 5.19 - Eléments de la PIB 93
Figure 5.20 - Exemple de la PIB DiffServ 93
Figure 5.21 - Taxonomie de la PIB QdS de DiffServ 94
Figure 5.22- Convention adoptée par la Figure 5.21 94
Figure 5.23 - Représentation en ASN.1 d'une partie de la PIB DiffServ 95
Figure 5.24 - Représentation en OWL d'une partie de la PIB DiffServ 95
Figure 5.25 - PIB de Filtrage en ASN.1 95
Figure 5.26 - PIB de filtrage en protégé 96

Figure 5.27 - Format OWL de la PIB de filtrage 96
Figure 5.28 – Exemple d'une règle de filtrage en OWL 97
Figure 5.29- Widget PibEditor intégrée à protégé 98
Figure 5.30 - Architecture de l'application de test 99
Figure 5.31 - Message COPS encapsulé dans SOAP pour autoriser tout le trafic Internet ... 101
Figure 5.32 -Instant T1: Installation de la politique P1 102
Figure 5.33 - instant T2: Retrait de la politique P1 102
Figure 5.34 - Exemple de règles de politique en SWRL 103
Figure A.1 – Entités fonctionnelles d'une architecture PBN 109
Figure A.2 - Encapsulation protocolaire 111
Figure A.3 – Format d'un message COPS 111
Figure A.4 - Entête d'un objet COPS 112
Figure A.5 - Format détaillé d'un message COPS 113
Figure A.6 - Echange de messages COPS 113
Figure A.7 - Format d'un objet COPS-PR 115
Figure A.8 - Représentation logique de la PIB 116
Figure A.9 - Représentation physique de la PIB 117
Figure A.10 - Forme d'une règle de politique 119
Figure A.11 - Différents niveaux de langages pour la gestion par politique 120
Figure B.1 - Exemple d'un graphe de ressource RDF 125
Figure B.2 - Interface graphique de l'éditeur d'ontologies "protégé" 128
Figure C.1 - Architecture de base de l'infrastructure .NET 130
Figure C.2 - Architecture du serveur Web IIS 131
Figure C.3 - Architecture de ASP.NET 132
Figure C.4 - Processus de compilation à la volée des pages ASP.NET 132
Figure C.5 - Le pipeline HTTP de ASP.NET 133
Figure C.6 - Evolution du Web 136
Figure C.7 - Cycle de vie d'utilisation d'un service Web 137
Figure C.8 - WSDL et Proxy Web 138
Figure C.9 - Annuaire UDDI 138
Figure C.10 - Application dans un contexte multi site 139
Figure C.11 - Message SOAP 140
Figure C.12 - Pipeline WSE 142
Figure C.13 - Traitement d'un message SOAP 143
Figure C.14 - Signature 145
Figure C.15 - Chiffrement 145
Figure C.16 - Message chiffré 145
Figure D.1 - Représentation en ASN.1 d'une partie de la PIB DiffServ 150
Figure D.2 - Représentation en OWL d'une partie de la PIB DiffServ 152
Figure D.3 - PIB de Filtrage en ASN.1 154
Figure D.4 - Format OWL de la PIB de filtrage 160

Chapitre 1

INTRODUCTION

1.1. MOTIVATIONS ET ENJEUX

Les opérateurs de télécommunications et les gestionnaires de réseau souhaitent automatiser le processus de configuration des nœuds et des équipements réseau. Cette automatisation a pour but à la fois de contrôler les flux d'information qui transitent dans ces nœuds et de gérer plus facilement les équipements réseau. De là est née la gestion par politique ou PBM (Policy-Based Management), à laquelle on peut ajouter le contrôle, qui en fait partie de façon intrinsèque.

La gestion par politique implique plusieurs composants. Les nœuds du réseau prennent le nom de PEP (Policy Enforcement Point). Les politiques y sont appliquées pour gérer le flux des utilisateurs. Le PDP (Policy Decision Point) est le point qui prend les décisions et choisit les politiques à appliquer aux PEP. La communication entre le PEP et le PDP s'effectue par le protocole COPS (Common Open Policy Server). Le système comporte également une console utilisateur, qui contient des outils de gestion des politiques. Ces derniers permettent notamment d'entrer les politiques dans une base de données, nommée policy repository, qui entrepose les règles de politique que le PDP vient rechercher pour les appliquer aux nœuds du réseau. Ces règles seront alors traduites automatiquement par le PEP en commandes de bas niveau propres à un constructeur et/ou à un équipement, avant d'être exécutées. Cela permet à l'administrateur de se concentrer uniquement sur la logique des politiques.

Bien sûr, il n'est pas possible de spécifier ces règles avec le langage humain trop ambigu. En fait, il existe des langages spécialisés: Les langages de spécification.

Un langage de spécification est un modèle de description de politiques indépendant du domaine d'application permettant d'atteindre cet objectif. Les langages de spécification les plus utilisés de nos jours sont Ponder, PCIM et PFDL.

En comparaison avec les approches traditionnelles précédentes de gestion de réseau, l'approche basée sur la politique offre une solution pour une gestion plus flexible et adaptative, permettant ainsi aux éléments du réseau d'être configurés instantanément pour chaque application et pour chaque consommateur. Développée à l'origine pour un environnement de réseaux LAN, cette flexibilité est coûteuse car cette approche nécessite d'adresser les problèmes de sécurité, de dimensionnement et des interactions complexes entre les composants appropriés de la gestion de réseau. D'ailleurs, selon le cadre de politique, des politiques sont définies ou modifiés par un outil administratif et l'intervention de l'administrateur est toujours exigée. Les réseaux actifs peuvent résoudre plusieurs des problèmes inhérents dans la technologie actuelle du PBNM. Ils permettent une extensibilité dynamique de l'architecture de la gestion, l'introduction de nouvelles applications ainsi que l'automatisation des tâches de gestion de réseau.

Mais, comme la plupart des architectures de réseaux actifs sont des systèmes fermés, nous

proposons une nouvelle architecture à base de Web services sécurisée et qui permet l'interopérabilité avec d'autres architectures de réseaux actifs.

La sémantique des politiques présente un autre défit dans les efforts de politique actuel. Les politiques qui peuvent être définies sont limitées par des modèles d'informations actuels. Actuellement, ceci comporte uniquement des classes pour la représentation des conditions de politique uniquement basées sur le temps et sur les entêtes de paquets. D'autre part, le eXtensible Markup Language (XML) a émergé dans le monde de l'Internet comme un format de représentation standard, permettant la définition, la transmission, la validation et l'interprétation de l'information. Ensuite plusieurs approches sont apparues, utilisant ce format pour décrire l'information de gestion, y compris ceux proposés par la DMTF et le groupe de travail de configuration du réseau de l'IETF.

Utiliser XML dans la gestion des réseaux a certains avantages. Plusieurs outils globaux et librairies qui traitent ce format peuvent faciliter le développement des applications de gestions. Pourtant, la Définition du Type de Document (DTD) ou les *schémas* de l'XML définissent uniquement le format de l'information, qui ne lui donne pas la sémantique. Un logiciel qui valide la syntaxe d'une série de balises ne peut pas déduire leur signification pour déduire l'état du réseau.

Le besoin d'un modèle conceptuel devient nécessaire pour la description de chaque ressource, de nouveaux langages évolués appelés ontologies voient alors le jour. Ils permettent la description abstraite d'un domaine quelconque d'informations mais en définissant un ensemble de concepts, une hiérarchie, des relations entre les données et surtout des règles qui gouvernent ces concepts. Ainsi on permet aux logiciels d'interpréter intelligemment les données qu'elles gèrent et de ne plus jouer le rôle de simple équipement de stockage passif.

Les langages de définition d'ontologie sont basés sur RDF (Ressource Description Framework) et son Schéma (RDF-S) qui sont des langages basés sur XML. RDF est un modèle conceptuel, abstrait et formel qui permet de décrire tout élément simplement et sans ambiguïté selon un mécanisme basé sur des déclarations RDF. RDF Schema permet de créer une hiérarchie de classes et de propriétés. RDF Schema permet surtout de définir un domaine et une portée d'une propriété afin de restreindre son champ d'utilisation.

Les langages vus précédemment pour le développement d'outils et d'ontologies ne sont pas compatibles avec le Web actuel et de manière plus flagrante avec le Web sémantique. D'où la création d'un langage de définition d'ontologies orientées Web, appelé OWL, pour Ontology Web Language, qui est une extension de RDFS et qui est proposé par le consortium du réseau mondial Web (World Wide Web). OWL étend RDFS en utilisant les notions qui y sont définies. Il permet en plus la création de relations complexes entre différentes classes et la définition de contraintes sur les classes et les propriétés, plus précises que dans RDFS

Ces langages d'ontologie peuvent être utilisés pour améliorer l'expressivité de la sémantique des spécifications de l'information de la gestion. Avec ceux-ci il est également possible de raisonner avec ces informations dans les tâches de la gestion.

Finalement, il faut noter que les politiques ont été proposées dans le but de cacher la complexité de transformation des objectifs à atteindre par l'utilisateur en une configuration au niveau du réseau. Aussi pour établir une passerelle entre ces deux niveaux, une hiérarchie de politiques, comprenant une succession de niveaux d'abstraction, a été définie. Le nombre de niveaux d'abstraction dans une hiérarchie de politique varie. On peut avoir une hiérarchie à trois niveaux, à quatre niveaux ou à six niveaux. Le niveau d'abstraction représente en général, le niveau de compréhension de la politique par les entités qui veulent l'interpréter. Par exemple, une politique de bas niveau sera spécifiée par des paramètres de type adresse réseau alors qu'une politique de haut niveau utilisera les noms des utilisateurs. Par ailleurs, plus le niveau d'abstraction est bas plus la politique est précise et concrète. Un processus d'affinage des politiques intervient pour

faire le lien entre ces différents niveaux d'abstraction. La question se pose de savoir si on peut trouver un modèle avec lequel on peut réduire le nombre de niveaux d'abstraction dans une hiérarchie de politique. C'est à ce niveau que nous proposons l'utilisation du langage SWRL qui est une extension de OWL et qui préserve sa sémantique intéressante. La syntaxe SWRL définit une règle qui est une relation d'implication entre un antécédent (*body*) et un conséquent (*head*). Si les conditions spécifiées dans l'antécédent sont vérifiées alors les conditions spécifiées dans le conséquent le sont aussi.

1.2. CONTRIBUTION

Le travail effectué dans le cadre de cette thèse a abouti à la conception et l'implémentation d'un service de gestion dynamique, contrôlable, intelligent et portable, dont les données (politiques et règles) sont définies au même niveau d'abstraction:

- Le protocole de gestion des réseaux implémenté est conforme au mode de provision (COPS-PR) du protocole COPS. En effet, les réseaux à base de politiques (PBN pour Policy Based Networking) préconisent dans leurs standards l'utilisation du protocole COPS (Common Open Policy Service) pour l'échange des messages entre les entités de l'architecture (PEP et PDP).
- La dynamicité de cette gestion est assurée par un service de déploiement actif.
- L'intelligence de ce service et la réduction du nombre de niveaux d'abstraction, sont assurées par une couche sémantique grâce à l'utilisation d'une ontologie OWL.
- La portabilité de ce service est garantie par l'utilisation de la technologie des Web services dans l'implémentation de l'infrastructure de déploiement.

On peut subdiviser les principales contributions de cette thèse selon les trois perspectives suivantes:

- **Proposition et implémentation d'une Architecture de communication orientée services** : Bien que les architectures de réseaux actifs fournissent un degré de flexibilité significatif au niveau application, la plupart sont des systèmes fermés et ne permettent pas l'utilisation des services existants basés sur le modèle client/serveur ou RPC. Elles ne permettent pas une interopérabilité avec d'autres architectures de réseaux actifs. Le protocole SOAP permet aux applications et aux architectures à base de Web services développées avec un langage de programmation et supporté par sur un système d'exploitation quelconque, de communiquer sans aucune modification. Pour assurer l'interopérabilité entre les architectures de réseaux actifs, il suffirait alors d'encapsuler les paquets de ces derniers dans des messages SOAP qui seront transportés par le protocole HTTP. C'est effectivement ce que nous proposons dans cette thèse, et c'est l'idée clef sur laquelle nous nous sommes basés dans la proposition d'une nouvelle architecture active pour la conception et le déploiement de services réseaux : ASWA, *Architecture à base de Services Web Actifs*. En effet, c'est une architecture à base de Web services qui s'appuie sur l'infrastructure de communication distribuée offerte par le web, pour un déploiement dynamique et contrôlé du code mobile.
  Cette contribution a été faite dans le cadre du projet Amarillo [167] et pourrait s'identifier principalement sur quatre plans :
  - Sur le plan architectural, ASWA définit un nœud actif en conformité avec la spécification DARPA dont les fonctionnalités se répartissent suivant un système d'exploitation (NodeOS), un environnement d'exécution (EE) et les Applications Actives (AA).
  - Sur le plan de l'implémentation, ASWA est une plateforme à base de composantes s'appuyant sur les Web Services, ce qui assure la portabilité au niveau des systèmes d'exploitation ainsi qu'au niveau des langages de programmation.
  - En ce qui concerne le déploiement, ASWA utilise l'approche nœud actif avec la possibilité de contrôler le chargement de code.
  - En terme de sécurité, ASWA permet de sécuriser le déploiement des services actifs en utilisant d'une part le protocole TLS (Transport Layer Secure) et d'autre part l'entête sécu-

risée des enveloppes SOAP des Web services en conformité avec la spécification WS-Security.

Afin de valider ces résultats qui caractérisent ASWA nous avons instancié et déployé une application de filtrage (pare feu) active et configurable dynamiquement

– **Proposition d'une Approche active pour la gestion des réseaux par politiques et implémentation du protocole COPS-PR**: COPS est l'un des protocoles standardisé et souvent utilisé dans la mise en œuvre de la gestion des réseaux par politique. Il permet aux routeurs et aux différents équipements du réseau d'échanger des informations avec un serveur centralisé qui stocke l'ensemble des politiques. Mais l'échange de messages COPS n'est pas en mesure de franchir les barrières d'Internet. Pour contourner ce problème de mobilité, nous avons encapsulé ces messages COPS dans des messages SOAP, tout en conservant l'interopérabilité avec les réseaux COPS existants standard. Pour cela, nous avons implémenté l'échange et le format des messages conformément à COPS. L'automatisation des tâches de gestion de réseau et le déploiement dynamique des différentes entités du protocole (PEP et PDP) ainsi que les politiques échangées sont réalisés grâce à l'utilisation de la plateforme ASWA.

– **Proposition et implantation d'un Modèle à base d'ontologies pour la spécification et la définition des politiques** : Nous proposons également dans cette thèse une nouvelle approche pour la définition des informations de gestion des réseaux à base de politiques afin de rendre celles-ci plus expressives dans leurs sémantiques (valeur ajoutée par rapport aux langages traditionnels). Pour cela, nous avons fait le choix de OWL, le langage d'ontologies conçu par le Consortium WWW et permettant la définition d'ontologies basées Web pour la représentation des politiques, ainsi que le langage SWRL (extension de OWL) pour la définition des règles de politiques.

Les principaux avantages de notre proposition se résument selon les points suivants :

- Mis à part les avantages de la syntaxe de type XML, les langages d'ontologies sont formels et leurs sémantiques est donc complète, d'autant plus qu'il existe des moyens permettant de les tester et les valider.
- L'absence d'une description explicite et partagée de ressources dans les langages Web traditionnels est une limitation pour l'exploitation automatique des informations. Les axiomes et contraintes introduites au niveau des ontologies peuvent être aisément interprétés par le système de gestion par politiques.
- Comparé aux modèles PFDL et PCIM, SWRL a l'avantage de rester dans la même vision que celle utilisée pour la représentation de la PIB à l'aide de OWL. En effet, SWRL étant une extension de OWL, permet d'utiliser la même ontologie pour la représentation de la PIB et la spécification des règles qu'elle contient (l'interpréteur de langage est donc unique). L'utilisation de SWRL marque un avantage des plus importants de notre solution puisqu'il permettra d'éviter une étape de traduction entre les règles de politiques à installer et la base de données de ces politiques.
- La PIB est traditionnellement décrite en ASN.1 de manière abstraite et chaque implémentation de système de gestion peut utiliser ses propres méthodes pour la définition et l'exploitation des politiques dans la PIB.
  Définir des instances (Individuals) des classes dans une ontologie permet d'étendre l'arbre de représentation de l'information et de représenter les politiques de cette PIB avec ce même langage.

1.3. ORGANISATION DE LA THESE

Suite à cette introduction, cette thèse commence par exposer l'état de l'art au niveau du chapitre 2. Cet état de l'art se présente sous trois axes différents : les réseaux actifs à base de composantes, les réseaux actifs dans la gestion des réseaux et enfin la définition des ontologies pour la représentation des informations de gestion des réseaux.

La présentation de notre contribution commence avec le chapitre 3 où nous détaillons la proposition et la conception d'une architecture de réseau actif à base de Web services qu'on a nommé ASWA. L'implémentation de cette architecture sur la plateforme .NET de Windows est décrite au niveau du chapitre 4

Dans le chapitre 5, nous proposons une approche active pour la gestion des réseaux par politique ainsi qu'un modèle sémantique pour la représentation des informations et des règles de politique.

Le dernier chapitre conclut le travail et discute les perspectives futures.

Des informations supplémentaires sont fournies au niveau de quatre annexes :

- Nous réservons les Annexes A et B pour l'exposé du contexte dans lequel nous avons effectué notre travail :
  - Au niveau de l'annexe A, nous exposons les détails relatifs à la gestion des réseaux par politique et plus spécifiquement le protocole COPS,
  - tandis qu'au niveau de l'annexe B, nous faisons un rappel sur le Web sémantique et sur les langages d'ontologies.
- L'annexe C donne un aperçu sur la technologie .NET qu'on a adoptée pour le développement des différents modules réalisés dans le cadre de cette thèse.
- L'annexe D décrit la représentation sémantique de deux exemples de PIBs qu'on a implémentés pour valider le modèle que nous avons proposé pour la représentation des informations de gestion.

Chapitre 2

ETAT DE L'ART

2.1. EMERGENCE DES RESEAUX ACTIFS

L'Internet repose sur le protocole d'interconnexion IPv4 [153] développé dans les années 60-70. Même si la version IPv6 [154] est dans un état de normalisation stable et bénéficie déjà d'expériences de déploiement partiel, les fonctionnalités de base n'ont pas évolué de manière importante : à la fois IPv4 et IPv6 permettent de transférer un paquet d'une source à une destination en fonction de son adresse. Il est très difficile d'étendre ces protocoles ou d'ajouter des fonctions spécifiques pour offrir de nouveaux services. Deux exemples typiques sont les mécanismes de qualité de service (QoS) et les services de communication multipoint - même si de tels mécanismes ou protocoles existent, ils ne sont pas déployés à l'échelle de l'Internet. Or, dans beaucoup de situations, des applications communicantes bénéficieraient de certaines fonctionnalités qui, placées dans le réseau, amélioreraient les performances ou rendraient un service utile.

Cette vision du réseau minimaliste et difficilement extensible a mené à l'apparition de réseaux actifs ou programmables. L'idée de ce type de réseaux est d'ouvrir la boîte hermétique des éléments réseau en permettant d'ajouter du code exécutable à la fonction du transfert de paquets. On définit alors une machine d'exécution associée à la communication dans un nœud du réseau. Ainsi, un nœud actif peut permettre une extension de fonctionnalités de protocoles de base pour répondre aux besoins d'applications, par exemple déployer un nouveau service ou personnaliser le comportement du réseau. L'intérêt de l'exécution dynamique du code sur un élément dans le réseau peut être encore amplifié si la machine d'exécution propose l'accès à certaines fonctions internes du nœud, liées soit à la communication (routage, duplication de paquets, mise en mémoire) soit à l'information sur l'état du réseau (niveau de congestion sur un lien, nombre de flots, voisinage connu).

Le code exécutable peut être soit installé sur un nœud par un moyen externe et s'exécuter au passage des paquets, soit venu des paquets qui traversent un routeur, chaque paquet portant une partie de code à exécuter sur le routeur. Même si la terminologie n'est pas clairement déterminée, nous utiliserons le terme *réseaux programmables* pour désigner la première approche et le terme *réseaux actifs* pour la deuxième. Notons que parfois on utilise le terme réseaux actifs dans le sens générique qui englobe les deux approches.

Dans l'approche à base de réseaux programmables, on conserve la structure des unités de protocoles qui traversent le réseau ; le code placé de manière dynamique dans certains éléments permet d'étendre les fonctionnalités du réseau. Le problème important de cette approche est l'interface de contrôle d'un nœud programmable. C'est pour cette raison que le développement des réseaux programmables a été entrepris par la communauté de recherche des réseaux OPENSIG [155] et poursuivis dans le projet P1520 [105] de l'IEEE. Cette voie propose de normaliser des interfaces programmables pour des commutateurs ATM, des routeurs IP et des réseaux de télécommunications sans fil afin de voir et de manipuler des dispositifs physiques du réseau comme des objets distribués avec des interfaces programmables bien définies. Les interfaces ou-

vertes permettent aux fournisseurs de services de manipuler des états du réseau en utilisant des outils intergiciels (middleware) tels que CORBA pour construire et gérer de nouveaux services [1]. D'où le modèle de réseaux programmables (Figure 2.1) qui incorpore d'une part trois plans, représentant les différentes applications et d'une autre part introduise un nouveau modèle d'exécution (ou *computation*) qui fournit un support programmable à travers les plans de transport, de contrôle et de gestion, et qui permet de programmer les différentes couches (application, transport, réseau et liaison) dans ces plans.

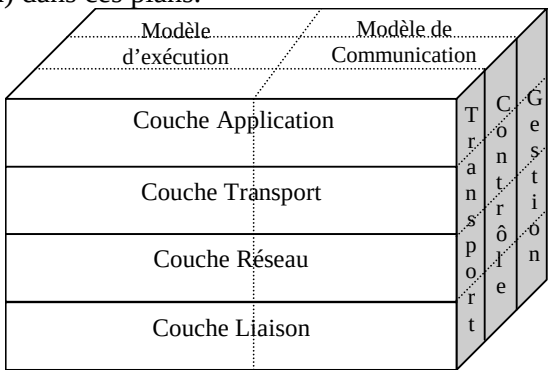

Figure 2.1. Modèle des réseaux Programmables

La deuxième voie a été initiée par les travaux de D. Wetherall et D. Tennehouse au MIT suivis par la communauté de recherche en réseaux, financée de manière importante par plusieurs projets de la DARPA [2]. Les premiers travaux ont proposé de considérer les paquets comme des capsules contenant du code pouvant être exécuté dans des routeurs qui fournissent un environnement d'exécution. Cette approche supprime la différence entre les données et le contrôle - un paquet ne devient que du code qui s'exécute dans le réseau. Ceci nécessite un nouveau format pour des capsules : ANEP (Active Network Encapsulation Protocol) [13] est le format utilisé dans Abone (Active Backbone), le réseau expérimental d'interconnexion de routeurs actifs [158].

Dans la section suivante, nous introduisons l'approche des réseaux actifs, vu que nos travaux se sont effectués dans le cadre de cette approche. La section 3 explique le modèle de référence des réseaux actifs. La section 4 fournit un aperçu sur quelques réseaux actifs. Dans la section 5 nous donnons des exemples sur l'état de l'art des réseaux actifs implémentant les fonctions et les mécanismes de Middleware. La section 6 donne des exemples sur des projets de réseaux actifs pour la gestion des réseaux. La section 7 introduit l'utilisation de l'ontologie dans la définition de l'information de gestion des réseaux et décrit quelques projets relatifs. Enfin, nous concluons dans la section 8.

2.2. DIFFERENTES APPROCHES DES RESEAUX ACTIFS

Un réseau actif est un réseau dans lequel tout ou partie de ses composants sont programmables dynamiquement, dans les différents plans (signalisation, supervision, données), par des entités tierces (opérateurs, fournisseurs de services, applications, usagers).

Le déploiement dynamique des services s'effectue soit dans le même flux que les données (approche intégrée ou par paquets actifs), soit dans un flux séparé (approche discrète).

Dans un réseau actif, les applications ne sont plus exécutées seulement aux extrémités (postes client et serveur) mais elles sont réparties sur le réseau actif. Certaines entités du réseau (routeurs de bordure, par exemple) traversées par les paquets de communication prennent en charge une partie des applications (multimédia, par exemple).

Plusieurs approches [14] ont été proposées pour implémenter une architecture de réseaux actifs. Voici trois approches majeures :

- Approche « ad-hoc » : Il y aurait certains services standard ou modules dans les nœuds qui seront sélectionnés et invoqués à partir d'options (Flags) présentes dans les paquets. Le reste du paquet sera considéré comme donnée, devant être traitée par les routines invoquées. Les

modules actuels de « IP Processing » peuvent être considérés comme un exemple d'une telle approche.

- Approche discrète ou le modèle de « nœud programmable» : Cette approche permet aux codes utilisateurs de s'exécuter sur les différents nœuds. Le code est téléchargé dans les nœuds hors bande, et le flux normal de paquets est traité en tant que données ou entrées pour ce code. Dépendamment de la politique de sécurité, le téléchargement de ces codes peut être limité seulement aux administrateurs.
  L'approche discrète est certainement préférable dans le cas où les programmes sont relativement grands à comparer aux paquets. Elle maintient une modularité entre les données de l'utilisateur et le programme, ce qui peut être très utile pour des tâches de gestion de réseau. Nous citons quelques plateformes qui en sont des exemples : La plateforme *DAN* [5] proposée par *Washington University* et par *ETH Zurich*, l'architecture *ANTS* [6] proposée par *MIT*.
- Approche intégrée ou le modèle « capsule » : Chaque paquet est traité en tant que programme devant être exécuté à chaque nœud intermédiaire. C'est l'approche la plus générale des trois. *Smart packets* [12] proposée par *BBN technologies*, *The active IP option* [109] proposée par MIT et *l'architecture M0* [108] proposée par l'université de Californie en sont des exemples.

2.3. MODELE DE REFERENCE DES RESEAUX ACTIFS

Dans un effort de standardisation, la communauté de recherche de réseaux actifs DARPA a publié en 1999 un modèle de référence sur l'architecture commune des composants et des interfaces d'un nœud actif appelé : le Modèle d'Architecture pour les Réseaux Actifs (Architectural Framework for Active Networks) [3]. Ce modèle décrit comment traiter des paquets et gérer des ressources dans un nœud actif. Dans ce modèle, un nœud de réseau actif est divisé en trois composantes majeures:

- Un Système d'Exploitation du Nœud (Node Operating System ou NodeOS) offrant le support système en matière d'ordonnancement de tâches, de gestions des entrées-sorties et de la mémoire,
- Un ou plusieurs Environnements d'Exécution (Execution Environment, EE). Un EE pouvant être une machine virtuelle Java par exemple.
- Des Applications Actives (AA) dynamiquement chargées et interprétées par les EE afin de mettre en œuvre de nouveaux services.

L'organisation générale de ces composantes est illustrée dans la Figure 2.2.

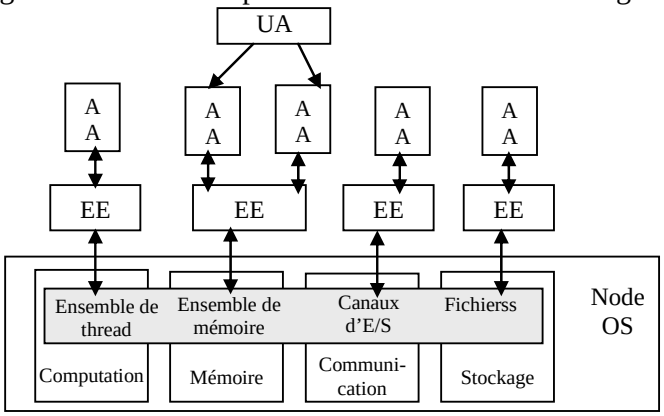

Figure 2.2. Les composantes génériques d'un nœud actif

Par analogie à un système d'exploitation traditionnel, le NodeOS prend en charge la gestion des ressources (Bande passante, mémoire, cycles CPU,…) et fournit un ensemble de capacités de base utilisées par des EEs pour implémenter une machine virtuelle. Un EE offre une interface de programmation ou une machine virtuelle qui peut être programmée par les AAs. Il interprète les paquets actifs entrants qui contiennent le code nécessaire constituant l'application active (AA). Les utilisateurs obtiennent des services du réseau actif via ces AAs. Le NodeOS isole un EE de la gestion des ressources et de l'effet du fonctionnement des autres EEs. De leur côté les

EEs cachent au NodeOS les détails de l'interaction avec les utilisateurs. Quand un EE demande un service ou une ressource au NodeOS, la demande est accompagnée d'une signature grâce à laquelle, le NodeOS peut décider d'allouer le service ou pas.

Chaque NodeOS a un EE particulier appelé l'Environnement d'Exécution de Gestion (MEE) qui assure les tâches suivantes :

- Maintenance de la base de données de la politique de sécurité.
- Chargement de nouveaux EEs, ou mise à jour et configuration des EEs existants.
- Support de l'installation des services de gestion des réseaux depuis un centre de gestion distant.

Il est recommandé que les EEs offrent la possibilité de servir plus d'une AA en même temps. Une première spécification de l'interface d'un NodeOS est décrite dans [156] ainsi qu'une architecture pour la gestion des réseaux actifs dans [157].

2.4. PANORAMA DE QUELQUES PLATEFORMES DE RESEAUX ACTIFS

Nous allons voir dans cette section, à travers divers projets de recherche, les avantages que les réseaux actifs peuvent apporter au réseau Internet. Ils permettent une évolution rapide des services et une meilleure adaptation des fonctionnalités aux besoins des applications. Les réseaux informatiques et ceux de télécommunications convergeraient alors pour former un réseau global unique plus efficace, plus sûr et plus performant. Dans ce panorama, nous n'avons pas l'intention de représenter une revue approfondie de tous les domaines, mais nous essayons plutôt d'identifier les contributions essentielles de ces divers projets en termes d'innovation et d'utilité relativement à un domaine et selon des caractéristiques spécifiques.

2.4.1. Capsules: ANTS and BEES

ANTS [6] a été développé au MIT. C'est une plateforme qui permet le déploiement dynamique d'une multitude de protocoles de communication pour l'offre de plusieurs services dont le transport du code mobile, le chargement du code à la demande et les techniques du cache. Ces services de base permettent aux architectures réseaux d'introduire ou d'étendre les protocoles réseaux existants.

ANTS offre un environnement de réseau programmable pour la définition d'une architecture basée sur les capsules (des paquets qui transportent des données et du code). Les exemples de tels services de réseau programmable incluent des services améliorés du multicast, le routage IP mobile et le filtrage au niveau application. Les capsules représentent des unités atomiques des interfaces de traitement et d'acheminement des informations. Les nœuds actifs traitent les capsules, assurent leur routage et supportent le mécanisme de distribution du code pour le déploiement autonome de nouveaux services.

Un réseau ANTS est constitué de nœuds interconnectés qui supportent une plateforme java et qui communiquent entre eux en s'échangeant des capsules à l'aide du protocole UDP. Chaque capsule transporte une référence sur sa routine de traitement faisant partie d'un protocole défini par l'application qui a injecté la capsule pour la première fois dans le réseau. Ainsi lorsqu'une capsule arrive à un nœud et demande l'exécution d'un protocole qui n'est pas présent, le nœud actif demande à l'émetteur de cette capsule de lui transmettre le code associé. Une fois le code reçu, le traitement de la capsule s'effectue.

Les trois principales contributions de ANTS sont certainement son mécanisme de déploiement dynamique de code, sa gestion des protocoles au sein des nœuds et le support de cache de données.

Par contre, les 2 failles principales de ANTS se situent au niveau de la sécurité et de la performance :

- Au niveau de la sécurité, un code java, produit par n'importe qui, pourrait être exécuté. Ce manque au niveau des services d'authentification et d'autorisation impose un traitement similaire à tous les flux du réseau, et empêche les utilisateurs d'acquérir des ressources addition-

nelles de CPU ou de réseau. En outre, les utilisateurs ne peuvent non plus accéder à des objets simples comme les fichiers ou des classes supplémentaires.

- Au niveau de la performance, ANTS ajoute une couche supplémentaire de routage à n'importe quelle transmission de paquet.

Dans un effort de remédier aux imperfections de ANTS, et vu son succès, l'université de Utah a implémenté en 2002 la plateforme BEES [7] qui n'en est qu'une amélioration, surtout au niveau de la sécurité. Bees est un environnement d'exécution de réseau actif qui télécharge et exécute des protocoles de communication écrits en Java. Le système fournit un mécanisme d'authentification et d'autorisation flexible, un accès aux ressources privilégiées basé sur la capacité, et des services pour découvrir et surveiller les nœuds voisins. Bees peut de même permettre l'isolement, l'arrêt, et le contrôle de ressources sur le code mobile.

2.4.2. Extensions Actives: Switchware

Switchware [8], projet développé à l'université de la Pennsylvanie, tente d'équilibrer la flexibilité d'un réseau programmable contre les besoins de sûreté et de sécurité dans une infrastructure partagée telle que l'Internet. Les travaux menés dans ce projet visent à définir une architecture homogène incluant différents paradigmes tels que le nœud actif, le paquet actif ainsi que la sécurisation d'un réseau actif. Les aspects langage de programmation de réseau actif sont également au cœur des travaux de Switchware.

L'architecture générale d'un nœud Switchware est basée sur 3 couches. La première représente un niveau de système d'exploitation et d'environnement d'exécution de base. La seconde offre une facilité d'enrichissement du système d'exploitation par des extensions actives (fonctions offertes aux paquets actifs sur un nœud). La troisième couche représente la fonction de traitement de paquets actif comportant chacun du code exécuté sur chaque nœud traversé. L'architecture Switchware repose sur 3 principaux composants qui sont : ALIEN, PLAN et SANE.

ALIEN

ALIEN constitue l'un des fondements du projet Switchware. ALIEN explore l'architecture d'un nœud actif comportant à la fois des extensions actives chargées par l'opérateur du réseau et un support pour des paquets actifs comportant chacun le code associé à leur traitement.

Le langage de programmation des extensions ainsi que des paquets actifs est OCAML étendu avec des primitives de manipulation de paquets actifs (accès, envoi, réception). L'architecture d'un nœud repose sur 3 niveaux:

- un système d'exploitation standard (Linux) comportant un interpréteur OCAML;
- une interface de nœud offrant un accès au réseau et au système d'exploitation ainsi qu'un composant de chargement de code permettant de charger des extensions actives;
- des librairies de routines utiles.

L'intérêt d'ALIEN est d'offrir dans un environnement standard, toutes les fonctions de base requises d'un nœud pour les fonctions de niveau supplémentaire.

Langage de programmation de paquets (PLAN)

PLAN (Packet Language for Active Networks) constitue une approche complémentaire d'ALIEN du point de vue langage de programmation de réseau actif et s'appuie sur l'architecture d'ALIEN pour la construction d'un nœud actif. En effet, PLAN n'exploite pas un langage de programmation générique mais définit son propre langage inspiré du langage fonctionnel.

La communication entre nœuds actifs se traduit dans un programme par une évaluation de fonction à distance et donne lieu à la génération et l'envoi de paquets actifs au travers du réseau. L'intérêt principal de disposer d'un langage restreint réside dans les limites que l'on peut placer sur celui-ci afin de maîtriser les actions réalisables par les paquets dans les nœuds. Cela permet notamment de restreindre la sécurité placée sur le code dans les paquets. De plus, cela permet de fournir un ensemble de primitives dédiées à l'évaluation à distance et au traitement des paquets dans le langage.

Environment sécurisé (SANE)

La sécurisation d'un nœud ainsi que des paquets et extensions actives est le résultat des travaux menés dans SANE (Secure Active Network Environment). Ici, un mécanisme en couches est défini. Le niveau le plus bas inclut la couche physique du routeur actif ainsi que la couche de système d'exploitation. Ce niveau a pour objectif de garantir que le système est chargé dans un état correct.

Le mécanisme de bootstrap développé dans SANE s'appelle AEGIS et garantit que le système est correctement lancé en ne permettant le chargement dans la machine physique que du code (en fait l'OS) signé d'un tiers auquel on fait une totale confiance. Ceci permet de charger l'OS ainsi que l'environnement d'exécution de base.

Pour cela, une extension du BIOS d'un PC a été réalisée afin que l'initialisation passe par l'environnement AEGIS. Dans une seconde étape, pour le chargement des extensions actives, cette architecture offre des mécanismes d'authentification de ces extensions. L'interface qu'offrent ces extensions aux paquets actifs étant clairement définie, ceux-ci ne peuvent pas la contourner. Les paquets actifs eux-mêmes ne nécessitent pas, en raison des limitations faites sur les programmes contenus, de sécurisation particulière. Ce niveau est classifié comme niveau public.

Sur ces paquets actifs, PLAN fournit des garanties importantes pour la sécurité notamment celle que tout paquet se termine (pas de boucles infinies) et qu'un paquet ne peut aller fouiner dans des données d'un autre paquet et ce grâce au typage fort. La limite de ressources garantit cette terminaison à l'échelle du réseau. Ces fonctions sont offertes dans ALIEN.

Finalement un mécanisme de contrôle d'accès à des extensions actives privilégiées avec authentification des paquets entrants permet de restreindre le champ d'accès de chaque paquet.

Intérêts et limites

L'approche Switchware est certainement l'approche la plus complète dans le domaine des réseaux actifs aujourd'hui. Elle propose un langage, un format de paquets, une architecture de sécurité et atteint des performances intéressantes (60 Mbps sur un lien de 100Mbps dans une approche de nœud actif). Cependant, certains points de l'approche sont discutables, notamment l'impossibilité pour un paquet actif de déposer des états dans un nœud pour des paquets futurs et l'absence de cache de code nécessitant l'envoi dans tout paquet actif du code de traitement associé et, lors de chaque réception dans un nœud, du rechargement de ce code. Ceci entraîne un coût énorme sur le traitement des paquets.

2.4.3. Réseaux Actifs Virtuels: Netscript

NetScript [10], projet réalisé à l'université de Colombie, propose une architecture de réseau programmable (nœud actif pur). Il fournit une architecture pour programmer des réseaux, une architecture de nœud programmable dynamiquement dans le réseau, un langage appelé NetScript pour développer des logiciels de réseau sur cette architecture programmable ainsi qu'un ensemble de classes Java (toolkit NetScript).

NetScript est développé pour supporter la notion de Réseaux Actifs Virtuels en tant qu'abstraction programmable. Les abstractions de Réseau Actif Virtuel [11] peuvent être composées et gérées de manière systématique. De plus, NetScript automatise la gestion du nœud au travers d'extensions du langage qui génèrent des MIBs. Les tâches principales d'un nœud NetScript sont le traitement des flux de paquets et l'allocation des ressources du nœud pour permettre ces traitements. La programmation des fonctions de traitement (services) est réalisée sous forme de programmes, appelés boîtes, interconnectés par des flux de paquets. Les boîtes forment un système réactif dans lequel les données sont transmises d'une boîte à l'autre. Une boîte peut être décrite en utilisant le langage NetScript, un langage fortement typé qui crée des abstractions permettant de programmer les fonctions des nœuds du réseau. Cette description est ensuite compilée afin de générer le code Java correspondant. Un protocole est donc décrit sous forme de composition de boîtes élémentaires.

La composition de bout en bout, qui utilise le concept de VAN (Virtual Active Network), permet la construction d'éléments actifs.

Un VAN est constitué d'un ensemble de moteurs NetScript (NetEngine) dans un réseau virtuel, permettant le déploiement de code, la gestion de la configuration, la sécurité et la gestion des ressources de bout en bout. Un VAN est créé en reliant des moteurs NetScript au travers de liens virtuels. Un lien virtuel correspond à un ensemble de liens et de nœuds physiques et peut interconnecter n'importe quel nombre de moteurs NetScript pour gérer des liens de multidiffusion. Le VAN est géré en tant qu'entité, ce qui ouvre des possibilités nouvelles pour la gestion de la sécurité à l'intérieur du réseau. NetScript est une approche qui se classe à la fois dans la famille des réseaux programmables et dans celle des réseaux actifs (sous-classe nœud actif).

Avantages et limites

Le principal apport de NetScript est la programmabilité offerte à tous les niveaux des réseaux (Ethernet, IP, UDP, TCP). L'approche apporte une seconde innovation qui est son utilisation en supervision de réseau. En effet, ce type de nœud est parfaitement adapté pour créer des moniteurs spécifiques de type RMON mais bien plus puissants. Cependant, ce n'est pas dans ce but que NetScript a été initialement créé. Du point de vue des composants, la principale lacune est l'absence à ce jour du langage complet pour la spécification des composants ainsi que la partie concernant la génération automatique de MIBs qui fut en partie à l'origine de l'approche.

2.4.4. Services actifs : AS1 (Active Service Version 1)

Pendant que beaucoup de projets se concentrent sur l'architecture et l'implémentation des nœuds actifs ainsi que sur des langages spécifiques pour la composition des services actifs, une nouvelle approche est née en se basant sur l'argument suivant : un ensemble très large d'applications (i.e. service de transcodage ou proxy) n'ont pas vraiment besoin d'une architecture active complexe où beaucoup de problèmes ne sont pas encore totalement résolus. Ces applications peuvent être supportées par une architecture de services programmables construits au dessus des services d'Internet existants si elle permet aux utilisateurs de télécharger et d'exécuter du code (application) à des emplacements stratégiques dans le réseau. Ce code ou application est appelé "service actif ". Bien sûr une telle approche ne permet pas de programmer des niveaux plus bas dans le réseau, ce n'est donc pas la solution pour résoudre des problèmes comme "SYN-flooding" or le routage intelligent etc. Pourtant cette infrastructure de services actifs enrichit le domaine des réseaux actifs et elle résout le problème de compatibilité avec le réseau Internet actuel.

Dans [160], les auteurs ont proposé une infrastructure appelée AS1 (Active Service version 1) comprenant les six composants suivants :

- Service de localisation : c'est l'entité avec laquelle un client doit prendre contact pour initialiser un service actif désiré. Le service actif est appelé "agent de service" ou "servent". Il y a un mécanisme pour qu'un client ait un rendez-vous avec un AS1. On peut soit utiliser la communication multipoints soit utiliser une méthode centralisée grâce à un serveur.
- Service d'environnement qui définit la partie active des services actifs, i.e. le modèle de programmation et les interfaces pour manipuler les ressources disponibles pour des services actifs dans l'AS1. Mais cet environnement ne permet pas aux services actifs de manipuler la table de routage, d'utiliser des fonctions d'envoi des paquets ou de gestion des réseaux.
- Service de gestion qui alloue des ressources aux services actifs pour avoir un équilibre de charge du CPU. Ce service effectue aussi le contrôle d'admission des services actifs ou la création de ceux-ci.
- Service de contrôle : une fois un service actif instancié dans AS1, le client doit pouvoir dynamiquement le reconfigurer et le contrôler via un service de contrôle.
- Service d'attachement qui aide un client n'ayant pas de moyen de contacter directement les AS1s à avoir ce contact grâce à un mécanisme de "tunnel".
- Service de composition qui permet à des clients de contacter plusieurs AS1s et d'interconnecter des services actifs dans ces AS1s.

Le modèle AS1 a été implémenté en se basant sur la plateforme ”MASH” qui est un interpréteur du langage Tcl [161] avec des capacités de multimédia en temps réel et de réseaux. L’interface proposée par AS1 est un ensemble de classes d’objets Tcl qui peuvent être invoqués par des services actifs (des programmes Tcl). Un service de transcodage de vidéo a été ainsi réalisé comme un service actif.

2.5. RESEAUX ACTIFS A BASE DE MIDDLWARE

Bien que les plateformes des réseaux actifs offrent une grande flexibilité au niveau des applications, ils sont parfois des systèmes fermés qui n’offrent pas des services pour les applications client/serveur déjà existantes. En utilisant le support middleware, les applications distribuées déjà existantes peuvent être modifiées aisément pour être conforme aux architectures de réseau actif. La présence d’une interface de middleware bien définie permet le lien entre différentes architectures, formant ainsi un réseau actif global sans le besoin de protocole d’encapsulation intermédiaire.

Les middlewares offrent le support pour l’authentification des utilisateurs et les traitements. En outre, le service d’isolation des procédés des clients existe dans l’architecture des middlewares pour prévenir toute collision éventuelle entre les clients. La protection de la mémoire est assurée par défaut permettant au client d’accéder aux données dans l’espace qui lui est réservé, toute autre tentative d’accès doit être authentifiée.

Plusieurs systèmes existants peuvent être identifiés en tant que réseaux actifs à base de Middleware. Le système le plus évident qui peut être considéré à base de Middleware, est le réseau actif ANTS dû à l'utilisation inhérente du modèle distribué de communication par objet de Java. Cependant, ANTS ne prévoit pas encore la communication de système ouvert et le support de multi fournisseurs.

Dans les sections suivantes nous décrirons les propositions et implémentations suivantes de tels systèmes.

- Chameleon [33] est implémenté sous Linux. Il utilise une solution propriétaire qui propose un langage de description des composants pour pouvoir négocier avec les différents Environnements d'Exécution. Le déploiement des services se fait à partir d'une station d'administration en utilisant les paquets actifs. Il fait partie du projet FAIN [31] qui servira comme une entrée aux activités de standardisation, par exemple, IEEE P1520 ou IETF.
- ComAN [34] permet aux utilisateurs de développer des routines de traitement de paquets personnalisées qui se nomment des «modules routeur». Ces modules sont des composants DCOM [17] de Microsoft, ils sont liés à un environnement de Microsoft Windows et ils s'exécutent dans l'espace d'adressage utilisateur. Plusieurs langages peuvent être utilisés pour développer ces modules routeurs et ils peuvent être activés ou désactivés sur demande, sans interruption de service. Ils sont chargés en mémoire lors de leur première invocation, et doivent être installés manuellement par un administrateur avant leur utilisation. L'authentification de tous les composants du réseau actif est gérée par le système d'exploitation. ComAN n’est pas conforme à la spécification DARPA.
- HABA [35] est une architecture active pour la conception et le déploiement de services réseaux. C’est une architecture multi-tiers à base de composantes logicielles réutilisables, s’appuyant sur la certification du code mobile, et les techniques de cache pour un déploiement dynamique contrôlé, dans un environnement distribué, et utilisant EJB (Entreprise Java Beans) [21].

2.5.1. FAIN

FAIN [31] est un projet de collaboration de recherche et de développement dans le cadre du programme IST partiellement financé par la Commission de l'Union Européenne. C'est une architecture de réseau active à base de Middleware, visant à fournir un système intégré à base de matériel et de logiciel facilement configurable, orienté vers le déploiement de service dans des réseaux actifs hétérogènes. Il propose une architecture générique pour les réseaux actifs permettant

leur intégration avec la technologie des agents mobiles et les objets distribués. Le projet FAIN implémente deux approches :

- L'approche « réseau programmable », qui permet aux applications de contrôler le réseau au moyen d'interfaces normalisées. Ceci étant réalisé dans un premier temps sur les commutateurs ou les routeurs programmables.
- La deuxième approche est basée sur le transfert du code exécutable au moyen de paquets actifs ou capsules.

Une nouvelle architecture 3-tiers de nœud est envisagée (Figure 2.3). Cette architecture représente un modèle générique pour développer les éléments d'un réseau actif qui se composera ainsi d'un ensemble de tels nœuds actifs reliés par une variété de technologies de réseau, par exemple, ATM, Ethernet et UMTS.

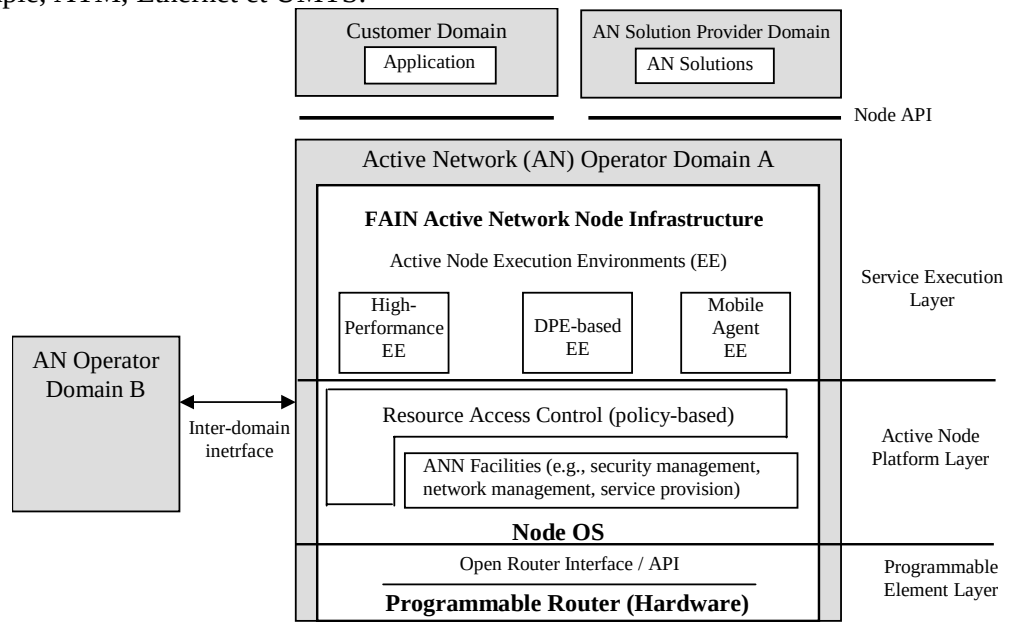

Figure 2.3. Architecture d'un nœud actif FAIN

Pour avoir un contrôle ouvert et sécurisé des ressources du réseau, un nœud actif est construit sur un élément du réseau programmable, par exemple, un router IP ayant une interface ouverte. La plateforme de traitement dans le nœud fournit une couche à travers laquelle les composants téléchargés ou injectés réagissent avec les réseaux et l'un avec l'autre. En général, elle consiste en un système d'exploitation local; un ou plusieurs environnements de traitement distribué, et des fonctionnalités système. Au dessus de cette plateforme, un ensemble d'environnements d'exécution (EEs) sera créé pour accueillir des composants du service. En plus, une interface ouverte de nœud sera définie, et elle représente une abstraction des ressources du routeur s'étendant des ressources informatiques (par exemple, unité centrale de traitement, et mémoire) aux ressources réseaux (par exemple, bande passante et mémoire tampon).

FAIN est conforme à la spécification DARPA. Un nœud actif FAIN est sous le contrôle d'un NodeOS. Le NodeOS sert de médiateur pour la demande des ressources, y compris la transmission, le traitement, et le stockage entre les EEs et isole ces derniers les uns des autres. Les différents EEs peuvent avoir différents niveaux de confiance; et le NodeOS protégera le nœud en imposant des limites de frontière et de ressources sur chacun de ses EEs.

PromethOS [32] est l'implémentation prototype courante de ce NodeOS. Il est basé sur Linux 2.4.x et devrait être capable de s'exécuter pour n'importe quel genre de EE qui est situé dans l'espace utilisateur. PromethOS est basé sur l'architecture de Netfilter [159], il joue le rôle de routeur pour les composants implémentés dans le noyau. Chaque composant sous PromethOS est un module chargeable dans le noyau Linux. PromethOS fournit une architecture pour la gestion de ces modules.

La couche d'exécution de service (Service Execution Layer) de l'architecture du nœud FAIN est conçue pour supporter des environnements d'exécution concurrents. Les EEs cachent aux utilisateurs la plupart des détails de la plateforme et implémente l'API du réseau et exécutent les composants logiciels.

Chameleon [33] est l'implémentation prototype actuelle pour le modèle de service, et pour l'installation et la configuration des composants logiciels dans les réseaux actifs hétérogènes. Le modèle de service est basé sur une solution propriétaire qui propose un langage de description des composants pour pouvoir négocier avec les différents Environnements d'Exécution. Le déploiement des services se fait à partir d'une station d'administration en utilisant les paquets actifs. Il permet également la maintenance et la configuration de PromethOS.

Le déploiement de service au niveau du réseau et la distribution des composants dans des environnements pourrait être implémenté en utilisant le protocole actif d'encapsulation de réseau (ANEP), qui fournit les possibilités pour des utilisateurs d'avoir leurs paquets acheminés à un EE particulier dans un nœud [13]. De toute façon, le programme peut être transporté via l'approche programmable intégrée de commutateur ou l'approche discrète de capsule. Le téléchargement de service est lancé quand un paquet avec une référence à un service inconnu arrive au nœud, une variante également utilisée dans FAIN consiste à utiliser une interface basée sur CORBA pour déclencher l'installation de service.

2.5.2. COMAN

ComAN (Component Object Model Active Network) [34], implémenté au département des technologies électriques et électroniques de l'université de Canterbury en Nouvelle-Zélande, est une architecture de réseau actif de bout en bout à base de middleware.

Architecture

COMAN a été conçu pour être une architecture simple et légère de réseau actif de bout en bout (Figure 2.4), facilement installée dans n'importe quelle topologie de réseaux, permettant ainsi une grande flexibilité dans le traitement et le routage des paquets pour n'importe quel service ou application future.

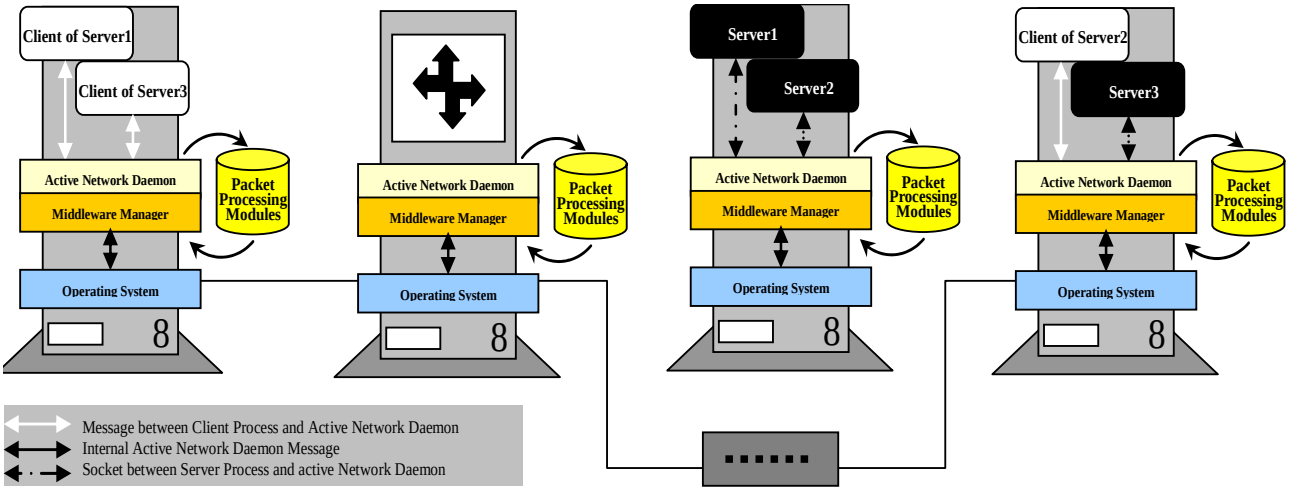

Figure 2.4. Architecture de COMAN

Cette architecture présente un atout qui est le niveau élevé de connectivité présent dans les applications réseaux qui utilise les services de COMAN et dans les autres réseaux actifs à base de middleware. De même, COMAN a été conçu pour avoir un niveau élevé d'authentification pour les utilisateurs et les applications.

Caractéristiques du système

- *Flexibilité*

Les paquets définis par les utilisateurs, ou même les instructions de routage, appelés modules de routeur, peuvent être définis dans plusieurs langages entre autre C, C++, Java etc. Ces mo-

dules sont des composants DCOM [17] de Microsoft, ils sont liés à un environnement de Microsoft Windows et ils s'exécutent dans l'espace d'adressage utilisateur.

Pour simplifier, COMAN utilise la logique du transfert IP conventionnel pour le routage à moins que l'utilisateur demande d'utiliser une méthode qu'il a définie. COMAN supporte n'importe quel nombre de connexions offrant le transfert à condition que les paquets soient placés en FIFO dans la queue. De plus COMAN offre une simple API.

-*Sécurité*

Comme étant un service DCOM, l'authentification des composants des réseaux actifs est maintenue par le système d'exploitation, au niveau des utilisateurs et de la machine. Par conséquent il est possible de restreindre les utilisateurs qui accèdent à COMAN en essayant d'invoquer des méthodes, de faire le chargement des modules, de s'enregistrer, etc. Si un module malveillant essaye d'être exécuté dans le réseau, seules les données dans l'espace d'adresse du service local de COMAN sont affectées car COMAN dans ce cas fonctionne en mode sûr (safe mode). De la même façon, l'architecture de COMAN utilise un mécanisme d'isolation des processus, écartant ainsi l'accès non autorisé aux mémoires des clients et isolant toute collision éventuelle avec d'autres utilisateurs ou aussi avec des services de COMAN.

- *Accessibilité*

Plusieurs instances distantes des réseaux actifs de COMAN peuvent communiquer pour former un seul réseau actif logique, avec une architecture qui revient à un transfert IP conventionnel pour tous les nœuds intermédiaires passifs. De la même façon d'autres réseaux actifs à base de middleware sont capables de communiquer directement avec COMAN grâce à l'attachement (binding) des middlewares qui existe entre DCOM, CORBA et Java.

2.5.3. HABA

HABA (Hyper Active Components Architecture) [35] est implémenté au centre informatique de la faculté d'ingénierie à l'université Saint-Joseph au Liban, à l'aide de composants middleware Java : Les Enterprise Java Beans (EJB) [21] dont le paradigme est : «Move, Plug and Work» d'une plateforme à l'autre sans besoin de modification de la source.

En fait, c'est l'environnement d'exécution seulement qui est implémenté sous forme de composants, sachant que le serveur d'application joue le rôle de NodeOS, et les applications actives sont intégrées sous le format compressé « jar ». Le serveur d'application permettra de gérer à travers un conteneur les instanciations de composants, les accès aux ressources, ainsi que la gestion de la sécurité.

Les composants de l'environnement d'exécution ne sont qu'une simple modélisation des fonctionnalités génériques d'un nœud actif, fonctionnalités qui peuvent changer dynamiquement par remplacement, ajout ou suppression de l'une de ses composants.

Ainsi pour assurer les fonctionnalités d'un nœud actif, les composants suivants doivent être installés:

- Un « Listen Bean » chargée d'attendre et de recevoir les capsules arrivant et de communiquer à l'« Applications Bean »
- L'«Applications Bean» qui sera chargé de gérer l'accès au cache d'Applications Actives pour chercher le code requis par la capsule reçue. Si ce code est trouvé, il s'exécute et la capsule sera routée vers le nœud suivant. Sinon le Listen Bean sera responsable du téléchargement de ce code.

Tandis que pour assurer les fonctionnalités d'un nœud de contrôle, les composants suivants doivent être installés (Figure 2.5):

- Un « Listen Bean » chargé d'attendre et de recevoir les capsules arrivant au nœud de contrôle.
- Un « Routing bean » chargé de gérer tous les aspects du routage des requêtes. Comme par exemple chercher la route sur laquelle un déploiement de code doit être effectué, et passer ces informations au « Deployer Bean ».

- Le « Deployer bean », sera alors responsable du déploiement du code sur les nœuds correspondant au chemin fourni par le Routing Bean.
- Le code requis sera cherché par l'«Applications Bean» qui sera chargé de gérer l'accès au cache d'Applications Actives.
- « History bean » chargé de gérer l'aspect archivage des requêtes, en vue d'optimiser le déploiement et de ne pas déployer un code sur un nœud qui le possède déjà.

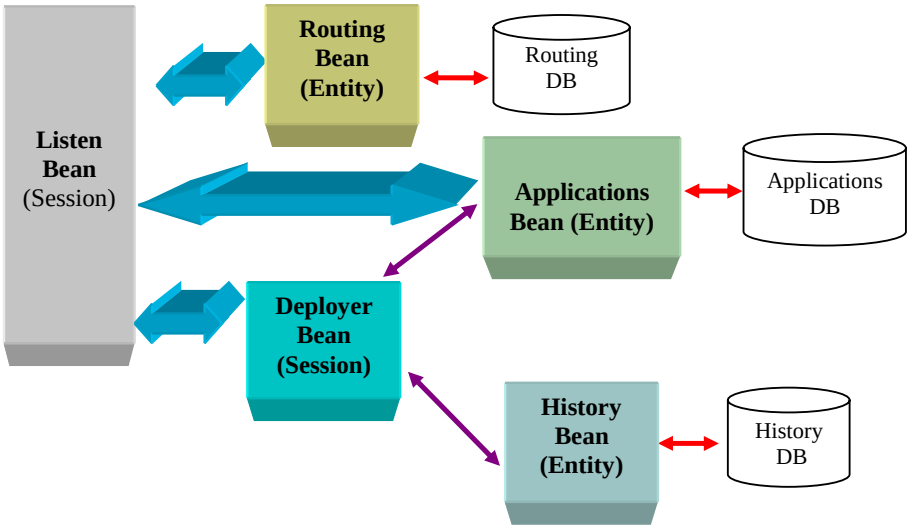

Figure 2.5. Nœud de Contrôle dans HABA

HABA utilise le format « jar » des services à déployer pour ce qu'il assure aux niveaux portabilité du code, économie de la taille de ce code, et variété des types de fichiers pouvant intégrer un même fichier « jar ».

Au niveau du déploiement, HABA utilise l'approche nœud actif. Mais la possibilité d'attribuer dynamiquement différents comportements aux nœuds actifs, permet à HABA de diviser le réseau en plans et d'offrir ainsi plusieurs modes de déploiement sur ces différents plans architecturaux. Parmi ces possibilités on distingue un mode de déploiement parallèle sous contrôle, un mode de déploiement à la ANTS, ainsi qu'un mode de déploiement séquentiel à la demande.

En terme de sécurité, HABA offre différents niveaux de sécurité suivant le profil du service à déployer. Ceci est facilité par les différents plans de l'architecture du réseau utilisés pour le déploiement. L'authentification du code déployé ainsi que la protection des nœuds actifs sont assurés par un même modèle intégré qui consiste à déployer des certificats sur les nœuds, suivant le niveau de sécurité requis par le service à déployer.

La confidentialité dans le transport du code pourrait être assurée par simple configuration du serveur d'application.

2.6. RESEAUX ACTIFS ET GESTION DES RESEAUX

2.6.1. Architecture des réseaux à base des politiques

Comme décrit dans la section A.2, la gestion par politique ou PBM [123] pour Policy-Based Management, est une technologie émergente qui présente de nombreuses différences par rapport aux anciennes techniques comme SNMP [124]. Celle-ci concernait en grande partie la détection et réparation des problèmes dans un réseau. En effet, la nouvelle optique consiste à considérer le réseau fonctionnel et à permettre à l'administrateur de le régler pour atteindre certains objectifs définis à l'avance.

Le groupe RAP [42] (*Ressource Allocation Protocol*) de l'IETF [126] est à l'origine de l'architecture utilisée de nos jours pour la gestion par politiques. Elle se caractérise par sa simplicité puisqu'elle ne regroupe que quatre composants.

Dans cette architecture, les nœuds d'un réseau sont appelés PEPs (*Policy Enforcement Point*). Lors d'une arrivée de flux à un nœud, ce nœud interroge une entité, qui a une vue globale du réseau, pour savoir comment réagir. Cette entité qui est l'élément décisionnel s'appelle le PDP (*Policy Decision Point*). Le PDP s'appuie sur un ensemble de règles de gestion des ressources système et réseau pour répondre à la demande des nœuds. Ces règles sont mises en place par l'administrateur à partir d'une console d'administration conviviale. La politique (section A.5) est justement la définition de l'ensemble de ces règles capables de gérer et de contrôler le comportement des éléments du réseau (services, équipements, utilisateurs). Cette politique répond aux questions concernant l'usage des ressources par les utilisateurs, les applications prioritaires, la différenciation des services en fonction des besoins… Le fait que les règles soient distribuées sur les nœuds permet une gestion plus globale des objectifs à atteindre. Parmi ces derniers nous pouvons par exemple citer la Qualité de Service, la sécurité ou encore les profils utilisateurs.

Le PDP recherche les règles de politique dans un annuaire des politiques appelée *Policy Repository* (on préconise LDAP [127] pour l'accès à l'annuaire). Mis à part son rôle d'identification des règles, le PDP doit pouvoir traduire celles-ci dans des formats compréhensibles par les PEPs comme la PIB.

La communication entre un PEP et un PDP est réalisée à l'aide de protocole de transaction comme COPS (Common Object Policy Service) [43] (section A.3). Ce protocole de signalisation, développé par l'IETF, permet le transport des demandes de configuration des PEPs et des réponses envoyées par le PDP concernant les politiques à appliquer. C'est un protocole très flexible de type requête/réponse fondé sur TCP. Il consiste à l'établissement de connexion du PEP vers le PDP pour l'envoi de requêtes, la réception des décisions qui les concernent et optionnellement, l'envoi de rapports au PDP sur le déroulement des opérations. Le PEP et le PDP peuvent à tout moment mettre à jour respectivement une requête et une décision.

COPS-PR (*Policy Provisioning*) [44] (section A.4) est l'un des *Client-type* de COPS. Il concerne, comme son nom l'indique, le mode *Provisioning* du protocole dans lequel un PEP ne contacte pas le PDP à chaque réception d'un flux pour savoir comment réagir (contrairement au mode *Outsourcing*). Ceci grâce, entre autres, à la possibilité au PDP d'envoyer les décisions aux PEP de manière non sollicitée (*unsolicited*) pour la mise à jour de leurs politiques.

Dans COPS-PR chaque client doit maintenir une certaine base de données où les configurations sont stockées. Cette base de données est la PIB [45]. Elle définit le modèle de données des informations et peut être décrite comme un espace de noms sous forme d'un arbre conceptuel. Les branches de celui-ci représentent la structure des données ou Classes de Provisions (*Provisioning Classes PRC*) et les feuilles représentent des instances de ces classes ou Instances de Provisions (*Provisioning Instances PRI*) qui sont référencées de manière unique. On peut avoir plusieurs instances de n'importe quel PRC.

La définition des PIBs se fait conformément à une spécification bien déterminée appelée SPPI [46] pour *Structure of Policy Provisioning Information*. Celle-ci présente de nombreuses constructions ainsi que leurs sémantiques, pouvant être utilisées pour la définition des PIBs.

2.6.2. Motivation pour utiliser les Réseaux Actifs dans la Gestion des Réseaux

L'intérêt d'utiliser des paquets et des nœuds actifs pour la gestion du réseau est multiple:

- La dynamicité des nœuds de gestion facilite le déploiement et la configuration du réseau, ainsi que la définition des fonctions de gestion.
- Une fonction de gestion devient un programme intégrable dans le réseau et non plus une description statique d'interface.
- La gestion est totalement intégrée au réseau, et pas superposée (comme avec la signalisation).

Des travaux ont été menés chez Lucent, visant à coupler un nœud actif à un routeur pour accomplir des fonctions de gestion. Le nœud est adressable par paquets actifs. Les domaines de gestion sont:

- le contrôle adaptatif
- la configuration d'équipements
- la détection des composants
- la gestion de la sécurité

Mais cette approche n'a pas de motivation spécifique aux réseaux actifs; l'architecture de gestion reste centralisée sur certains nœuds, et les fonctions de gestion ne sont pas modifiables.

L'avantage principal présenté par les réseaux Actifs est leur possibilité de permettre aux services utilisateurs de définir leurs propres mécanismes pour traiter des paquets dans des nœuds intermédiaires, ainsi il leur est possible de télécharger dynamiquement de nouvelles fonctionnalités directement à l'intérieur des routeurs selon leurs activités.

L'utilisation des services de réseau est répandue actuellement. L'importance des services exige qu'ils soient garantis au moyen d'un contrôle continu de sorte que des interventions instantanées soient assurées en cas de défaut.

Le concept de la disponibilité dans les réseaux de transmissions est devenu une caractéristique indispensable de la conception d'architecture de réseau. Pour améliorer la disponibilité dans la gestion de réseau, une gestion proactive est utilisée. La gestion proactive opère sur les symptômes qui prévoient des événements peu coopératifs afin de les éviter. Dans ce scénario, les réseaux actifs peuvent être un instrument approprié pour l'implémentation d'un système complet de gestion de réseau à base de politique. Ce nouveau système doit fournir :

- La possibilité d'étendre dynamiquement les informations dans la base de gestion.
- Un protocole pour télécharger cette information directement dans le nœud actif cible.
- La possibilité d'améliorer la fonction de gestion dans le nœud actif par recombinaison avec des services de gestion existants.

Les projets qui analysent les synergies qui peuvent être obtenues en regroupant des technologies de réseaux actifs et la gestion par politique ne sont pas nombreux.

Certains des travaux les plus connus et admis sont Seraphim [54], ANDROID [55], PxP [56], Polynet [58], Spécification des Politique pour les réseaux programmables [59][60] et FAIN [60]. Tous ces travaux se concentrent sur la gestion des réseaux actifs à base de politique.

D'autre part, [66] propose d'utiliser le réseau actif comme un environnement de développement et d'essai pour les services et protocoles de réseau. Le but de cette proposition est de montrer comment l'environnement actif peut rendre plus facile et plus rapide le processus de développement et de tests des protocoles. Ce qui évite l'utilisation du matériel cher ou des simulations inexactes qui demandent un temps très grand. Comme étude de cas, [66] traite avec le protocole d'IETF pour la gestion de réseau à base de politique, COPS, comme protocole d'essai.

Dans les sections qui suivent, nous allons décrire les propositions et réalisations suivantes de tels systèmes :

- Smart Packets part du constat que la supervision engendre trop de trafic parce que les centres de supervision font du polling, qui est mal adapté à un grand nombre de nœuds. Les nœuds de routage doivent donc être actifs.
- FAIN est le modèle de système de gestion de réseau actif. C'est une adaptation du modèle de politique de l'IETF.
- A-PBM (Active policy-Based Management) permet l'interopérabilité entre différents domaines de gestion de ISPs (Internet Service Provider), répondant à des exigences de bout en bout requises par des utilisateurs. A-PBM adopte le modèle de gestion par politique, proposé par l'IETF et y inclut des capsules pour la communication entre les différents composants de ce modèle.

2.6.3. Smart Packets de BBN

L'approche Smart Packets [12], développée chez BBN, est une approche intégrée motivée par la croissance exponentielle de la capacité de traitement disponible sur les nœuds des réseaux,

la nécessité de réduire le plus possible le trafic de gestion qui a tendance à exploser avec le nombre de nœuds et finalement le besoin de décentraliser les fonctions de gestion afin de soulager les stations de supervision. Afin de répondre à ces besoins, Smart Packets utilise intelligemment le réseau actif pour décentraliser la gestion.

Le principe de cette approche est le suivant: un paquet de supervision est un programme véhiculé dans un paquet actif. La principale restriction sur ces paquets est la taille maximale fixée à 1Ko afin que ceux-ci tiennent dans une trame Ethernet. Dans les différents nœuds du réseau, l'approche propose une architecture d'accueil et d'exécution de ces fonctions de gestion.

Le format de paquets Smart Packets est donné dans la Figure 2.6. On y retrouve l'entête de trame IP, l'encapsulation dans le standard de paquets actifs ANEP et finalement le paquet Smart Packet. Celui-ci est composé d'un numéro de version, d'un type, d'un contexte et des données véhiculées. Les différents types possibles sont:

- Program: le paquet contient un programme envoyé vers un agent ;
- Data: le paquet contient des données renvoyées vers la station de supervision depuis un agent;
- Error: le paquet véhicule une erreur d'un agent vers un superviseur ;
- Message: permet à un agent ou un gestionnaire d'envoyer des données qui ne sont ni un programme, ni une réponse à un envoi et une exécution de programme.

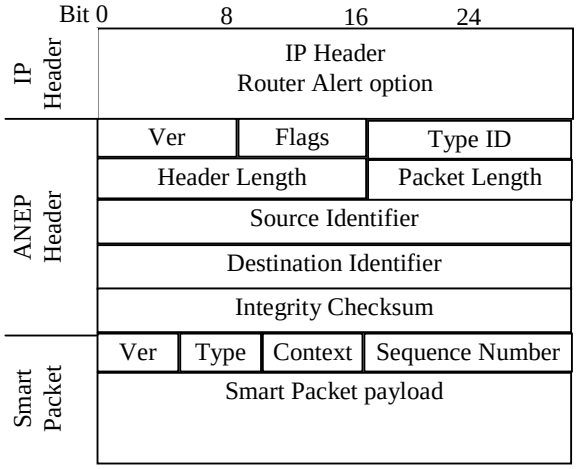

Figure 2.6 - Le paquet Smart Packets

Permettre la programmation de fonctions de gestion requiert l'utilisation d'un langage de programmation. Smart Packets définit son propre langage de programmation de paquets. Appelé Sprocket, celui-ci est un sous-ensemble de C++ (simplifié et sans pointeur) étendu par des primitives d'accès et de manipulation d'informations de gestion issues d'un agent SNMP (Get, Set, GetNext) ainsi que par des primitives de manipulation de paquets actifs.

Tout code Sprocket est compilé en un assembleur spécifique appelé Spanner. La principale différence avec les assembleurs classiques est que celui-ci ne fournit pas d'instruction d'accès mémoire direct mais uniquement via des variables déclarées ainsi qu'une pile pour des raisons de portabilité. Tout nœud capable de traiter des Smart Paquets dispose d'une machine virtuelle qui démultiplexe les paquets ANEP [13] et exécute le code Spanner. L'authentification et la confidentialité sont assurées par un chiffrement des champs non mutables d'un paquet ANEP, couplé à l'utilisation d'un système de clefs publiques/privées.

Les fonctions d'autorisation sont implantées par des listes de contrôle d'accès par agent. Ces listes définissent pour chaque client connu, les fonctions de manipulation de MIB possibles ainsi que la liste des objets de celle-ci accessibles par le client et enfin pour chaque objet, les opérations précises autorisées.

En conclusion, l'approche SmartPackets est l'une des premières approches actives publiées.

Dédiée à la supervision, elle est particulièrement intéressante car elle justifie totalement l'utilisation de l'actif pour des fonctions de supervision et démontre l'apport d'un langage de programmation de paquets dédié à la fonction visée, concept qui sera repris plus tard dans les réseaux actifs via les langages spécifiques de domaines pour des fonctions autres que la supervision. Cependant, l'approche prototype développée souffre d'un certain nombre de limites, notamment sur la taille de programmes, limitant les fonctions déportées à des traitements restreints. Finalement, très peu d'expérimentations et aucun résultat chiffré sur l'apport de ce type de délégation n'a été reporté.

2.6.4. L'approche FAIN pour la gestion de réseau

L'architecture de gestion développée dans le projet FAIN est une synergie entre la gestion de réseau à base de politique et les technologies de réseaux actifs afin de répondre aux exigences principales pour la gestion des réseaux actifs.

La gestion du réseau actif [65] exigera les éléments suivants:

- Politiques : Conception des politiques de gestion requises pour contrôler le réseau et les nœuds actifs.
- Composants de gestion dans les nœuds actifs: Conception des composants de gestion pour les nœuds actifs, qui exécuteront des politiques au niveau de ces nœuds et surveilleront l'utilisation de leurs ressources. L'exécution des politiques implique la traduction des politiques cible en des configurations de ressources du nœud.
- Nœuds de Gestion: Un ensemble de nœuds de gestion qui supportent des mécanismes pour permettre à des administrateurs de réseau de contrôler les réseaux actifs, y compris l'installation et le traitement des politiques des réseaux.

L'architecture de gestion de FAIN est une architecture de gestion à deux niveaux qui se compose d'une station de gestion au niveau réseau (NMS pour Network Management System) et de plusieurs stations de gestion au niveau élément (EMS pour Element Management System), un EMS étant implémenté sur chaque nœud à contrôler. Comme dans TMN [67], un système de gestion d'éléments (EMS) contrôle un ou plusieurs types spécifiques d'éléments de réseau (NE pour Network Element) de télécommunications. Typiquement, le EMS contrôle les traitements et les configurations dans chaque NE mais ne contrôle pas le trafic entre les différents NEs dans le réseau. Pour assurer la gestion du trafic entre lui-même et tout autre NE, le EMS communique avec des systèmes de gestion de réseaux (NMS) de niveau plus élevé. La Figure 2.7 montre une version simplifiée de l'architecture de gestion de FAIN.

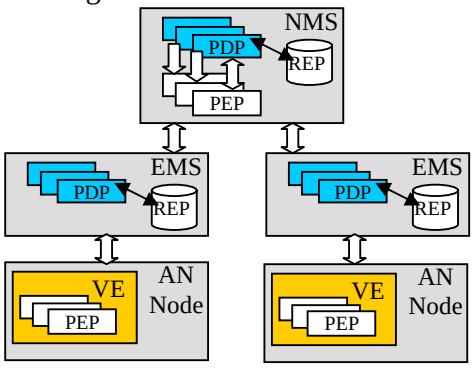

Figure 2.7. Architecture de gestion de FAIN

L'architecture de gestion à base de politique d'un élément de réseau actif (PBANEM) est illustrée par la Figure 2.8. C'est une adaptation du modèle de politique de l'IETF [50] pour la gestion des nœuds FAIN, dans lequel les composants principaux du modèle de politique sont le point de décision de politique (PDP pour Policy Decision Point) et le point de renforcement de politique (PEP pour Policy Enforcement Point) [51]. Le PDP s'occupe de deux aspects principaux de PBNM. Tout d'abord, il prend en charge la récupération des politiques venant des NMS ou des applications actives, et procède à leur distribution aux PEPs appropriés. Deuxièmement, il

détermine les politiques à appliquer dans chaque scénario selon sa connaissance du statut du nœud. Le PEP est responsable d'appliquer ces politiques sur les ressources contrôlées appropriées. La détection et la résolution locale de conflit [52] sont, en fait, une des fonctionnalités principales du PDP. Le module de vérification de conflit est implémenté au niveau du PDP (et pas au niveau de chaque PEP) puisque la détection locale de conflit exige la compréhension de la sémantique des politiques (conditions et actions). Cette connaissance de la sémantique des politiques peut être dynamiquement améliorée en téléchargeant des extensions au composant de contrôle de conflit du PDP, au PEP qui va appliquer cette politique ou au niveau de la politique elle-même.

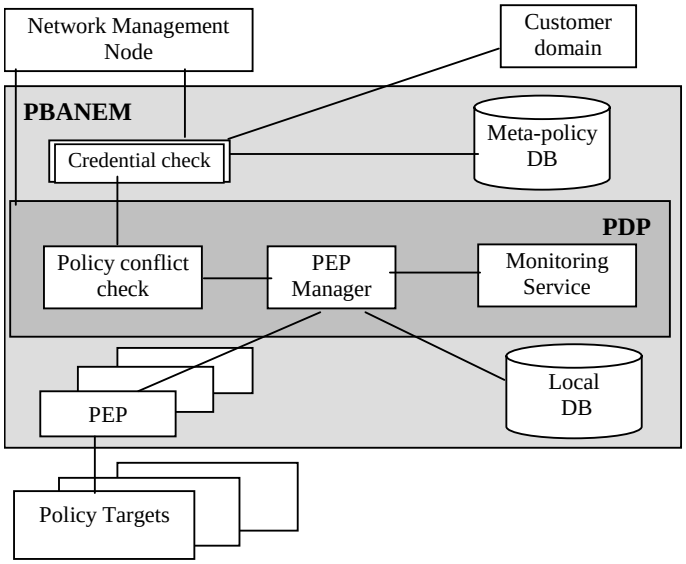

Figure 2.8. Gestion des éléments de réseaux actifs à base de politique

Pour résoudre les problèmes non prévus par le PDP (par exemple, ceux qui viennent d'une application active), une approche active a été adoptée. Dans cette approche, les paquets actifs contenant les métapolitiques, peuvent contenir également un code permettant l'exécution des nouvelles politiques. Ce code, qui n'est autre que l'interpréteur des conditions ou des actions de ces politiques, s'installe dynamiquement sur le PDP. Ceci permet au PDP d'acquérir des nouvelles connaissances sur les classes de politiques utilisées par le PEP.

PBANEM est également responsable de la création, la suppression et la modification des EEs. Avant la création d'un EE, PBANEM consulte les politiques en vigueur, pour vérifier si la personne qui réalise cette tâche a les privilèges nécessaires pour l'accomplir et s'il existe sur le nœud des ressources suffisantes à allouer pour ce EE. De même, PBANEM gère les priorités des EEs pour s'assurer que les EEs les plus critiques ont toujours suffisamment de ressources pour qu'ils s'exécutent.

2.6.5. Gestion active à base des politiques (A-PBM) des réseaux multi-domaines

Afin de fournir la qualité de service requise par les entreprises qui sont situées dans différents points du monde, les ISPs doivent coopérer entre eux. Par conséquent les ISPs ont besoin de fournir à leurs clients des services différenciés basés sur les accords convenus au niveau du service (SLAs pour Service Level Agreements) qui permettent à des clients de contrôler la qualité de service fournie par l'ISP. Cette qualité de service nécessite une connexion qui s'étend à travers plusieurs domaines ISP, ceci rend la gestion de la communication de bout en bout plus complexe. En effet, un changement dans les obligations du client implique un certain nombre de changements dans chaque réseau ISP.

L'approche gestion par politique est le modèle le plus adapté pour la définition et l'application des contrats concernant la qualité de service. Mais à cause de la complexité du pro-

cessus de gestion de politique dans le contexte des opérateurs multi-domaines et de leurs implémentations ainsi que les problèmes de sécurité, [57] a proposé de remplacer l'approche de gestion à base de politiques client/serveur par une approche de capsule. Cette architecture de gestion, appelée "PBM actif" (A-PBM), utilise une base dynamique de l'information de gestion (DMIB) pour installer, conserver et contrôler de nouveaux services par l'intermédiaire des réseaux actifs. Ceci permet d'améliorer la scalabilité et offre une meilleure flexibilité aux utilisateurs et aux opérateurs.

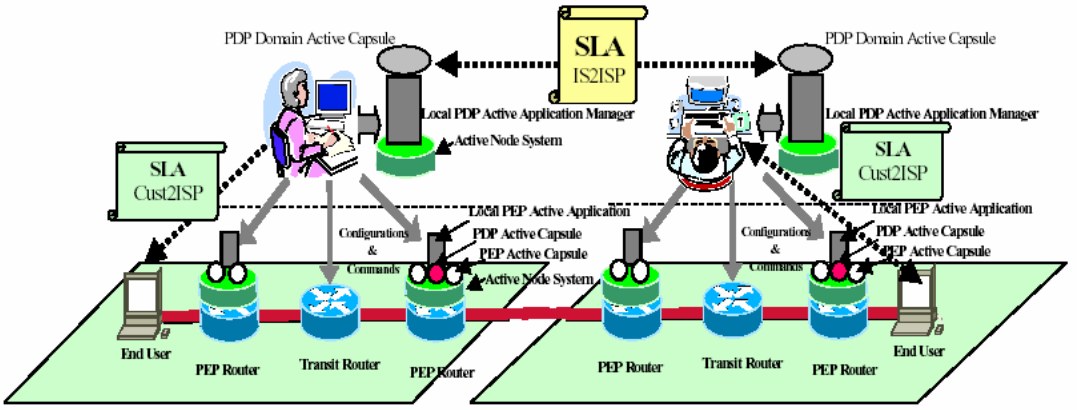

Figure 2.9 - Architecture de A-PBM inter-domaine

L'architecture A-PBM (Figure 2.9) définit un ensemble de services et de capsules actifs selon leurs rôles respectifs dans cette architecture:

- **Service actif du PEP local**: C'est un service actif qui joue le rôle de PEP. Il contient les routines locales de contrôle et de gestion. Il exécute principalement des fonctions de paramétrage et d'application des règles, ainsi que l'envoi de capsules PEP quand il a besoin d'interagir avec l'administrateur de service actif du PDP local pour demander des décisions.
- **Capsules PEP**: Celles-ci sont employées principalement en tant que capsules autonomes de négociateur entre le PEP et le PDP dans le même domaine. Les capsules de PEP sont employées pour obtenir des décisions de PDP. Les capsules de PEP diffusent toute information concernant la connexion en cours. Elles sont envoyées par le PEP au PDP afin de notifier un événement particulier dans le réseau (demande d'ouverture de RSVP [69], dégradation de QdS, etc..).
- **Administrateur de service actif du PDP local**: C'est un service actif qui prend la décision concernant l'information qui est transportée par les capsules du PEP. Il interagit avec les diverses bases de données (BDD des règles de politiques, MIB, BDD de sécurité, etc..) afin de rechercher les règles qui peuvent être déclenchées. Une fois que les décisions sont identifiées, celles-ci sont envoyées à l'aide des capsules PEP. Si une configuration liée à de nouvelles politiques est définie, ce service envoie des capsules PDP aux services actifs PEP distants pour effectuer la nouvelle configuration. Il prend également en considération des interactions inter domaine, quand une décision doit être négociée avec l'administrateur de service actif du PDP d'un domaine à distance. Dans ce dernier cas, il envoie des capsules PDP Inter-Domaine.
- **capsules PDP**: Quand un service actif de PDP a pris une décision, il envoie une capsule PDP pour appliquer des politiques directement dans les composants des services actifs de PEP dans tous les éléments du réseau qui sont concernés par cette nouvelle décision.
- **Capsules PDP Inter-Domaine**: Quand un PDP doit prendre une décision liée à une connexion inter-domaine, il doit identifier l'ensemble de domaines distants nécessaires pour la connexion et envoyer une capsule Inter-Domaine pour négocier les conditions des services requis par le client.

Ainsi, pour permettre le déploiement et l'installation dynamique des services actifs, ces derniers sont définis comme étant des objets DMIB (Dynamic Management Information Base).

DMIB est une nouvelle approche dynamique de MIB, proposée par [57] et facilitant la définition et le déploiement d'une façon dynamique des objets.

Pour valider cette approche, un prototype a été implémenté en utilisant la plateforme de réseaux actifs ANTS.

2.7. UTILISATION DES ONTOLOGIES DANS LA GESTION DES RESEAUX

2.7.1. Web sémantique et Ontologie

Comme décrit dans l'Annexe B, le Web sémantique [140] est une extension du Web traditionnel, dans lequel les informations sont décrites d'une manière précise et définitive pour permettre une meilleure coopération entre les utilisateurs et leurs machines.

Qui dit sémantique dit obligatoirement « sens » donné à un certain vocabulaire. Il est donc nécessaire de mettre en place une méthode de développement de vocabulaires spécifiques à un domaine particulier : c'est le rôle des ontologies.

Comme décrit dans la section B.2.4, une ontologie définit les termes utilisés pour décrire et représenter un champ d'expertise. Elle associe les concepts de base d'un certain domaine et les relations de ces concepts de manière compréhensible par les machines. Elle encode la connaissance d'un domaine particulier ainsi que les connaissances qui couvrent d'autres domaines, permettant ainsi la réutilisation de ces connaissances.

RDF (*Resource Description Framework*) [70] est la base technologique de la vision du Web Sémantique de Tim Berners-Lee. RDF est un Meta-langage XML [23] permettant de représenter des graphes sous forme d'ensembles de "triplets" élémentaires. RDF-S (*RDF Schema*) [71] est un langage de schéma permettant de modéliser et de valider des documents RDF. DAML+OIL (DARPA Agent Markup Language plus Ontology Inference Layer) [73] est un langage utilisant RDF et RDF Schema pour bâtir des systèmes d'inférences.

OWL [74] est un vocabulaire XML basé sur RDF et permettant de définir des ontologies Web structurées. OWL est un langage d'ontologie très complet grâce aux caractéristiques suivantes [81]:

- Il permet la définition :
  - des partitions, aussi bien d'une documentation de classes,
  - d'attributs, mais sans pouvoir distinguer ceux d'une classe et ceux d'une instance, ceci en réduisant le domaine (local ou global),
  - des propriétés, comme la cardinalité ou la contrainte sur un type.
- Des sous-classes peuvent être créées par héritage multiple et décompositions disjointes et exhaustives. Une classe peut également être complémentaire d'une autre classe.
- OWL permet la définition d'axiomes logiques de premier ordre selon les relations algébriques de type symétrie, transitivité et unicité. Les classes et propriétés peuvent être également universelles ou extensibles.
- La possibilité de définir des déclarations, des relations et des instances de concepts.

Le langage OWL ajoute de nombreuses définitions au Web Sémantique, mais il pose certains problèmes à cause de quelques limitations en particulier dans la définition des propriétés. Pour résoudre ce problème, SWRL [143] (Semantic Web Rule Language ou langage de règles du Web sémantique), une extension à OWL, a été développée pour rendre le langage encore plus expressif pour la description de ces propriétés.

2.7.2. Ontologie pour les spécifications de l'information de la gestion

Il existe plusieurs modèles de gestion de réseau utilisant différentes technologies pour la gestion de ressources, telle que SNMP [124], CMIP [163] et WBEM [162]. Chaque modèle a défini son propre langage de définition de l'information de gestion:

- SMI (Structure of Management Information), avec ses différentes versions, pour le SNMP.
- GDMO (Guidelines for Definition of Managed Objects) pour CMIP.
- MOF/CIM (Managed Object Format/Common Information Model) pour WBEM.

D'autre part, XML [23] a émergé dans le monde de l'Internet comme un format de représentation standard, permettant la définition, la transmission, la validation et l'interprétation de l'information. Ensuite plusieurs approches sont apparues, utilisant XML pour décrire l'information de gestion, y compris ceux proposés par la DMTF et le groupe de travail de configuration du réseau de l'IETF. Cependant, les formats XML seuls ne permettent pas de donner une définition sémantique formelle à ces informations.

Le besoin d'un modèle conceptuel devient nécessaire pour la description de chaque ressource. De nouveaux langages évolués appelés ontologies voient alors le jour. Ils permettent la description abstraite d'un domaine quelconque d'informations en définissant un ensemble de concepts, une hiérarchie, des relations entre les données et surtout des règles qui gouvernent ces concepts.

La représentation par une ontologie, des informations relatives aux ressources permet d'intégrer plus de sémantique et d'intelligence au niveau des nœuds et de leur donner la possibilité de jouer un rôle plus important dans le processus de traitement des flux selon certains comportements prédéfinis.

Il existe deux groupes d'ontologies :

- Les ontologies légères (Lightweight) qui concernent la modélisation d'information d'un certain domaine de manière simple sans axiomes ou restrictions.
- Les ontologies lourdes (Heavyweight) qui permettent la représentation de liens sémantiques plus spécifiques et qui permettent de faire des déductions sur la connaissance qu'elles contiennent.

Les modèles existants de l'information de gestion peuvent être considérés comme appartenant au groupe des ontologies légères: Les modèles comme CIM [75] définissent l'information du domaine de gestion d'une façon formelle et sont consentis par les groupes de travail. Cependant, ils n'incorporent pas la sémantique nécessaire à la déduction de nouvelles bases de connaissance à partir de celles existantes.

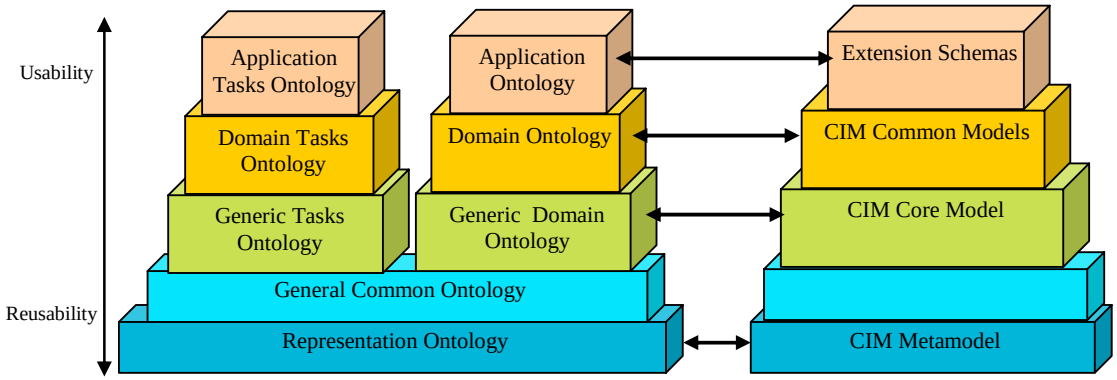

Figure 2.10. Correspondance entre les architectures de CIM et de l'Ontologie

Mais, il est toujours possible de faire une certaine correspondance entre l'architecture de types d'ontologie [76] et la CIM, comme le montre la Figure 2.10. Les ontologies sont généralement basées sur d'autre ontologies, suivant une structure de pyramide dans laquelle les ontologies les plus générales et les plus réutilisables, sont au niveau bas, et les plus utilisables et les plus spécifiques sont au sommet. CIM a une structure semblable dans laquelle seulement un niveau commun général d'ontologies n'est pas présent. En outre, CIM ne supporte pas des fonctionnalités d'ontologie permettant de manipuler sa base de connaissance par des méthodes de résolution de problèmes (Problem Solving Methods) [76].

Un autre point à accentuer est que l'ontologie peut être appliquée non seulement pour définir l'information, mais pour ajouter également l'expressivité et le raisonnement. Un système de gestion de réseau qui a un moteur de déduction, comme dans [77], pourrait employer la même ontologie pour définir toute l'information de gestion et également définir ses règles de comportement d'une manière unifiée.

Il existe déjà des travaux qui appliquent des ontologies pour définir des politiques et l'information de gestion. Certains de ces travaux sont décrits dans les sections suivantes.

2.7.3. Gestion des politiques à base de DAML dans l'environnement KAoS

KAoS (Knowledgeable Agent-oriented System) [90] est un environnement créé pour satisfaire les besoins dynamiques et complexes des applications à base d'agents. KAoS est une collection de services orientée composants, compatibles avec plusieurs plateformes populaires à base d'agents, y compris Nomads [91], la grille CoABS [92] de DARPA, l'environnement ALP/Ultra*Log Cougaar [94] de DARPA, CORBA [15] et Voyager [95]. L'adaptabilité de KAoS est due principalement à une infrastructure extensible basée sur JAS (Java Agent Services) [96] de Sun.

[165] définit des services de domaines et des services de politiques en vue d'assurer le déploiement et l'exécution des agents KAoS dans un large spectre d'environnements opérationnels. Les services de domaines KAoS permettent de structurer dans des domaines et des sous domaines, des groupes des composants logiciels, des personnes, des ressources et d'autres entités. Ceci permet de faciliter d'une part la collaboration entre les agents et d'autre part l'administration externe des politiques. Les services de politiques KAoS permettent la spécification, la gestion, la résolution des conflits et l'application des politiques dans les domaines. Les politiques sont représentées en DAML+OIL sous forme d'ontologies appelées KPO (KAoS Policy Ontologies).

Les Ontologies des politiques dans KAoS (KPO) distinguent les autorisations (c.à.d. les contraintes qui permettent ou interdisent une certaine action) et les obligations (c.à.d. les contraintes qui exigent qu'une certaine action soit effectuée, ou bien servent à renoncer à une telle condition) [99].

Les fonctionnalités de gestion de politiques peuvent être divisées en deux catégories : génériques, et celles spécifiques aux applications ou à la plateforme.

Les fonctionnalités génériques incluent des composants réutilisables pour :
- La création et la gestion de l'ensemble des ontologies de base.
- Le stockage, la résolution des conflits et l'interrogation.
- La distribution et l'application des politiques.
- La révélation des politiques.

Pour les fonctionnalités spécifiques aux applications ou à la plateforme, l'architecture KAoS peut être étendue par la définition des nouvelles ontologies décrivant les entités spécifiques aux applications ou à la plateforme ainsi que les types d'actions appropriés.

La version en cours de KPO définit des ontologies de base pour les actions, les acteurs, les groupes, les endroits, les entités diverses liées aux actions (par exemple, ressources informatiques), et les politiques. Il y a approximativement 80 classes et 40 propriétés définies dans les ontologies de base. L'ontologie des acteurs par exemple contient les classes qui représentent les personnes et les composants logiciels qui peuvent être soumis à des politiques. L'ontologie des actions définit les différents types d'activités de base comme la surveillance, la communication, le déplacement, l'accès, etc.

Mais, pour une application donnée, ces ontologies de base seront encore étendues avec des classes, des individus, et des règles additionnels. Ces nouvelles entités utilisent les concepts définis dans les ontologies de base comme étant des super concepts. Ceci permet à l'architecture de

discerner des concepts spécialisés en interrogeant l'ensemble des ontologies à la recherche d'une sous classe ou d'une sous propriété d'un concept défini au niveau des ontologies de base.

Dans KaoS, une politique n'est autre qu'une déclaration permettant ou contraignant l'exécution d'un certain type d'action par un ou plusieurs acteurs par rapport à de divers aspects d'une certaine situation. Les politiques sont représentées par une définition DAML de la classe des acteurs définissant les différentes instances d'actions qui sont contrôlées par cette politique. Voici un exemple de politique défini dans DAML:

```
<daml:Class rdf:ID="P1Action">
 <rdfs:subClassOf rdf:resource="#CommunicationAction" />
 <rdfs:subClassOf>
  <daml:Restriction>
     <daml:onProperty rdf:resource="#performedBy" />
     <daml:toClass rdf:resource="#MembersOfDomainArabello-HQ" />
   </daml:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <daml:Restriction>
    <daml:onProperty rdf:resource="#hasDestination" />
    <daml:toClass rdf:resource="#notMembersOfDomainArabello-HQ" />
  </daml:Restriction>
 </rdfs:subClassOf>
</daml:Class>
<policy:NegAuthorizationPolicy rdf:ID="P1">
  <policy:controls rdf:resource="#P1Action" />
  <policy:hasSiteOfEnforcement rdf:resource="#ActorSite" />
  <policy:hasPriority>1</policy:hasPriority>
  <policy:hasUpdateTimeStamp>446744445544</policy:hasUpdateTimeStamp>
</policy:NegAuthorizationPolicy>
```

Dans l'exemple précédent, la politique définie interdit tous les membres du domaine Arabello de communiquer avec une entité quelconque en dehors du domaine.

Les changements ou les ajouts aux politiques en vigueur, ou un changement de l'état d'un acteur (par exemple, un agent joignant un nouveau domaine ou se déplaçant vers une nouvelle hôte) ou d'une autre entité exigent une déduction logique pour déterminer tout d'abord quelles sont les politiques en conflit et en second lieu comment résoudre ces conflits [100].

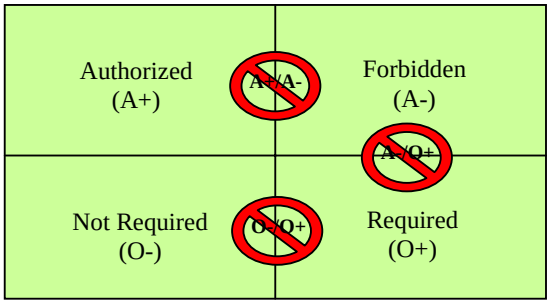

Figure 2.11. Les trois types de conflit

La Figure 2.11 montre les trois types de conflit qui peuvent actuellement être manipulés: l'autorisation positive contre l'autorisation négative (c.à.d. il est simultanément autorisé et interdit d'effectuer une certaine action), l'obligation positive contre l'obligation négative (c.à.d. il est en même temps requis et non requis d'effectuer une certaine action), et l'obligation positive contre l'autorisation négative (c.à.d. il est requis d'effectuer une action interdite). Des algorithmes de résolution de conflit et d'harmonisation de politiques sont développés dans KAoS pour permettre à des conflits de politique d'être détectés et résolus même lorsque les acteurs, les actions, ou les cibles des politiques sont indiqués aux niveaux énormément différents de l'abstraction. Ces algorithmes comptent en partie sur une version de « Standford's Java Theorem Prover » (JTP) [98] qui est intégré à KAoS.

2.7.4. Intégration sémantique des modèles d'information de gestion

Le travail dans [164] a été partiellement financé par le ministère espagnol de la Science et de la technologie sous le projet GESEMAN (Tic2002-00934). Ce travail montre comment l'application des ontologies peut être utile dans un scénario de gestion multi-domaine où différentes technologies sont utilisées pour surveiller et contrôler les ressources. Dans un tel scénario, il est nécessaire d'intégrer plusieurs modèles d'information dans un modèle commun. Cette tâche devient très difficile dans le cas où il faut traiter la sémantique de l'information.

Les techniques d'intégration courantes sont basées sur une méthode syntaxique : Un mappage se fait au niveau des langages et pas au niveau des spécifications. Après translation, l'information reste isolée, ceci est dû au manque au niveau de la sémantique qui décrit la relation permettant la connectivité au reste du modèle.

[164] propose l'utilisation des techniques de fusionnement et de mappage des ontologies pour définir une méthode pour l'intégration des modèles de l'information de gestion en se basant sur une méthode de translation sémantique.

En effet, comme décrit dans la Figure 2.12, une spécification commune de toutes les ressources gérées peut être obtenue en fusionnant différents modèles. D'autre part, il est également nécessaire de transformer le modèle fusionné utilisé par le *manager* vers les modèles d'information définis pour chaque agent. Pour ce faire, un ensemble de règles, permettant la translation sémantique de l'information de gestion, est défini au niveau d'une ontologie de mappage.

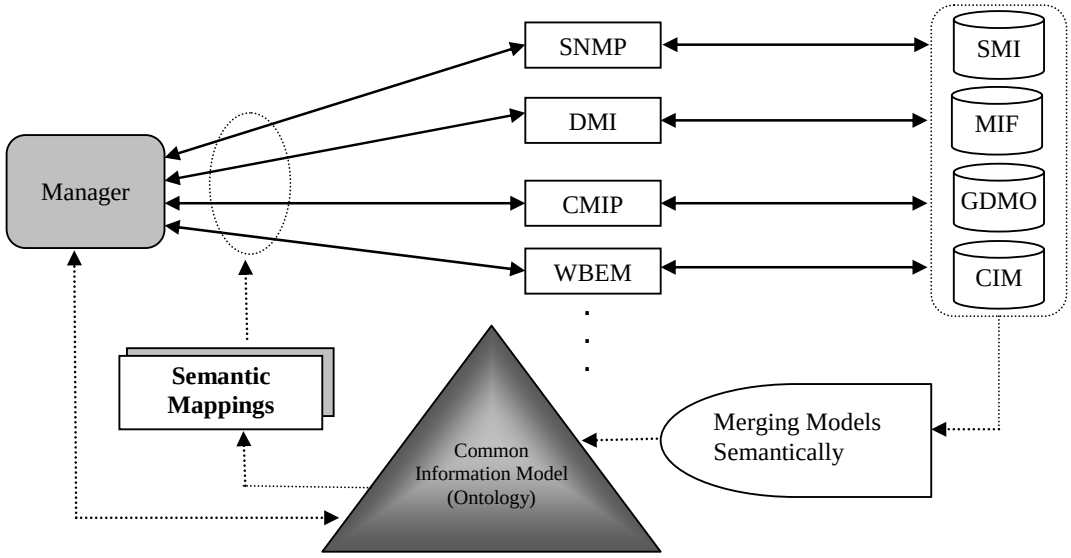

Figure 2.12. L'application d'Ontologie dans la gestion de réseau

Cette nouvelle méthode d'intégration s'appelle M&M (Merge and Map). Elle est représentée dans la Figure 2.13. Elle propose un ensemble d'étapes pour aider dans l'acquisition du modèle commun et des règles de mappage. Elle diffère des méthodes habituelles d'intégration d'ontologie, qui permettent soit le fusionnement, soit le mappage mais pas les deux processus en parallèle.

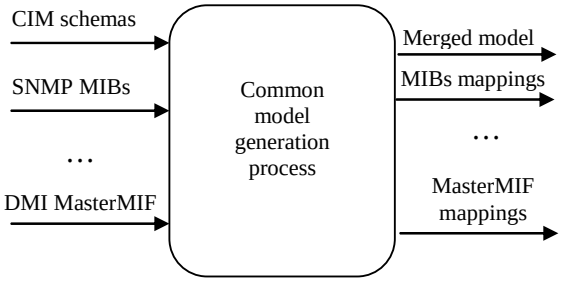

Figure 2.13 - Le processus de fusionnement et de mappage de l'information de gestion

Le fusionnement est basé sur le processus expliqué dans [82] mais adapté pour les particularités de gestion de réseau. D'autres techniques existent, mais ne sont pas applicables à l'information de gestion de réseau, parce qu'elles traitent seulement les classes [83] (les propriétés ne sont pas prises en considération) ou les valeurs des instances [84] (qui ne sont pas déjà connues durant le fusionnement des modèles de la gestion d'information).

Le but de cette méthode est d'assister la personne qui s'occupe de la mise en œuvre de ce processus. Pour ceci l'ensemble suivant des heuristiques est appliqué pour trouver des candidats ayant une grande probabilité d'être fusionnés :

- **Candidats ayant des chaînes de caractères semblables**. Deux classes ou propriétés sont candidates à fusionner avec une probabilité élevée si elles ont une sous chaîne semblable incluse dans leurs identifiants ou dans leurs descriptions. Des synonymes peuvent également être utilisés s'ils sont disponibles.
- **Candidats ayant une hiérarchie d'héritage semblable**. Deux classes sont candidates à fusionner avec une probabilité élevée si leurs classes parentes sont semblables, parce que les classes dérivées d'une classe sont habituellement semblables aux classes dérivées d'une autre classe qui a été fusionnée avec la première.
- **Candidats par domaine de propriété**. Deux propriétés sont candidates à fusionner avec une probabilité élevée si les classes les contenant sont également semblables. En même temps, deux classes sont candidates à fusionner avec une probabilité élevée si les propriétés qu'elles contiennent sont également semblables.

Pour décrire les règles de mappage qui traduisent les instances de l'information d'un modèle concret au modèle commun, une ontologie simple de mappage a été définie en utilisant DAML+OIL [73] comme langage de spécification. La Figure 2.14 montre, à l'aide d'un diagramme de classe, une représentation de cette ontologie. Elle est basée sur des idées contenues dans [86], mais en réduisant sa complexité. Une approche plus simple, comme celle décrite dans [85], n'est pas applicable parce qu'elle traite seulement des classes.

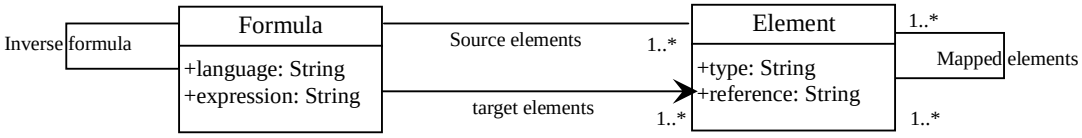

Figure 2.14. Mappage d'Ontologie

Dans cette ontologie, chaque *élément* (classes, propriétés, etc.) admet une *formule* de traduction. Chaque élément a certaines propriétés telles que le *type* ou la *référence* selon ses spécifications. Chaque *formule* a une *expression* écrite dans un langage concret pour traduire l'ensemble d'*éléments* de source et de cible. Des relations entre les éléments mappés et la formule inverse sont également incluses. D'autres approches, comme le qualificateur de MappingStrings utilisé dans CIM, peuvent seulement représenter les mappages directs.

Des mappages typiques parmi les modèles de l'information de gestion ont été identifiés. Ils peuvent être les suivants ou en sont une combinaison:

- Direct, si c'est une relation de 1:1 dans laquelle aucune transformation n'est nécessaire. Dans ce cas, la valeur contenue dans un élément dans les deux modèles est exactement identique. Les mappages directs sont les plus communs et sont proposés par défaut.
- Ensemble de valeur, si c'est une relation de 1:1 dans laquelle pour chaque valeur d'un élément il y a une autre valeur pour l'élément de l'autre modèle. On propose ce genre de mappage si au moins l'une des deux propriétés fusionnées est une énumération de valeurs possibles.
- Types de données, si c'est une relation de 1:1 dans laquelle les éléments des deux modèles ont de différents types de données qui doivent être convertis.
- Opération arithmétique sur un élément, si c'est une relation de 1:1 dans laquelle la valeur d'un élément est obtenue par calcul à partir de la valeur de l'élément de l'autre modèle. Ce genre de mappage est utile si les unités de mesure des deux éléments sont différentes.

- Opérations arithmétiques sur quelques éléments, si c'est une relation 1:n dans laquelle la valeur d'un élément est obtenue par la combinaison arithmétique des valeurs de quelques éléments de l'autre domaine. Dans ce cas, l'utilisateur devrait définir l'expression de mappage.
- Chaînes de caractères, si c'est une relation 1:n dans laquelle la valeur d'un élément se compose en concaténant les différentes chaînes de caractères, qui ne sont autre que les valeurs des éléments dans l'autre domaine. De même, l'utilisateur doit définir l'expression de mappage.

En conclusion, ce travail utilise les techniques d'intégration d'ontologie pour améliorer l'interopérabilité des informations de gestion de réseau. Des travaux antérieurs traitent ce problème d'une façon très limitée, vu qu'ils sont basés surtout sur la translation syntaxique. La méthode M&M prend en considération la sémantique contenue dans l'information. Ceci permet de générer un modèle commun qui peut être utilisé par un administrateur indépendamment des domaines de gestion sous-jacents. Ce fait améliorera le développement d'une application de gestion qui sera capable de corréler les données qui jusqu'à maintenant n'avait pas d'association directe parce qu'elles appartiennent à des modèles différents.

2.7.5. Application du Langage Web d'Ontologie aux définitions des informations de la gestion

Le travail dans [78] a été partiellement financé par le ministère espagnol de la Science et la technologie sous le projet GESEMAN (TIC2002-00934). Ce projet propose l'utilisation d'un langage Web d'ontologie pour définir les informations de gestion. Ce choix permet de combiner les avantages de XML et des ontologies en vue d'améliorer la gestion des réseaux, des systèmes, des applications et des services.

Les avantages principaux de l'utilisation des langages Web d'ontologies découlent du fait que:

- Ces langages Web d'Ontologie sont liés à la sémantique Web et disposent d'un nombre considérable d'utilisateurs et d'outils disponibles.
- Comparés aux langages de représentation des informations de gestion, les langages Web d'ontologie ont été formalisés, de sorte que leur sémantique soit saine et complète, et qu'ils puissent être employés par les systèmes intelligents.
- Quelques ontologies communes peuvent être réutilisées pour la définition d'information, comme ceux qui adressent les unités de mesure. Mais ces ontologies ne sont pas spécifiques pour la gestion et par la suite ne peuvent pas définir toutes les constructions nécessaires communes dans le domaine de la gestion. Elles ne permettent pas également la définition des méthodes ou des opérations, comme la plupart des bases de l'information de gestion.

OWL, le langage Web d'ontologie proposé par le consortium du réseau mondial Web (World Wide Web), peut être directement employé pour spécifier les informations de la gestion, parce qu'il supporte la plupart des constructions définies par les langages de représentation des informations de gestion:

- Les classes peuvent être définies en utilisant la balise owl:Class, et ses dérivés avec rdfs:subClassOf.
- Alors, les attributs sont spécifiés en utilisant owl:DatatypeProperty pour des valeurs littérales ou owl:ObjectProperty comme références aux instances des classes. La restriction peut également être exprimée avec cette construction. Une particularité des langages d'ontologie est que les propriétés n'appartiennent pas directement à une classe. Pour attacher une propriété à une classe, il suffit alors d'inclure la balise rdfs:domain.
- Finalement, d'autres constructions peuvent également être définis: comme par exemple, les associations qui sont des classes avec deux owl:ObjectProperty ou plus, et les événements et les notifications qui sont une spécialisation d'owl:Class.

Le travail réalisé dans [78] consiste à analyser les structures de OWL et à comparer ces structures à celles utilisées par les langages usuels pour la définition de l'information de gestion.

Suite à cette analyse, [78] propose une translation directe des classes définies dans les langages orientés objets telles que CIM et GDMO vers OWL. Pour les langages à base de tables telles que SMI, [78] propose de commencer par une transformation antérieure des tables en classes avant de faire la translation vers OWL.

Les lignes suivantes montrent un exemple de translation vers OWL de la classe CIM_ManagedSystemElement, définie dans CIM Core Model.

```
<owl:Ontology rdf:ID="CIM_Core">
<rdfs:comment>The version 2.8 of CIM Core model</rdfs:comment>
<rdfs:label>CIM Core Model</rdfs:label>
<owl:versionInfo>2.8</owl:versionInfo>
<owl:priorVersion
            rdf:resource="http://www.dmtf.org/standards/cim/cim_schema_v27">
</owl:Ontology>
<owl:Class rdf:ID="CIM_ManagedSystemElement">
<rdfs:comment>
      CIM_ManagedSystemElement is the base class
      for the System Element hierarchy. […]
</rdfs:comment>
<rdfs:subClassOf rdf:resource="#CIM_ManagedElement"/>
</owl:Class>
<owl:DatatypeProperty rdf:ID="InstallDate">
<rdfs:comment>
      A datetime value indicating when the object was installed.
      A lack of a value does not indicate that the object is not installed.
</rdfs:comment>
<rdfs:domain rdf:resource="#CIM_ManagedSystemElement"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Name">
<rdfs:comment>
   The Name property defines the label by which the object is known.
   When subclassed, the Name property can be overridden to be a Key property.
</rdfs:comment>
<rdfs:domain rdf:resource="#CIM_ManagedSystemElement"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Status">
<rdfs:comment>
    A string indicating the current status of the object.
    Various operational and non-operational statuses are defined. […]
</rdfs:comment>
<rdfs:domain rdf:resource="#CIM_ManagedSystemElement"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
```

Cette translation peut être faite pour toutes les classes définies dans CIM core model. Cependant, tous leurs qualificateurs ne peuvent pas être directement exprimés en OWL. Par exemple, des attributs peuvent être spécifiés avec une longueur maximale ou un niveau d'accès qui ne sont pas définis en OWL. Ceci ne pose aucun problème, grâce à l'extensibilité de ce langage.

En effet, les attributs de n'importe quel modèle d'information ont habituellement un ensemble de propriétés appelées les facettes, tel que le type de données ou la description. La plupart des facettes de gestion commune sont incluses en OWL. D'autres peuvent être exprimées comme une extension à ce langage, en les spécifiant dans RDFS pour inclure les propriétés qui n'existent pas dans OWL. De cette façon, l'information définie est d'autant plus complète.

Dans la table ci-dessous, [78] résume ces facettes et montre leurs constructions dans trois langages usuels (SMI, GDMO et CIM) ainsi que leur représentation OWL:

Langage	SMIv2	GDMO	MOF/CIM	OWL
Restriction sur le type de donnée	SYNTAX	WITH ATTRIBUTE SYNTAX	Attached	rdfs:range, owl:allValuesFrom
Documentation	DESCRIPTION	BEHAVIOUR	Description	rdfs:comment
Identifiant unique	::=	REGISTERED	MappingString	rdf:ID

		AS		
Etat de l'Implémentation	STATUS	n/a	Version, Revision	owl:versionInfo, owl:priorVersion, owl:backwardCompatibleWith, owl:incompatibleWith, owl:DeprecatedClass, owl:DeprecatedProperty
Distinction	INDEX, AUGMENTS	WITH ATTRIBUTE	Key, Propagated, Weak	owl:FunctionalProperty
Cardinalité	n/a	n/a	[n], Max, Min	owl:maxCardinality, owl:minCardinality, owl:cardinality
Réference	REFERENCE	n/a	Model Correspondence	rdfs:seeAlso
Redéfinition	n/a	DERIVED FROM	Override	rdfs:subPropertyOf
Accès	MAX-ACCESS	Attached	Read, Write	n/a
Unités	UNITS	n/a	Units	n/a
Valeur par défaut	DEFVAL	DEFAULT VALUE	=	n/a

2.8. CONCLUSION DU CHAPITRE

Nous avons présenté dans les premières sections de ce chapitre l'évolution des réseaux actifs à travers plusieurs projets de recherche. Mais en dépit du grand nombre de travaux qui leur sont consacrés, il faut admettre que les réseaux actifs n'ont pas encore pu démontrer leurs avantages par rapport à d'autres alternatives. Ils n'ont pas non plus connu de grand succès auprès des constructeurs de routeurs comme Cisco ou Alcatel.

On peut de même constater qu'il n'y a pas une architecture commune pour tous les types d'applications. Parmi les architectures des réseaux actifs les plus intéressantes, sont celles qui utilisent le support du middleware dans leur implémentation.

En effet, les structures middleware existantes supportent des mécanismes de communication distribuée, des mécanismes de déploiement et de chargement dynamique de services, ainsi que des mécanismes d'appel de méthodes distantes et une gestion avancée de la sécurité. En utilisant le support middleware, les réseaux actifs peuvent profiter, dans leur conception, de tous ces avantages.

Mais les systèmes et architectures middleware actuels, s'ils sont en voie de standardisation (comme CORBA, COM+ ou EJB), restent néanmoins complexes, adaptés à certains besoin applicatifs, de par leur héritage du modèle RPC, incompatibles les uns avec les autres, et ne sont pas prévus pour être spécialisés pour d'autres domaines d'applications ou architectures. Ceci conduit à une prolifération de solutions ad hoc, adaptés à un domaine d'application particulier, mais difficilement réutilisables ou interopérables avec l'existant. Ceci concerne surtout les supports d'exécution pour la gestion de la qualité de service (comme le temps réel ou la tolérance aux fautes) et plus généralement pour l'adaptation des services à des besoins non prévus au moment du déploiement de l'application.

Les architectures de réseau actif à base de Middleware qu'on a étudié sont soit disponibles pour un système d'exploitation spécifique (COMAN disponible seulement sous Windows), soit elles utilisent un seul langage de programmation (HABA est basée sur un serveur d'application Java), soit encore en phase d'étude (FAIN implémente un prototype sous Linux) et surtout ne sont pas dédiés à la gestion d'un réseau comme l'Internet.

D'autre part, l'évolution et la croissance rapide de l'Internet ont entraîné le développement de nombreux nouveaux services ayant des contraintes de qualité de service différentes. Il devient, dès lors, de plus en plus complexe de contrôler et de gérer le réseau et ses services de manière classique. En effet, il ne s'agit plus de récolter des informations de gestion prédéfinies mais éga-

lement de configurer dynamiquement l'ensemble des dispositifs attachés au réseau pour supporter les besoins de ces nouveaux services. Alors que la gestion à base de politique constitue une solution efficace pour la gestion de réseau, son déploiement est difficile à mettre en œuvre à cause des nombreuses configurations à réaliser et à surveiller dans l'environnement fortement distribué et hétérogène qu'est le réseau d'un opérateur. Les technologies basées sur les réseaux actifs ont émergé comme outil facilitant la gestion des réseaux et le déploiement dynamique de services.

La proposition de remplacer les politiques par les paquets actifs est une idée intéressante. Mais les architectures proposées ne répondent pas aux besoins, en effet :

- SmartPackets est basée sur une méthode classique de gestion.
- La solution qui consiste à combiner l'approche réseau actif (RA) et la gestion de réseau à base de politique (PBM) pour profiter d'une part de la flexibilité et la mise à l'échelle qu'offre la technologie RA pour la gestion distribuée et d'autre part, du potentiel offert par l'approche PBM dans l'automatisation de la distribution de ces tâches de gestion dans le réseau opérateur, représente une avancée dans ce domaine :
  - FAIN a été proposé pour la gestion des réseaux actifs et ne permet pas de faire ni la gestion des réseaux passifs, ni même pas la gestion des nœuds actifs d'autres plateformes.
  - Dans A-PBM, un modèle est conçu pour la gestion multi-domaine. L'inconvénient majeur de ce modèle, qui s'éloigne de celui préconisé par IETF, est le manque d'un standard au niveau de déploiement et au niveau de la représentation de l'information, ce qui pose un problème d'interopérabilité entre cette plateforme et celles existantes.

Le principal inconvénient de toutes ces approches est le manque de dynamicité. En effet, l'objectif de ces approches est d'utiliser la connaissance de la sémantique d'une politique pour spécialiser des services pour l'application de la politique et pour la détection et la résolution de conflit. Une fois ce service généré sous forme d'un code exécutable plus rien ne peut être fait. Il est notamment impossible de reconfigurer le programme pour réagir à un événement ou introduire une nouvelle règle si cela n'avait pas été prévu dans le code généré. Un autre problème se présente alors qui est la capacité d'exécuter un grand nombre de politiques (programmes) à l'intérieur d'un nœud actif.

D'où l'utilité des langages sémantiques dans la représentation des informations de gestion et des politiques. Dans cette approche, les ressources utilisées sont des entités sémantiques catégorisées et décrites de façon componentielle dans une ontologie.

Au sein d'une ontologie, les concepts sont définis les uns par rapport aux autres (modèle en graphe de l'organisation des connaissances), ce qui autorise un raisonnement et une manipulation intelligente de ces connaissances.

Les travaux décrits dans ce chapitre et qui proposent d'utiliser les ontologies dans la représentation de l'information de gestion de réseau sont très récents :

- [165] traite la gestion des politiques pour le déploiement et l'exécution des agents et pas la gestion de réseau à base de politiques. L'objectif dans [165] est de pouvoir définir d'une façon dynamique un objet de politique de sécurité Java.
- L'objectif de [164] est d'unifier la représentation de l'information de gestion et de la définir dans une seule base de données sous forme d'ontologie. Ce qui permet de faciliter la recherche et le traitement de ces informations indépendamment du modèle de gestion utilisé.
- [78] propose la représentation de l'information de gestion en OWL en appliquant une traduction directe. Ceci pose beaucoup de difficulté dans la mise en œuvre de cette traduction surtout que les propriétés n'appartiennent pas directement à une classe. Ce qui manque c'est la définition d'un modèle conceptuel qui permet de catégoriser et d'interpréter ces informations ainsi que la définition des règles de politiques.

Cet état de l'art n'est surtout pas exhaustif, mais il a essayé de présenter les principales approches et applications dans trois domaines principaux qui sont :

- Les réseaux actifs et surtout les architectures à base de middleware.

- La gestion des réseaux à base de politique.
- La définition d'un modèle sémantique pour l'information de gestion.

Ces trois domaines représentent les trois piliers du contexte dans lequel a évolué notre travail. Un travail qui a abouti à la proposition d'une architecture et l'implémentation d'une plateforme mettant en œuvre un service de gestion dynamique, contrôlable, intelligent et interopérable.

Chapitre 3

PLATEFORME DE RESEAU ACTIF A BASE DE WEB SERVICES

3.1. INTRODUCTION

Les avantages des réseaux actifs sont bien identifiés [107]. Ils fournissent un déploiement dynamique des services, une distribution de la charge du réseau, une connectivité entre les équipements et une personnalisation du réseau suivant les applications utilisées. Le but principal des réseaux actifs est de rendre les applications encore plus performantes par notamment la distribution de leur traitement.

Nous avons vu dans le Chapitre 2 que les architectures des réseaux actifs sont groupées selon leurs différentes approches [14]. L'approche paquets actifs, qui se caractérise par le transport du code actif au niveau des paquets. Les Smart Packets [12], le projet Active IP [109], et l'architecture M0 [108], en sont des exemples. Dans l'approche nœud actif, les paquets ne transportent pas le code, mais une référence à des fonctions prédéfinies qui résident dans les nœuds actifs. ANTS [6], DAN [5] ainsi que HABA [35] en sont des exemples. Dans ANTS [6], DAN [5] et HABA [35] a été adopté un schéma de déploiement de code à la demande, avec la possibilité de mettre en cache ce code, pour des utilisations ultérieures.

Dans ce chapitre, nous proposons une nouvelle architecture de communication support pour les réseaux actifs ainsi que les services supportés par cette dernière. Ce travail a été réalisé dans le cadre du projet Amarillo qui s'est basé sur les travaux et résultats du projet Amarrage, et qui en est en fait une continuité. L'aspect et résultat qui reste toujours fondamental est celui de l'architecture à trois plans : un plan de contrôle des réseaux actifs, un plan réseau actif et enfin un plan de transport. Les travaux de recherche et les publications associés [168] exposent largement ces acquis.

Cette nouvelle architecture active, permettant la création et le déploiement de services réseaux, a été baptisée ASWA, Architecture à base de Services Web Actifs. C'est une architecture à base de composantes logicielles réutilisables: les Web services [25]. Elle s'appuie sur l'infrastructure de communication distribuée offerte par le Web, pour un déploiement dynamique et contrôlé du code mobile.

ASWA offre une conjonction améliorée des caractéristiques de déploiement spécifiques à ANTS [6], DAN [5] et HABA [35], tout en proposant une réponse encourageante à un réseau actif sécurisé et interopérable dans un environnement hétérogène.

Pour résumer, cette contribution pourrait s'identifier principalement sur quatre plans :

- Sur le plan architectural, ASWA définit le nœud actif en conformité avec le « DARPA Architectural framework for Active Networks » [3], où les fonctionnalités d'un nœud actif se répartissent en trois composantes majeures: le système d'exploitation du nœud, l'environnement d'exécution, et les Applications Actives.
- Sur le plan de l'implémentation, ASWA comme son antécédent HABA [35], est une plateforme à base de composantes. Cette approche offre plus de flexibilité et de modularité au niveau de la manipulation et de la maintenance du code, ainsi qu'elle garantit l'extensibilité par

ajout et déploiement dynamique de nouvelles composantes en vue de modifier dynamiquement les fonctionnalités de n'importe quel nœud du réseau. Ainsi l'interopérabilité avec ANTS peut être effectuée par simple composition. ASWA s'appuie sur l'architecture des Web Services [25] pour atteindre ces objectifs. En effet, un Web Service [25] fournit une infrastructure basée sur le protocole SOAP [26] pour la communication entre les composantes distribuées. SOAP est un protocole indépendant du langage de programmation, du système d'exploitation et du matériel. En fait, il peut être utilisé pour assurer la communication entre n'importe quels éléments supportant TCP/IP via HTTP, SMTP, FTP ou autres (voir Figure 3.1). Comme la quasi-totalité des langages informatiques supporte ces protocoles, un Web service développé sous une plate-forme Microsoft .NET [110] en C# pourra être utilisé par les langages : Perl, PHP, Python, Delphi, Cobol, ou autres. L'utilisation de ASWA garantit l'interopérabilité entre langages, systèmes et réseaux hétérogènes.

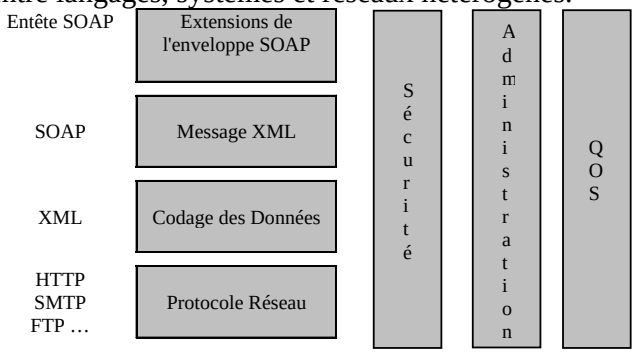

Figure 3.1 - Pile réseau

- Au niveau du déploiement de code, ASWA utilise l'approche du nœud actif pour mieux contrôler le chargement de code, vu que cette approche offre des mécanismes de chargement et d'exécution de code séparés.
- Sur le plan de la sécurité, ASWA utilise le protocole SOAP [26] pour insérer des mécanismes de sécurité tel que l'authentification par secrets pré partagés ou par certificats. ASWA utilise également le protocole HTTPS (HTTP over SSL) pour sécuriser le déploiement. Mais comme on l'a souligné, le protocole SOAP [26] consiste à faire circuler du XML [23] basé sur le protocole HTTP, ce qui lui permet de franchir les pare-feu. ASWA définit un "schéma XML" [24] dont l'instanciation permet de définir la liste de contrôle d'accès d'un pare-feu personnalisé sous forme d'un fichier XML qui sera déployé sur chaque nœud actif. Ce fichier XML permettra alors de surveiller le déploiement des services actifs de nœud en nœud ainsi qu'il permettra de contrôler le traitement des paquets actifs.

Mais, pourquoi encore une telle architecture ?

1. Cette architecture se situe au niveau applicatif et nous détache des contraintes liées aux équipements. Dès qu'on adresse les réseaux actifs, on avance les arguments liés à leur infaisabilité. Et ceci est dû essentiellement aux équipements fermés.
2. Nous intégrons dans ce projet la problématique de déploiement de services actifs, les réseaux actifs sont bien entendu nécessaires mais deviennent transparent.
3. Le choix des « Web services » comme infrastructure pour cette architecture cadre parfaitement avec les choix actuels au niveau de l'industrie. Ceci contribue à une exploitation unifiée des services.

Dans la section 2 nous situons ASWA dans son contexte vis-à-vis de quelques plateformes de réseaux actifs à base de composantes. L'apport et l'intérêt d'utiliser les Web services dans l'implémentation des réseaux actifs sont discutés dans la section 3. Dans la section 4 nous détaillons les concepts de ASWA et dans la section 5 son architecture. La section 6 traite la sécurité dans ASWA. L'installation anticipée du code dans ASWA est introduite dans la section 7. Pour valider ASWA, l'implémentation d'une application active, qui joue le rôle d'un pare-feu distribué, est détaillée dans la section 8. Enfin, nous concluons ce chapitre dans la section 9.

3.2. POSITIONEMENT DES PLATEFORMES ACTIFS

Actuellement, il n'existe pas de plateformes de réseaux actifs à base de Web Services. Les architectures décrites dans le Chapitre 2 et qui proposent d'utiliser les services du middleware ou les environnements à base de composants dans la mise en œuvre des plateformes du réseau actif sont très récentes. Ces architectures sont soit disponibles pour un système d'exploitation ou pour un langage de programmation spécifique, soit encore en phase d'étude.

La Table 3.1 suivante présente un récapitulatif de différentes plateformes actives à base des composantes.

	Déploiement	**Sécurité**	**Interopérabilité**	**Portabilité**
COMAN	Installation Manuelle de composantes DCOM	Gérée par le Système		Windows seulement
Chameleon	A partir d'une station d'administration en utilisant les paquets actifs	Gérée par le NodeOS qui est PromethOS		Linux seulement
HABA	Fichiers Jar, à partir du nœud précédent, ou parallèle à partir d'un Serveur de code	Protection des Nœuds Actifs par Déploiement de Certificat	✓	Toutes les plateformes implémentant un serveur d'applications Java
ASWA	Installation des Fichiers Services Web sur un chemin spécifié	Protection en utilisant les entêtes des messages SOAP	✓	Toutes les plateformes implémentant les Web services

Table 3.1 - Les plateformes actives

3.3. APPORT DES WEB SERVICES POUR LES RESEAUX ACTIFS

Une relation étroite existe entre les services middleware et les réseaux actifs. Les deux environnements ont pour objectif la distribution du traitement. Les middlewares couvrent généralement un réseau d'entreprise ou restreint, alors que les réseaux actifs peuvent couvrir un réseau à grande échelle. Malgré cette relation étroite, très peu de plateformes de réseau actif essaient de profiter, dans leur conception, de ces avantages des structures middleware existantes qui supportent des mécanismes de communication distribuée, des mécanismes de déploiement et de chargement dynamique de services, ainsi que des mécanismes d'appel de méthodes distantes et une gestion avancée de la sécurité.

Avec le protocole SOAP, les applications et les architectures à base de Web services développées avec n'importe quel langage de programmation et basé sur tout type de système d'exploitation, pourront communiquer sans modification. Il suffit pour assurer l'interopérabilité entre les architectures de réseaux actifs à base de Web services et d'autres architectures de réseaux actifs, d'encapsuler les paquets de ces derniers dans des messages SOAP et de confier leur transport à une couche de communication fondée sur HTTP.

Ceci n'est pas aussi facile pour les plateformes actives actuelles à base de composants middleware qui nécessitent pour leur communication des modifications liées au fait que les middlewares existants sont limités dans leurs capacités, difficiles à utiliser et incompatibles les uns avec les autres.

Il existe d'autres avantages liés à l'utilisation des Web services dans l'implémentation des réseaux actifs:

1. Au niveau de l'implémentation: La définition des routines de traitement de paquets comme étant des Web services peut simplifier leur mise en œuvre. Les applications actives (AA) peuvent utiliser ces services selon leurs besoins et à travers les conventions standards des

Web services, elles les découvrent grâce à UDDI [28] qui est l'annuaire des Web services, elles peuvent négocier leur utilisation dynamiquement, elles peuvent les choisir et les exécuter en temps réel.

2. Au niveau de la sécurité: les protocoles SSL et IPSec sont usuels pour offrir les services de confidentialité, d'intégrité et d'authentification des parties. Les spécifications des Web services permettent d'améliorer la sécurité granulaire des systèmes basés sur le protocole SOAP. Elles définissent l'aptitude à échanger des justificatifs d'identification (par exemple, les certificats X.509 ou les tickets Kerberos), de vérifier l'intégrité d'un message et de mettre en vigueur la confidentialité de ce message. D'autre part, un Web service s'exécute comme étant un processus isolé ce qui permet de protéger le réseau et d'éviter l'effondrement de son architecture active si un service s'arrête brusquement. De plus chaque Web service est un processus client qui fait accès aux données définies uniquement dans son propre espace d'adressage ce qui permet d'assurer la protection de la mémoire.
3. Au niveau de l'extensibilité des réseaux actifs : elle est assurée par les Web services. A partir d'un Web service, on peut construire des services ou des applications Web dérivés des précédents. En effet, le point crucial des Web services est l'imperméabilité entre les clients et les serveurs qui ignorent tout de l'implémentation, dite privée, des opérations respectives de leurs correspondants et ne se connaissent qu'à travers des descriptions publiques.

3.4. ASWA : LES CONCEPTS

Un nœud actif ASWA est organisé en conformité avec la spécification DARPA [3]. Il définit 3 composantes majeures : une composante système d'exploitation (NodeOS), une composante environnement d'exécution (EE), et une composante applications actives (AA). Ces composantes interagissent ensemble pour assurer le déploiement et l'exécution des Applications Actives. La Table 3.2 suivante décrit la pile des éléments constituant le nœud ASWA ainsi que leur correspondance avec la spécification DARPA [3].

Noeud Actif [3]	**ASWA**
AA	Web Services applicatifs dynamiques
EE	Web Services fonctionnels prédéfinis
NodeOS	Serveur d'Application et Serveur Web

Table 3.2 - Eléments d'un noeud ASWA

Pour garder une compatibilité avec les réseaux classiques, les nœuds actifs seront déployés seulement sur quelques points spécifiques du réseau. Ainsi, nous aurons à gérer un réseau formé de deux plans principaux: un plan actif, et le plan transport [36].

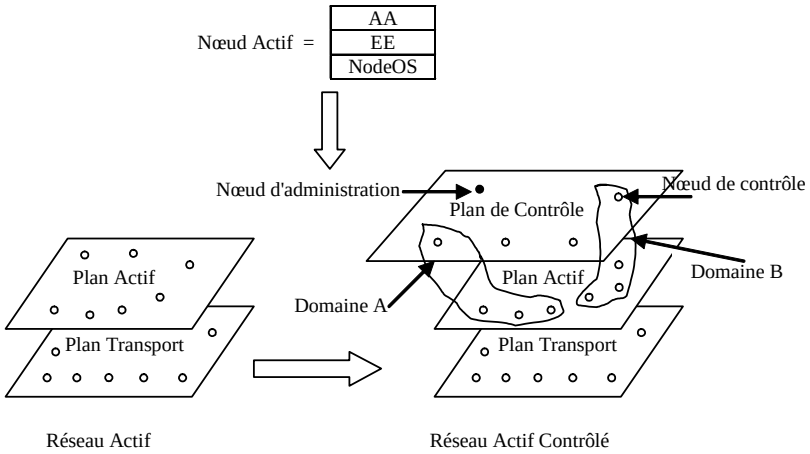

Figure 3.2 - Architecture en 3 plans proposée par le projet RNRT Amarrage

Avec cette approche, nous proposons de contrôler et de sécuriser le déploiement de code en

définissant un troisième plan de contrôle actif. Cette architecture en 3 plans [111], décrite à la Figure 3.2 réalise ces buts par l'introduction de deux nouveaux types de nœuds, les nœuds de contrôle et le nœud d'administration. Cette approche est celle qui a été validée dans le projet RNRT Amarrage.

Comme illustré dans la Figure 3.3, en regroupant les nœuds suivant leurs fonctionnalités nous obtenons :

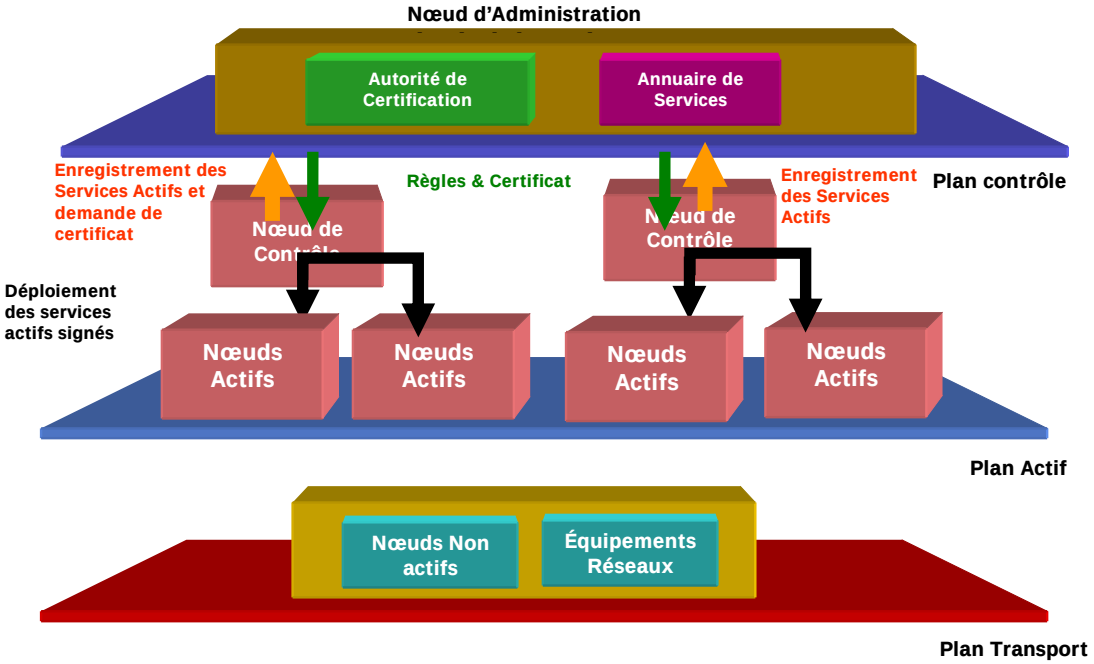

Figure 3.3. Architecture de ASWA

– Un plan de contrôle actif : Ce plan regroupe les nœuds de contrôle et un ou plusieurs nœuds d'administration. Le nœud d'administration est responsable de la configuration et de l'administration des nœuds de contrôle, ainsi que de l'enregistrement de nouveaux services. Tandis que les nœuds de contrôle sont responsables du contrôle et du déploiement de code. Le nœud d'administration agit de même en tant qu'autorité de certification, et distribue les certificats aux nœuds de contrôle avec lesquels il a une relation de confiance, pour que ces nœuds puissent signer le code avant de le déployer. Ces nœuds communiquent ensemble pour avoir une vue globale du réseau et des services supportés.
– Un plan actif : Ce plan regroupe les nœuds actifs. Le NodeOS de ces nœuds offre les primitives de base permettant l'accès aux ressources locales du nœud. L'EE permet le traitement des données transmises par accès aux champs données et adresses des capsules, ainsi qu'en consultant les tables de routage ou en appliquant les fonctions de routage pour router les capsules suivant leur destination. Ces nœuds communiquent avec les nœuds de contrôle.
– Le plan transport : Ce plan regroupe les nœuds qui transportent les capsules sans aucun traitement.

La notion de domaine fut introduite au sein de cette architecture, pour réaliser une meilleure structuration [111]. Chaque domaine regroupe un nœud de contrôle et plusieurs nœuds actifs. Chaque nœud de contrôle est autonome dans son domaine; Il gère et contrôle les nœuds actifs de son domaine. Chaque nœud de contrôle met à jour une table de routage permettant d'acheminer un flot de capsule d'un service donné dans un domaine ou vers un autre domaine suivant le meilleur chemin associé à ce service. De plus, un nœud de contrôle d'un domaine A peut demander au nœud d'administration un code de service non présent au niveau de son cache. Le nœud d'administration routera alors la requête vers le nœud de contrôle le plus proche d'un autre domaine en possession de ce code.

3.5. ARCHITECTURE DE ASWA

Les Web services [25] reposent sur une architecture orientée services ou SOA (section C.4.5) où trois entités principales rentrent en jeu, comme le montre la Figure 3.4:

- des consommateurs de services,
- des fournisseurs de services,
- des références de services.

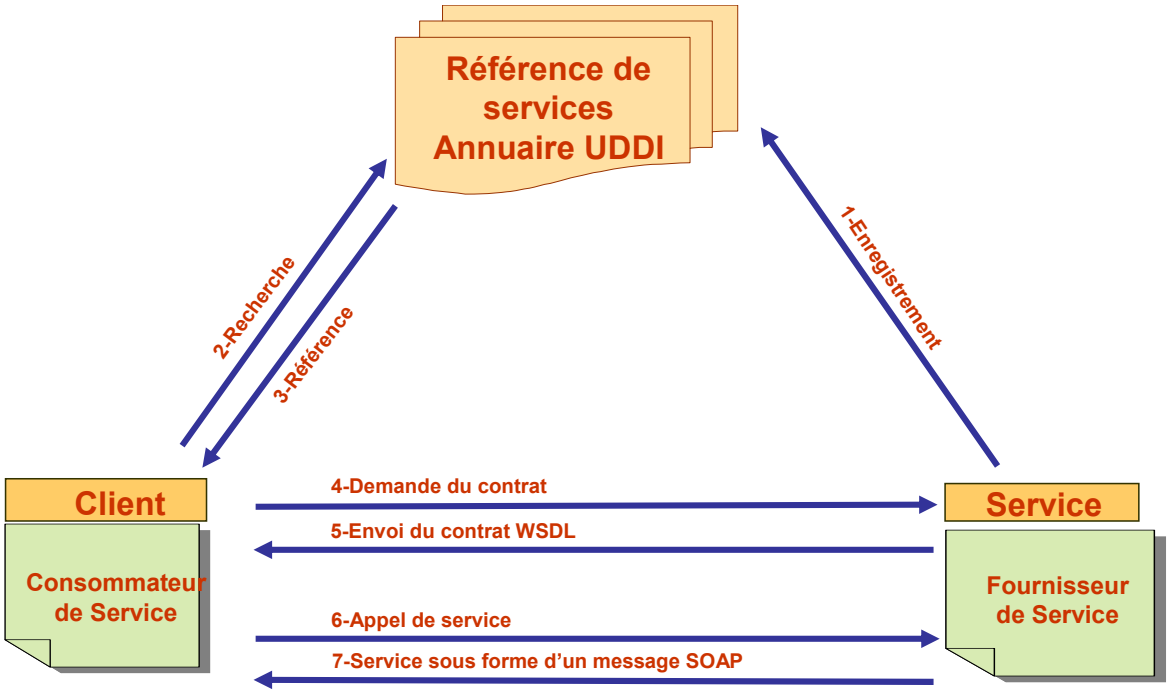

Figure 3.4 - Architecture SOA

Cette architecture peut parfaitement s'adapter à notre proposition de contrôle du réseau actif. En effet, le serveur d'annuaire UDDI permet d'enregistrer les nouveaux services, il peut donc jouer le rôle d'un nœud d'administration. Un nœud de contrôle ne serait alors qu'un fournisseur de services, tandis qu'un nœud actif serait le consommateur de ces services.

Pour implémenter cette architecture notre choix s'est fixé sur la plate-forme .NET de Microsoft qui est assez complète en terme de gestion de la sécurité (authentification et autorisation réalisée simplement par traitement de l'enveloppe SOAP), bien qu'il soit possible d'utiliser indifféremment des API sous Solaris, Linux, ou autres.

3.5.1. Format du paquet dans ASWA

Le format du paquet actif de ASWA est illustré dans la Figure 3.5:

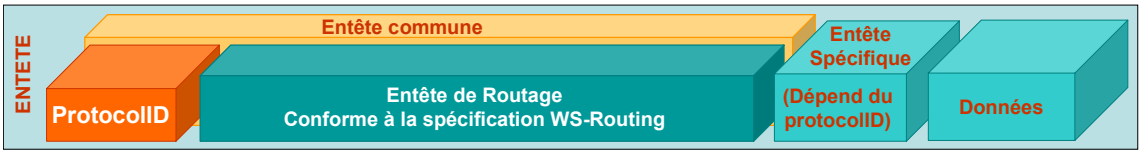

Figure 3.5 - Format d'un paquet

Le paquet ASWA est composé d'un entête et d'un corps. L'entête est structuré en:

- un entête commun qui contient:
  a. le protocolID représentant le type de l'application active associée à ce paquet et
  b. un entête de Routage conforme au protocole WS-Routing [30].
- un entête spécifique à chaque Application Active. L'utilité de cet entête sera vue en détail dans le cas de l'application du pare-feu actif.

Les entêtes du paquet ASWA seront encapsulés dans l'entête d'un message SOAP et les

données dans le corps. Le tout sera transporté sur le réseau par le protocole HTTP, ceci est illustré dans la Figure 3.6.

```
POST /iroute/iroute.asmx HTTP/1.1
Host: aswa1
Content-Type: text/xml; charset=utf-8
Content-Length: 10
SOAPACTION: http://tempuri.org/Fwd
<?xml version="1.0" encoding="utf-8"?>
<!-- Enveloppe  SOAP définie en XML -->
<soap: Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
<!-- Entête du paquet actif -->
<!--Entête commune -->
        <ProtocolID>ProtocolIDHere</ProtocolID>
        <!--Entête de Routage  -->
<!--Entête spécifique à chaque AA -->
<!-- Gestion de sécurité et autres … -->
  </soap:Header>
<soap:Body>
<!-- Données du paquet actif -->
<data>dataHere</data>
</soap:Body>
</soap:Envelope>
```

Figure 3.6 - Format XML du paquet actif encapsulé dans un message SOAP transporté via HTTP

3.5.2. Nœud d'administration

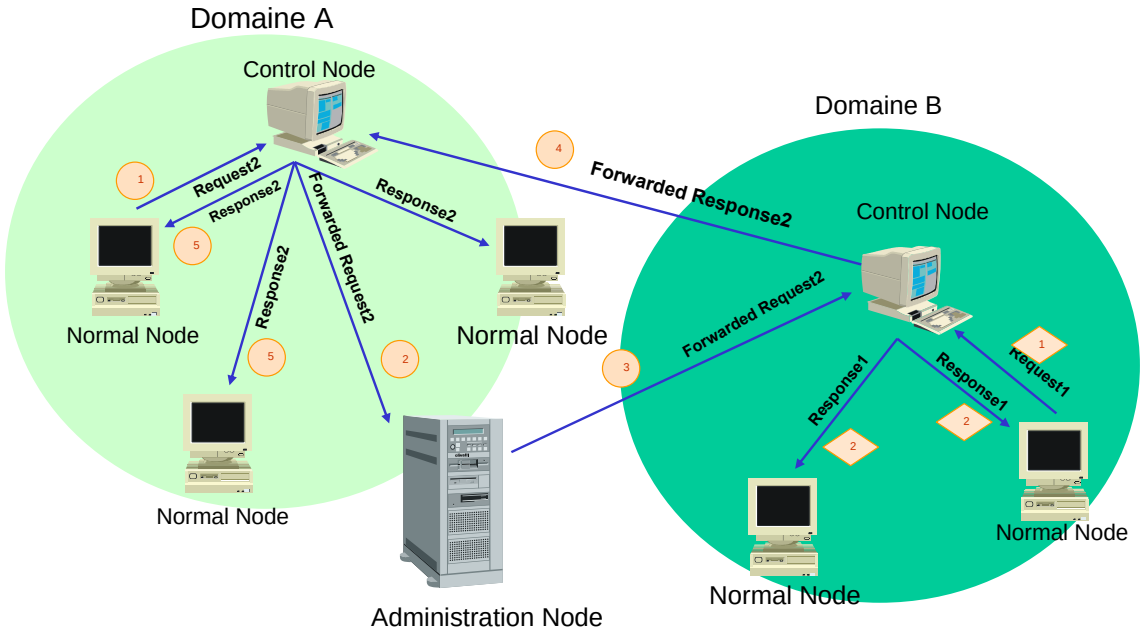

Figure 3.7 - Déploiement inter domaine du code actif

Le nœud d'administration n'est autre qu'un nœud actif qui accepte l'enregistrement de tous leurs services supportés par les nœuds de contrôle. Ceci est réalisé automatiquement à l'aide d'un service de registre qui fournit des renseignements concernant les différents Web Services disponibles, tels que le nom de la société qui héberge le service, l'URL (Universal Resource Locator) du site Web de cette société ainsi que l'emplacement du fichier WSDL [27] grâce auquel il pourra déterminer les détails fonctionnels de ce Web service XML. Ce nœud fonctionne de même en

tant qu'autorité de certification, dont la clé publique doit être distribuée par face à face à tous les nœuds actifs du réseau voulant recevoir du code authentifié. Ce nœud serait donc d'une part responsable de la réception des requêtes d'enregistrement provenant des nœuds de contrôle selon la Figure 3.4, et joue d'autre part le rôle d'un relais de requêtes de demande de code entre les différents nœuds de contrôle dans des domaines différents: Quand le nœud d'administration reçoit une requête de demande de code d'un nœud de contrôle, il détermine l'adresse du nœud de contrôle ayant le code du service requis, et route la demande vers ce dernier (Figure 3.7).

3.5.3. Nœud de contrôle

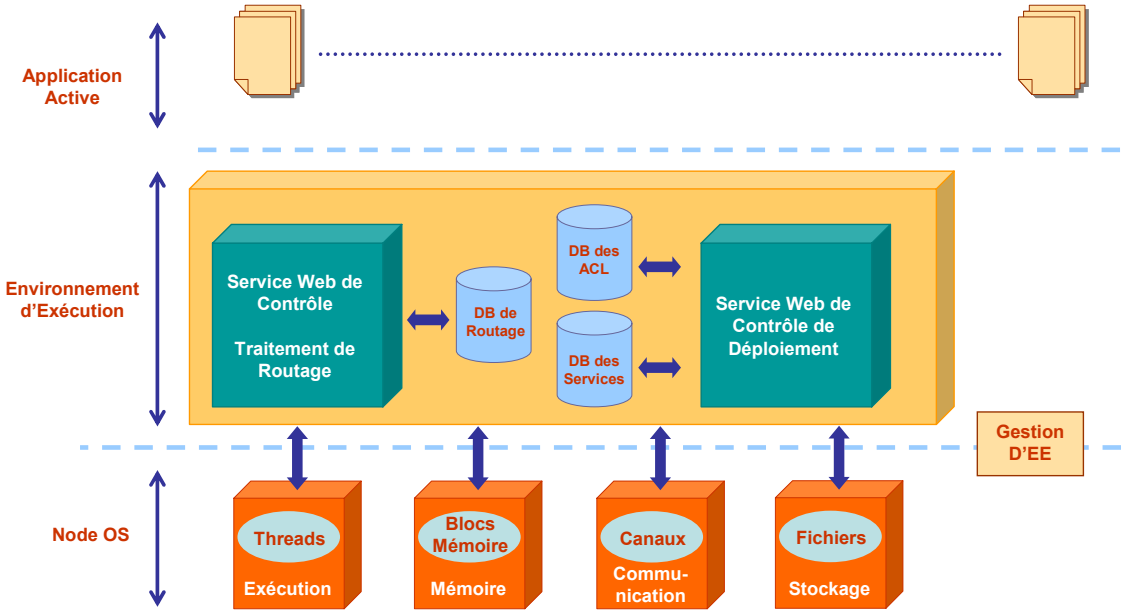

Figure 3.8 - Architecture d'un noeud actif de contrôle

Le nœud de contrôle est introduit au niveau de cette architecture pour contrôler la distribution du code dans le réseau. Un nœud de contrôle supporte au niveau de son EE, telle que le montre la Figure 3.8, un Web service de contrôle de déploiement ainsi qu'un service de traitement de routage. Le service de contrôle de déploiement s'occupe du déploiement du code des Web services actifs ainsi que du déploiement de fichiers de configuration, en particulier les ACLs (Access Control Lists). Tandis que le service de contrôle de routage est chargé de gérer tous les aspects de routage des requêtes. C'est un Web service dont le rôle est de sauvegarder la topologie de tous les réseaux du domaine auquel il appartient et de répondre aux requêtes des nœuds actifs dans leur recherche d'un « nœud suivant ».

Le nœud de contrôle doit pouvoir gérer plusieurs situations :

- déclarer tous les services actifs qu'il supporte selon le schéma de la Figure 3.4.
- répondre aux requêtes de demande du code de services actifs provenant d'un nœud d'administration en envoyant le code correspondant au nœud de contrôle d'un autre domaine qui en a besoin.
- répondre seulement aux requêtes authentifiées et autorisées provenant des nœuds actifs de son domaine.

Le noeud d'administration n'est autre qu'un nœud de contrôle particulier, pouvant supporter les mêmes composantes au niveau de son EE. Dans ce cas, le service de traitement de routage sera chargé de gérer et de sauvegarder la topologie de tous les domaines du réseau. Ainsi le nœud d'administration pourra fonctionner en tant que relais d'aiguillage de requêtes de demande de code, ou de demande d'exécution inter domaines de services, grâce à son service de contrôle de déploiement.

3.5.4. Nœud Actif

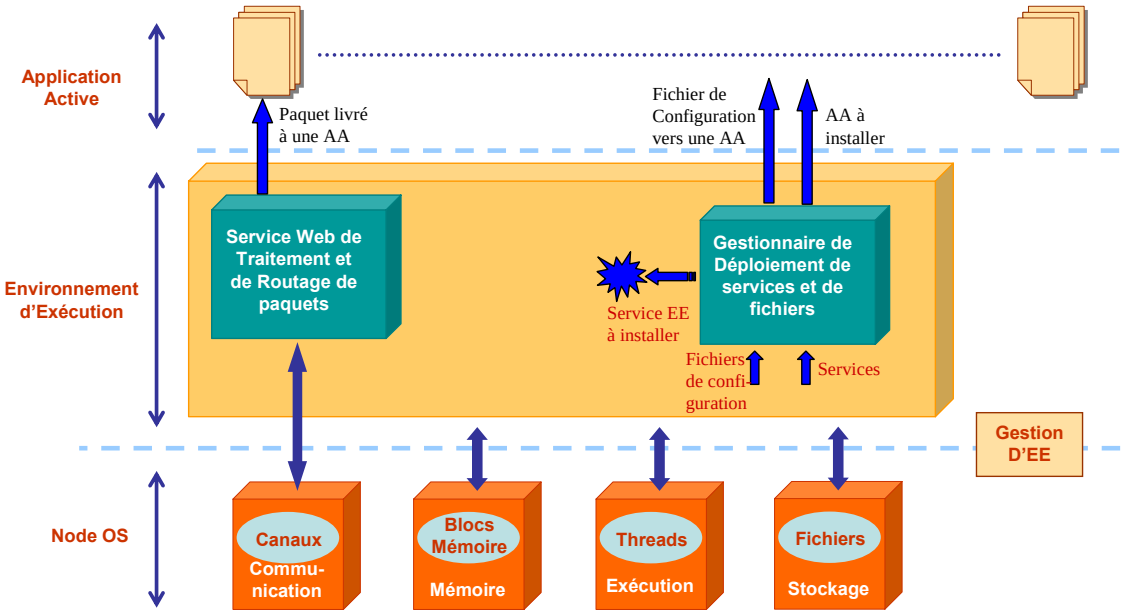

Figure 3.9 - Architecture d'un noeud actif

Un nœud actif n'est autre qu'un nœud supportant un ou plusieurs Web services actifs. L'environnement d'exécution d'un nœud actif supporte les trois Web services suivants (Figure 3.9):

- un Web service de traitement de paquets dont le rôle est de recevoir les paquets, et de leur faire subir un certain traitement, qui peut être soit de les livrer à l'application active concernée soit simplement de les livrer au service de routage.
- un service de routage de paquets dont le rôle est de router les paquets sur un chemin bien déterminé d'après la destination. Le routage de paquets s'effectue après les avoir reçus à partir de la composante de traitement de paquets. Ce routage peut se faire selon plusieurs modes décrits au niveau de la section 3.5.5.
- un Web service d'installation de code ou de fichiers qui permet au nœud actif la réception et l'installation dynamique de services actifs sous forme d'Applications Actives (AA) ou de nouvelles composantes pour l'Environnement d'Exécution (EE), ou même l'installation locale de fichiers de configuration nécessaires au fonctionnement d'une application active.

3.5.5. Fonctionnement du service de routage dynamique d'un nœud actif

ASWA se base sur la spécification « WS-Routing » [30] qui utilise l'entête du message SOAP pour définir les paramètres du routage. WS-Routing [30] est conçu pour fournir les informations nécessaires pour corréler les échanges de messages entre les nœuds de traitement SOAP. Cette spécification permet une indépendance de la plate-forme et du protocole utilisé, un allégement de notre architecture du réseau actif à base de Web services, une séparation entre les données utiles et les données de contrôle ou de traitement et une facilité de franchissement de barrières spécifiques de communications (pare-feu, protocoles spécifiques de transport, etc.).

3.5.5.1. Entête de routage du paquet actif

Le format du paquet avec entête de routage est illustré dans la Figure 3.10 suivante :

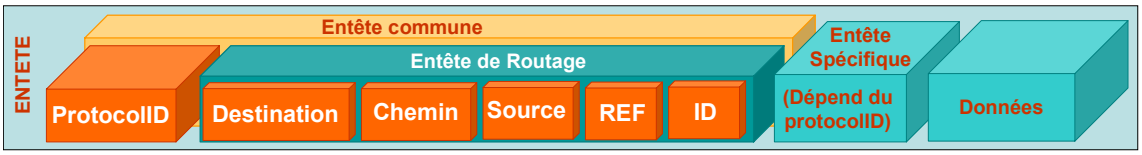

Figure 3.10 - Format d'un paquet avec entête de routage

- l'adresse source est l'adresse du client demandant un service donné
- l'adresse destination est l'adresse du destinataire qui doit éventuellement retourner un résultat au nœud source.

- le chemin est constitué de deux sous-champs optionnels. Le premier définit l'ensemble des nœuds intermédiaires à traverser par ce paquet et le second définit l'ensemble des nœuds intermédiaires à traverser par le paquet réponse.
- le champ « ID » contient l'identité unique de chaque paquet en circulation pour assurer la corrélation de ces derniers.
- Le champ « Ref » est facultatif et n'est autre que l'élément relatesTo qui indique une relation avec un autre paquet.

L'exemple dans la Figure 3.11 suivante montre le format XML du message SOAP conforme à la spécification WS-Routing [30]:

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
      <!-- Entête du paquet actif -->
      <!--Entête commune -->
            <ProtocolID>ProtocolIDHere</ProtocolID>
            <!--Entête de Routage  -->
                  <!—Méthode de Routage qui peut prendre une des valeurs suivates :
                        Integrated, Progressive ou Sequential -->
                  <RoutingMethod>RoutingMethodHere</RoutingMethod>
                  <path xmlns:m="http://schemas.xmlsoap.org/rp/">
                        <action> http://schemas.xmlsoap.org/soap/fault</action>
                        <!—Adresse destination -->
                        <to> http://localhost/Recepteur/Recepteur.asmx</to>
                              <!— Chemin -->
                        <fwd>
                              <via> http://localhost/RouteurA.ashx</via>
                              <via> http://localhost/RouteurD.ashx</via>
                        </fwd>
                        <!—Adresse source -->
                        <from> http://localhost/ClientApp.aspx</from>
                        <!— Champ ID -->
                        <id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</id>
                        <!—Champ REF -->
                        <relatesTo>uuid:20020802-1403-4351-b623-5dsf35sgs5d6</relatesTo>
                  </path>
      <!--Entête spécifique à chaque AA -->
      <!-- Gestion de sécurité et autres … -->
  </soap:Header>
  <soap:Body> ... </soap:Body>
</soap:Envelope>
```

Figure 3.11 - Format XML d'un message SOAP conforme à la spécification WS-Routing

En cas d'erreur, un élément «fault» est ajouté à l'entête. Il contient le code de l'erreur, sa description et le nœud qui l'a générée.

3.5.5.2. Modes de routage

Le service de routage dynamique disponible au niveau du EE d'un nœud actif peut fonctionner suivant trois modes:

- Le mode «ROUTAGE INTÉGRÉ» où les requêtes sont routées d'un nœud actif au suivant en se basant sur le contenu du champ «chemin» du paquet actif.
- Le mode «ROUTAGE PROGRESSIF» où chaque nœud actif envoie une requête au nœud de contrôle pour connaître le nœud actif suivant vers lequel il faut router le paquet.
- Le mode «ROUTAGE SEQUENTIEL» où chaque nœud actif route le paquet suivant sa table de routage définie selon la spécification WS-Referral [112].

C'est l'application cliente initiatrice de l'exécution de l'application active qui doit :

- fixer le mode de fonctionnement du routage des paquets,
- préciser l'application active à exécuter sur chaque noeud,
- préciser le nœud de départ et le nœud destination.

Routage dans le mode « INTÉGRÉ». Dans ce mode, le nœud actif de départ qui reçoit un paquet, demande au nœud de contrôle le chemin complet lui permettant d'atteindre sa destination (Figure 3.12). Un nouveau paquet sera construit qui contiendra le chemin complet.

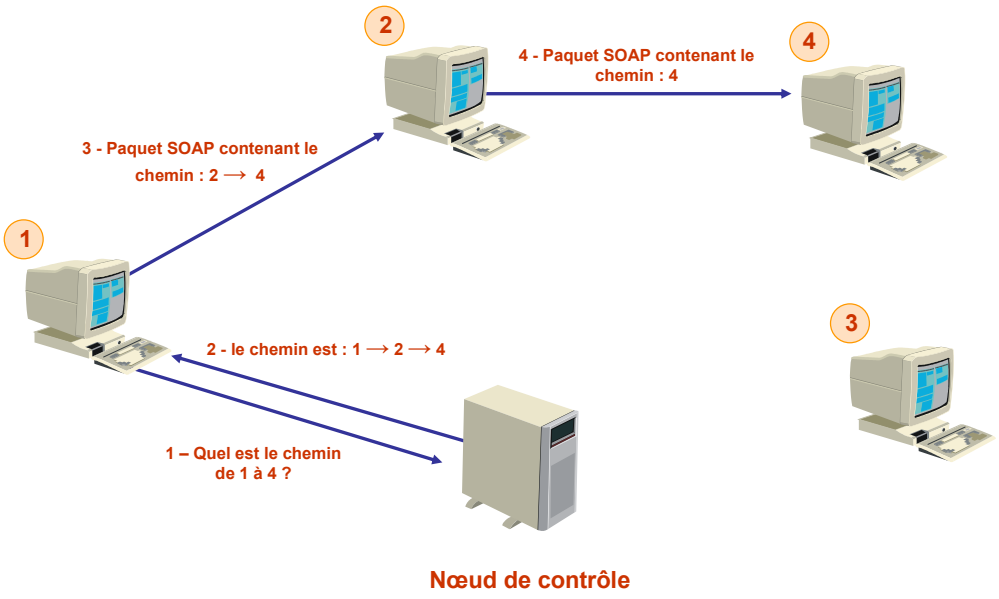

Figure 3.12 - Routage intégré

Routage dans le mode « PROGRESSIF ». Dans ce mode, à la réception d'un paquet, chaque nœud actif envoie au nœud de contrôle une requête de demande du nœud suivant sur le chemin à suivre (Figure 3.13). Dans ce cas, le champ « chemin » du paquet construit au niveau de chaque nœud actif contiendra une seule adresse qui est celle du nœud suivant.

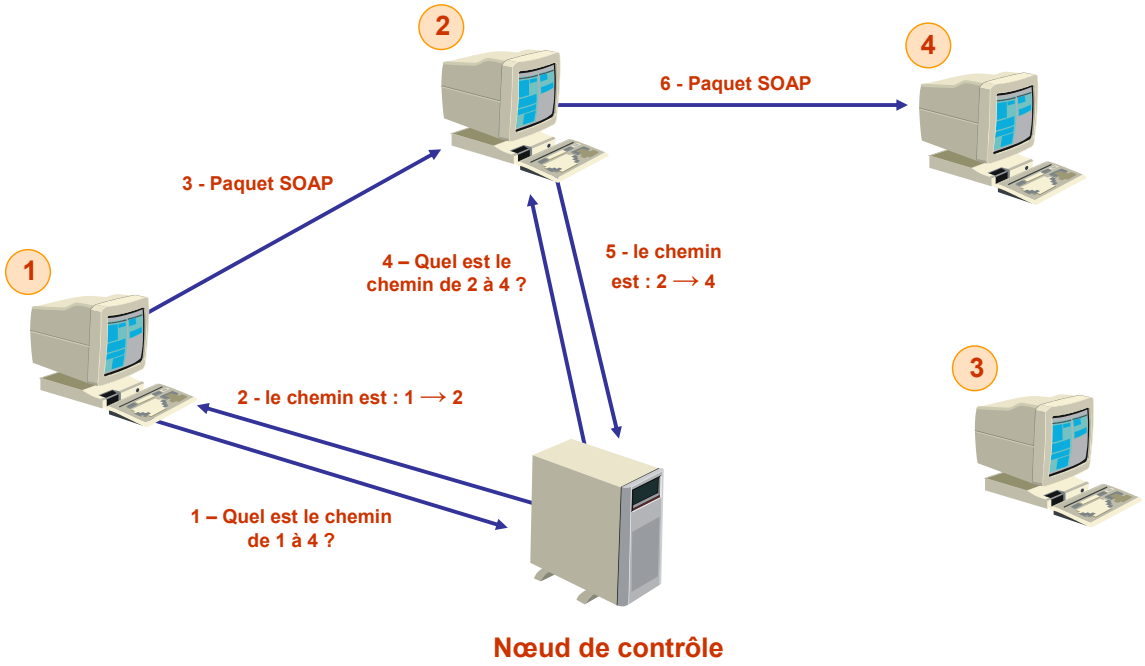

Figure 3.13 - Routage progressif

Routage dans le mode «SEQUENTIEL ». Dans ce mode, à la réception d'un paquet, chaque nœud actif cherche dans sa table de routage local le nœud suivant vers lequel il envoie le pa-

quet. Cette table de routage est définie selon les spécifications WS-Referal [112].

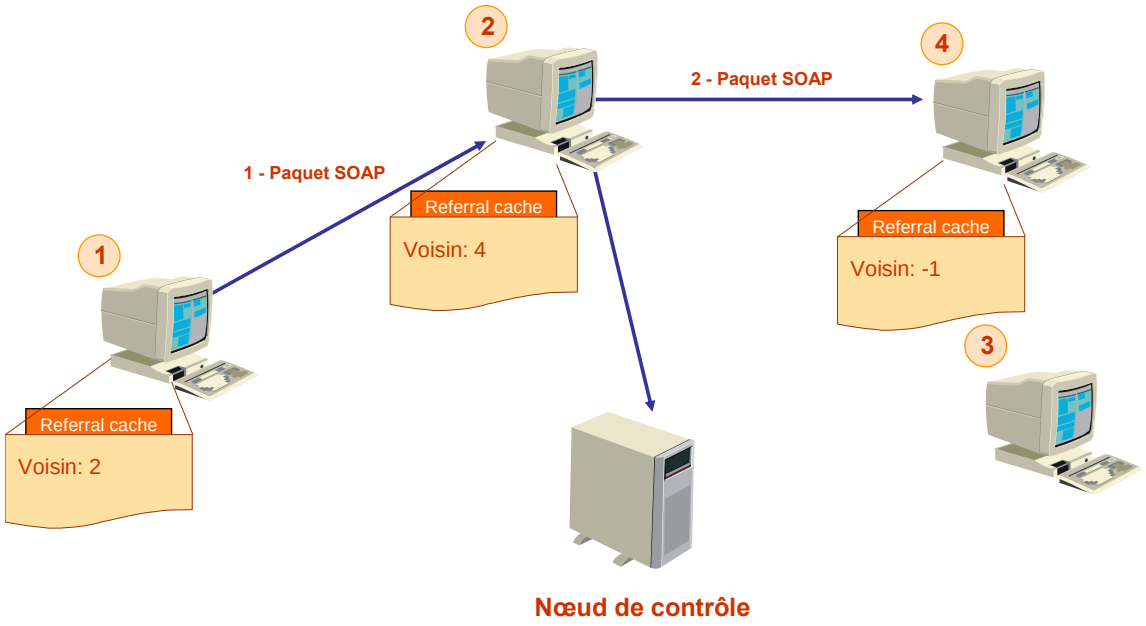

Figure 3.14 - Routage séquentiel

3.6. LA SECURITE DANS ASWA

Dans cette section nous discutons les problèmes de sécurité liés à l'utilisation d'un réseau actif ASWA. Nous commencerons par aborder les problèmes de sécurité des réseaux actifs en général. Nous passerons par la suite à l'étude des problèmes de sécurité liés à l'utilisation des Web services. Enfin nous détaillerons la solution de sécurité que nous avons implémentée au niveau du réseau actif ASWA.

3.6.1. Sécurité d'un réseau actif

Comme les réseaux actifs sont beaucoup plus flexibles que les réseaux passifs, le nombre des questions et de problèmes de sécurité qu'on devrait se poser et résoudre sont de loin plus nombreux [2]. Les paquets actifs contiennent des codes exécutables qui peuvent modifier l'état d'un nœud. Les nœuds deviennent des ressources publiques. Par suite la sécurité au niveau computation, où les codes des paquets sont exécutés, doit être très stricte.

Dans le monde actuel d'Internet, le problème de contrôle strict de ressource au niveau d'un nœud n'est pas nécessaire, car un paquet ne consomme qu'une petite mémoire utilisée pour le stocker. Avec les paquets actifs le problème de réservation de ressources est plus important, car un paquet actif peut consommer beaucoup plus de ressources et plus rapidement, ce qui peut facilement provoquer la saturation d'un nœud.

Dans les réseaux actifs, les paquets actifs peuvent abuser des nœuds actifs, des ressources du réseau et même d'autres paquets actifs. Les nœuds actifs peuvent aussi abuser des paquets actifs. Plusieurs problèmes peuvent donc se produire :

- Dégâts: un paquet actif peut détruire ou bien modifier les ressources ou les services d'un nœud en les reconfigurant, modifiant ou même les effaçant .Un nœud actif peut effacer un paquet actif. Enfin un paquet actif peut attaquer d'autres paquets actifs.
- Un paquet actif peut surcharger un service ou les ressources en consommant la capacité CPU du nœud, ce qui saturerait ce dernier et de nouveaux paquets ne pourront plus être exécutés.
- Un paquet actif peut avoir accès à des informations privées d'un nœud .Un nœud actif peut modifier le contenu d'un paquet.
- Un utilisateur peut envoyer un grand nombre de paquets actifs vers un routeur pour le bloquer en consommant toute sa capacité en ressource.

La protection des paquets et des nœuds actifs dans un environnement flexible comme les réseaux actifs est un problème très difficile à résoudre.

Pour la protection des nœuds actifs nous citons les techniques suivantes:

- Authentification des paquets actifs: chaque paquet doit avoir une authentification produite par des algorithmes de signature à clé publique.
- Surveillance et contrôle: un surveillant consulte la politique de sécurité pour déterminer le droit d'accès pour chaque paquet.
- Les deux premières techniques ne garantissent pas que le paquet ne fera pas de dégât après son exécution. Pour cela on a besoin d'une technique de limitation de ressources: Les limites peuvent être le temps maximal d'exécution permis à un paquet ou bien le nombre maximal de nœud que peut traverser un paquet (avec ses descendants).
- Le créateur d'un programme doit accompagner ce dernier d'un certificat (indiquant que le programme est correct) dans le paquet actif.

Pour la protection des paquets actifs deux méthodes se présentent :

- Chiffrement: Les paquets actifs sont chiffrés avant d'être transmis.
- Tolérance aux erreurs : cette technique est composée de la :
  - Reproduction: chaque paquet est reproduit à chaque nœud.
  - Persistance: les paquets sont temporellement mémorisés, en cas de panne d'un nœud le paquet n'est pas perdu.
  - Redirection: Un paquet pourra changer sa route en cas de coupure de lien.

La reproduction et la persistance ne sont pas adaptées à la majorité des réseaux car ils consomment beaucoup de mémoire et de bande passante. La redirection et le chiffrement consomme de la puissance CPU. La combinaison du chiffrement et de la tolérance aux erreurs pourrait être une solution efficace au problème de sécurité des paquets actifs.

3.6.2. Sécurité des Web Services

Notre environnement d'exécution est constitué d'applications et de Web services qui s'échangent des paquets SOAP. Or ce type de message est envoyé en texte clair ce qui expose les données transmises à la divulgation. D'autre part l'architecture de l'environnement étant basée sur l'existence du nœud de contrôle et celui d'administration et de décision, il faut garantir que les pouvoirs de contrôle et/ou de décision ne soient fournis qu'à des nœuds identifiés pour s'assurer que n'importe quel nœud ne puisse envoyer des règles erronées. Pour cela nous avons envisagé un service de certification où chaque nœud du domaine utilise son certificat :

- d'une part pour signer le message à envoyer et utilise celui de l'émetteur pour authentifier la source d'un message reçu
- et d'autre part pour chiffrer et déchiffrer le contenu de la balise « Body » du message SOAP.

3.6.3. Solution de sécurité dans ASWA

Le modèle de sécurité de ASWA permettra d'assurer les faits de base suivants :

- Un nœud de contrôle doit s'être authentifié par du Face à Face auprès du nœud d'administration pour que ce dernier accepte son enregistrement. En conséquence le nœud d'administration fournira un certificat à ce nœud de contrôle et mettra à jour la liste des services supportés à partir de la liste des services de ce dernier.
- Le code de chaque application active déployée sera signé en utilisant le certificat délivré par le nœud d'administration avant son envoi. Quand un nœud actif reçoit ce code à partir d'un nœud de contrôle, il doit vérifier la signature du nœud de contrôle en vue d'authentifier la source du code reçu. La vérification de la signature est possible à travers la réception du certificat dans l'entête du message SOAP avec le code signé. Ce message SOAP transportant le code signé est conforme à la spécification WS-Security [29] et il a la forme suivante :

```
<soap:Header>
        <!-- Entête du paquet actif -->
    <!--Entête commune -->
```

```
    <ProtocolID>ProtocolIDHere</ProtocolID>
    <!--Entête de Routage  -->
     <!--Entête spécifique à chaque AA -->
    <wsse:Security>
      <wsse:BinarySecurityToken
              ValueType="wsse:X509v3"
              EncodingType="wsse:Base64Binary"
              Id="X509Token">
            MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
      </wsse:BinarySecurityToken>
    </wsse:Security>
 </soap:Header>
 <soap:Body>
    <data>dataHere</data>
 </soap:Body>
</soap:Envelope>
```

Figure 3.15 - Certificat X509 dans un message SOAP

- Un message SOAP est envoyé en clair exposant les données sous forme de texte à la divulgation. Ceci rend notre réseau à topologie ouverte à cause de l'échange des tables de routage et des chemins. Nous proposons comme solution, de chiffrer le corps des messages SOAP en utilisant le chiffrement recommandé par la spécification WS-Security.

```
<soap:Header>
       <!-- Entête du paquet actif -->
     <!--Entête commune -->
    <ProtocolID>ProtocolIDHere</ProtocolID>
    <!--Entête de Routage  -->
     <!--Entête spécifique à chaque AA -->
    <wsse:Security>
      <wsse:BinarySecurityToken
              ValueType="wsse:X509v3"
              EncodingType="wsse:Base64Binary"
              Id="Id-114c8848-78e1-47eb-b5ce-19353c51004c">
            MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
      </wsse:BinarySecurityToken>
    </wsse:Security>
 </soap:Header>
 <soap:Body wsu:Id="Id-114c8848-78e1-47eb-b5ce-19353c51004c">
    <xenc:EncryptedData Id="EncryptedContent-ed7286ea-667c-4af0-9a82-613f3eebec33"
             Type=http://www.w3.org/2001/04/xmlenc#Content
             xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc" />
    <xenc:CipherData>
                    <xenc:CipherValue>
                    YOpl9ot+rvRDsd0cIp4y/vrxQo9dGKbf6EweZeEUV3Vq
                    +rOhTY8QbuxmTHDm4rWlmxQ/k/e1Oh+WbXG6GhEb
                    7DlmMP8v56Y916DUzsfGdJurkMJvtv3UIFgXxv16MrjU
             </xenc:CipherValue>
                    </xenc:CipherData>
  </xenc:EncryptedData>
 </soap:Body>
</soap:Envelope>
```

Figure 3.16 - Message SOAP chiffré conformément à la spécification WS-Security

- La haute disponibilité des services actifs est assurée par l'implémentation d'une topologie de réseau transparente en permettant aux paquets de changer leur route en cas de coupure de lien. Cette technique s'appelle la redirection et est réalisée à l'aide d'un routeur WSE implé-

menté conformément à la spécification WS-Routing [30].

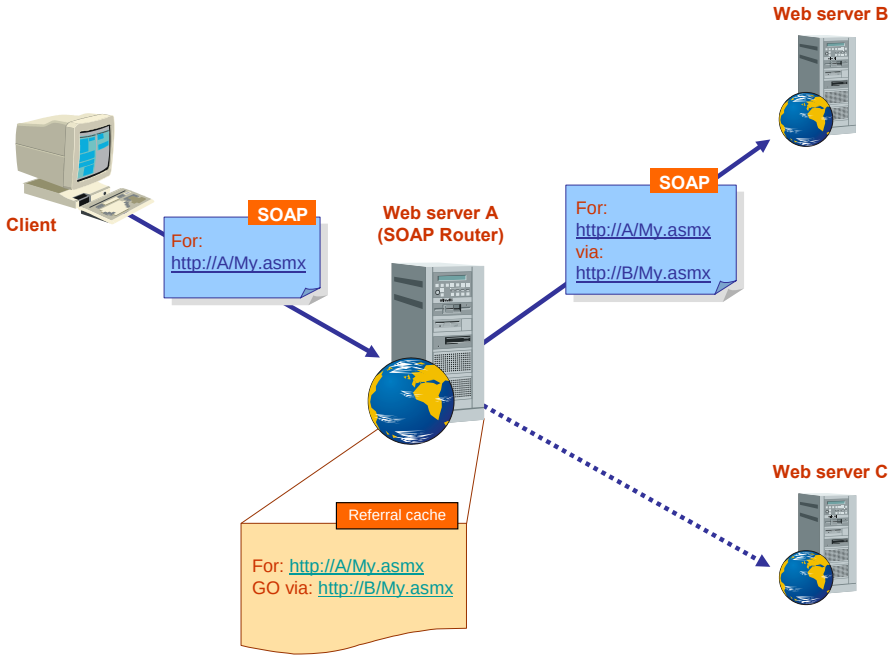

Figure 3.17 - Routage WSE

Après l'installation du routeur WSE au niveau d'un ordinateur intermédiaire, le client transmettra les messages SOAP vers ce routeur intermédiaire au lieu de les transmettre directement vers le service Web. Ensuite le routeur transmettra ces messages vers le service Web destinataire, dont l'emplacement peut être changé de façon transparente par simple modification du fichier de configuration (« Referral Cache ») de ce routeur intermédiaire. Ainsi, si le service Web destinataire tombe en panne, il suffit de l'arrêter et d'acheminer le message SOAP vers un serveur de secours. Une fois le serveur principal réparé, les messages SOAP sont re-acheminés vers ce serveur principal de façon flexible et transparente.

La Figure 3.17 nous montre la transmission d'un message SOAP vers un routeur WSE, qui à son tour transmet le message vers un autre serveur Web en se basant sur le contenu de son fichier *referral cache*. Dans notre cas le *referral cache* indique que le message doit être transmis vers le serveur B. Dans le cas où le serveur B ne répond pas aux requêtes, le contenu de ce *referral cache* sera mis à jour pour que la transmission de ce message se fasse vers le serveur C par exemple.

3.7. INSTALLATION ANTICIPE DU CODE DANS ASWA

La demande de déploiement du code d'un Web service actif sur un nœud ou d'un fichier de configuration est initiée par une application cliente à partir de l'interface Web de la Figure 3.18. L'accès à cette interface se fait à partir du nœud de contrôle du domaine et est permis à un administrateur après un processus de login. Après l'étape de l'authentification, l'initiation de la demande n'est acceptée que si cet administrateur en a bien l'autorisation. Dans ce cas une requête SOAP précisant l'application active à déployer est envoyée au service d'« installation de code» du nœud actif, qui téléchargera le service demandé à partir du nœud de contrôle en possession du code correspondant. Notons que chaque nœud de contrôle possède au niveau de son cache un ensemble de services actifs qui y sont initialement stockés. L'administrateur du réseau a la possibilité d'ajouter ou de supprimer des Applications Actives au niveau de ce cache.

Les informations devant être fournies par l'administrateur à partir de l'interface Web de la Figure 3.18 sont les suivantes:

- Le chemin sur lequel doit se faire le déploiement. Un chemin est représenté par un nœud de départ (le champ « Start Node ») et une destination (le champ « Target Node»).

- l'endroit où se trouve le Web service. Il faut distinguer le cas où c'est le nœud de contrôle du même domaine qui est en possession du code, ou bien celui d'un autre domaine. Dans ce dernier cas, la requête doit passer par le nœud d'administration selon la Figure 3.7.
- le nom du Web service à installer. Dans le cas où le service demandé se trouve sur un nœud de contrôle d'un autre domaine, c'est l'URL absolue du service qui doit être saisie.
- dans le cas d'installation de fichiers de configuration ou autres, le nom du fichier devra être précisé pour être téléchargé. Citons par exemple les fichiers de règles de filtrage et des ACLs utilisés par l'application active « pare-feu distribué ».
- L'utilisateur a le choix entre trois modes de routage (voir section 3.5.5.2) :
  - Le premier mode ou le mode intégré dans lequel le nœud de départ demande au nœud de contrôle le chemin complet entre le nœud courant et le nœud destination qui sera intégré dans chaque paquet.
  - Le deuxième mode ou le mode progressif dans lequel chaque nœud demande au nœud de contrôle le nœud suivant.
  - Le troisième mode ou le mode séquentiel dans lequel chaque nœud cherche dans son fichier de routage (« Referral Cache ») l'adresse du nœud suivant.

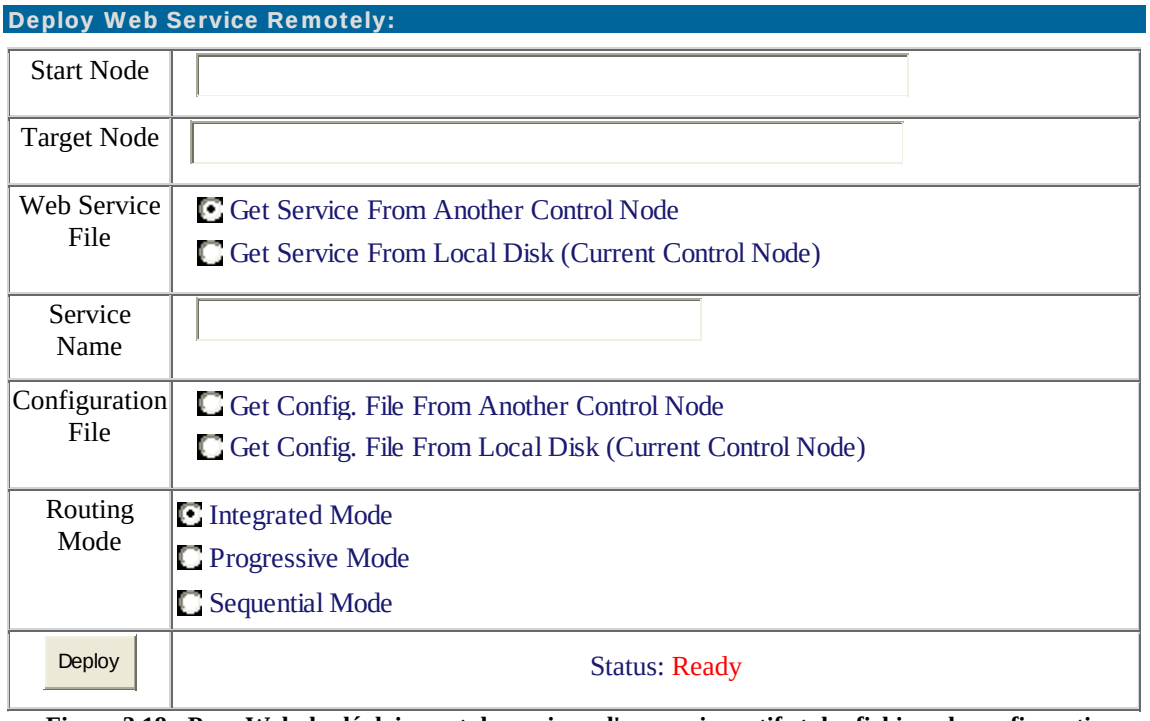

Figure 3.18 - Page Web du déploiement dynamique d'un service actif et des fichiers de configuration

Une fois que les routines de traitement personnalisées sont injectées sur les nœuds du chemin construit par le service de routage dynamique, les utilisateurs pourront envoyer leurs capsules dans ces nœuds programmables, de la même façon qu'avec les réseaux classiques. Quand une capsule arrive à un nœud, elle est reçue par la composante NodeOS qui la passe à la composante EE pour la traiter et la router vers le nœud « suivant ».

Cette méthode de déploiement de code où le mécanisme de chargement et celui d'exécution de code sont séparés, permet de mieux contrôler le chargement de code, qui peut être restreint dans notre cas à des utilisateurs autorisés suivant une liste de contrôle d'accès bien définie au niveau de chaque nœud de contrôle.

3.8. UNE APPLICATION ACTIVE: UN PARE-FEU DISTRIBUE

3.8.1. Définition d'un pare-feu distribué

Avec le nombre croissant d'attaques diversifiées sur les réseaux, la recherche et la réflexion sur de nouvelles propositions et de nouvelles approches pour la sécurité sont devenues de plus

en plus urgentes. Un des piliers de la sécurité des réseaux est le filtrage basé principalement sur les pare-feu. Mais vu que les pare-feu traditionnels montrent certaines limitations, de nouvelles propositions émergent en vue de distribuer le filtrage et de le rendre plus efficace.

Le filtrage traditionnel ou centralisé qui consiste à utiliser un pare-feu au périmètre du réseau, est une approche qui montre de plus en plus ses limites en terme de sécurisation des réseaux. Cela est dû à l'emplacement du pare-feu dans la topologie qui crée typiquement 3 zones, une zone interne, une zone externe et éventuellement une DMZ. Les inconvénients d'un tel modèle de filtrage sont les suivants [106] :

- Aucune attaque interne ne peut être détectée.
- Le pare-feu constitue un point de congestion (bottleneck)
- Le pare-feu est un point de vulnérabilité (single point of failure)
- Le pare-feu ne peut pas traiter un trafic chiffré, dans le cas par exemple d'un canal chiffré point à point.
- En cas de connexion secondaire au niveau d'une machine (connexion dial-up ou sans fil), le pare-feu central est contourné.

Le filtrage distribué du trafic est basé sur la distribution du mécanisme de filtrage sur les machines du réseau à protéger. Il propose l'application du mécanisme de filtrage au niveau de chaque machine hôte du réseau et non pas exclusivement sur une seule machine (le pare-feu) pour la protection de tout le réseau. Ceci permettra d'appliquer des politiques de sécurité propres à chaque machine, pour tout trafic entrant ou sortant. Le filtrage distribué peut aussi dépasser le niveau des machines, pour s'appliquer au niveau des utilisateurs réseau, et leur affecter ainsi des politiques de sécurité qui s'appliquent indépendamment de la machine à partir de laquelle ils ont accès au réseau.

En conclusion, le filtrage distribué sert à éliminer la dépendance de la topologie des pare-feu traditionnels, tout en préservant le contrôle centralisé par l'introduction d'un serveur de politique. Ce serveur sera configuré au besoin par un administrateur en y définissant les règles de filtrage nécessaires.

3.8.2. Conception d'un pare-feu distribué actif

Une ouverture importante proposée par les réseaux actifs est de permettre au comportement des périphériques spécialisés du réseau d'être altérés par les utilisateurs. L'exemple classique peut être le "pare-feu distribué". Les pare-feu effectuent déjà du filtrage basé sur les données d'une application lors du traitement de leurs paquets. Plutôt que d'être figé, ce filtrage pourrait être contrôlé par les utilisateurs du domaine protégé pour permettre un contrôle fin des services exportés.

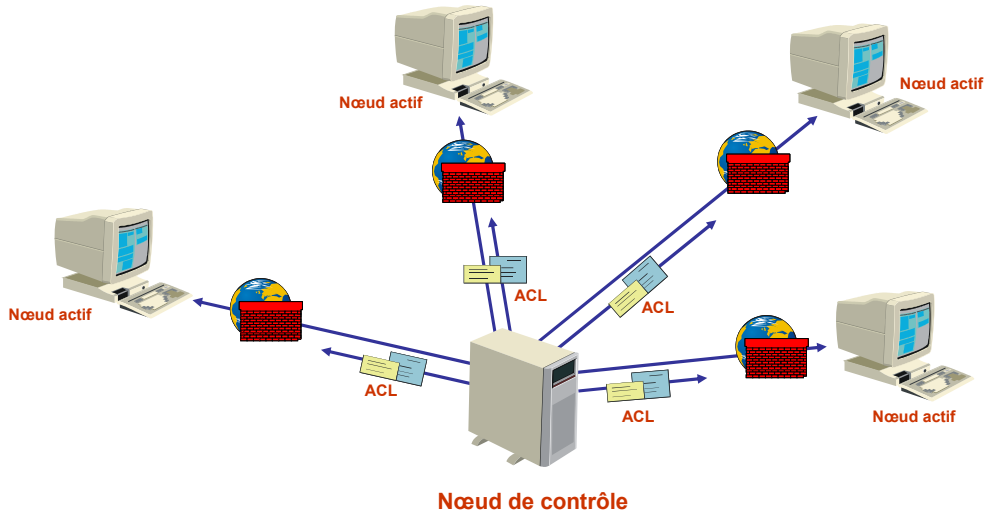

Figure 3.19 - Déploiement anticipé du pare-feu distribué

Pour valider notre plate-forme à l'aide d'un exemple fonctionnel, nous avons développé un « pare-feu distribué » sous forme d'une application active. Ce service ainsi qu'une liste de contrôle d'accès doivent être déployés à partir du nœud de contrôle, sur chaque nœud actif du domaine. Ceci est illustré dans la Figure 3.19.

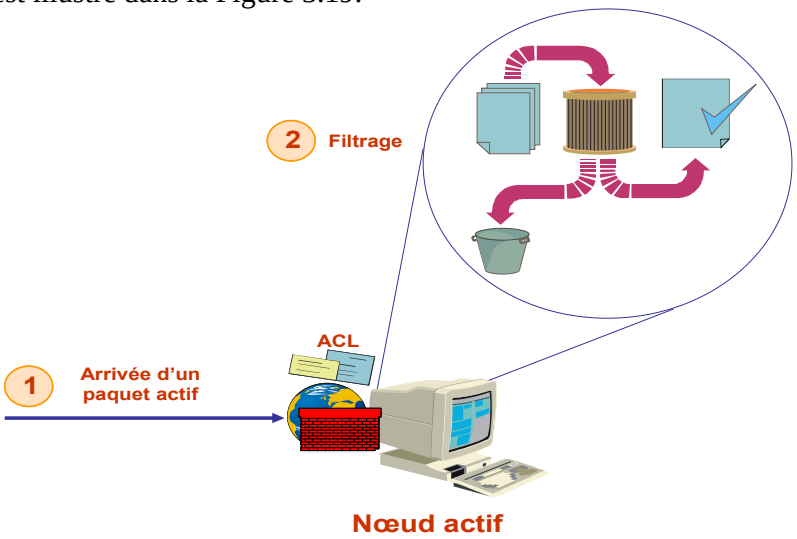

Figure 3.20 - Processus de filtrage

Une fois le pare-feu actif installé, il pourra être sollicité par l'environnement d'exécution pour effectuer le filtrage des paquets reçus avant que ces derniers soient transmis vers l'application active appelée. Ceci est illustré dans la Figure 3.20.

3.8.3. « XML SCHEMA » pour la validation des Listes de Contrôle d'Accès

Le filtrage des paquets se fait en se basant sur des listes de contrôle d'accès (ACLs) déployés sur les différents nœuds actifs. Un « schema XML » permettant de caractériser les règles de filtrage par groupe d'utilisateurs a été défini. Une ACL déployée sur un nœud actif ne serait en fait qu'un fichier XML validé par ce « schema », il sera dit "instance" de ce « schema ». Le format de ce « *schema* » est décrit à la Figure 3.21 suivante:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://localhost/XMLSchema" elementFormDefault="qualified"
        xmlns="http://localhost/XMLSchema"
        xmlns:mstns=http://localhost/XMLSchema
        xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:simpleType name="actionType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="allow" />
            <xs:enumeration value="deny" />
        </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="timeRangeType">
        <xs:sequence>
            <xs:element name="startTime">
                <xs:simpleType>
                        <xs:restriction base="xs:string">
                                <xs:pattern value="([01]\d|2[0-3]):[0-5]\d:[0-5]\d" />
                        </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="endTime">
                <xs:simpleType>
                        <xs:restriction base="xs:string">
                                <xs:pattern value="([01]\d|2[0-3]):[0-5]\d:[0-5]\d" />
```

```
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
        <xs:complexType name="ruleType">
            <xs:sequence>
                <xs:element name="containWords" type="xs:string" minOccurs="0" maxOccurs="1" />
                <xs:element name="timeRange" minOccurs="0" maxOccurs="100" type="timeRangeType">
                </xs:element>
                <xs:element name="srcaddr" type="xs:string" maxOccurs="100" minOccurs="0" />
                <xs:element name="dstaddr" type="xs:string" minOccurs="0" maxOccurs="100" />
                <xs:element name="srcport" type="xs:short" maxOccurs="100" minOccurs="0" />
                <xs:element name="dstport" type="xs:short" minOccurs="0" maxOccurs="100" />
                <xs:element name="protocol" type="xs:string" minOccurs="0" maxOccurs="10" />
                <xs:element name="UserGroup" type="xs:string" minOccurs="0" maxOccurs="100" />
                <xs:element name="srcPortRange" type="portRangeType" minOccurs="0" maxOccurs="100"/>
                <xs:element name="dstPortRange" type="portRangeType" maxOccurs="100" minOccurs="0"/>
                <xs:element name="srcAddrRange" type="addrRangeType" maxOccurs="100" minOccurs="0" />
                <xs:element name="dstAddrRange" type="addrRangeType" minOccurs="0" maxOccurs="100"/>
            </xs:sequence>
            <xs:attribute name="action" type="actionType" />
        </xs:complexType>
        <xs:complexType name="portRangeType">
            <xs:sequence>
                <xs:element name="fromPort" type="xs:short" />
                <xs:element name="toPort" type="xs:short" />
            </xs:sequence>
        </xs:complexType>
        <xs:complexType name="addrRangeType">
            <xs:sequence>
                <xs:element name="fromAddr" type="xs:string" />
                <xs:element name="toAddr" type="xs:string" />
            </xs:sequence>
        </xs:complexType>
        <xs:element name="aclRoot">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="aclRule" type="ruleType" minOccurs="0" maxOccurs="10000" />
                </xs:sequence>
            </xs:complexType>
        </xs:element>
</xs:schema>
```

Figure 3.21 - "XML SCHEMA" des ACLs

Ce « XML SCHEMA » est organisé de la façon suivante :

- L'élément « aclRoot » est l'élément racine du « XML schema ». C'est une séquence d'éléments « aclRule ».
- Chaque élément « aclRule » est associé à un groupe d'utilisateurs. Il définit le nom du groupe « userGrp » et les règles de filtrage associées à ce dernier. Les règles de filtrage définies au niveau de notre *schema* permettent d'effectuer un filtrage selon: le contenu du paquet (« containWord »), l'adresse source de ce paquet, son adresse destination, le protocole (TCP, IP, UDP, ICMP, ou n'importe quel ProtocolID qui représente une application active), le numéro de port source, le numéro de port destination, ainsi que selon un timeRange qui détermine la plage horaire où le paquet a le droit de traverser le noeud.
- A chaque élément « aclRule » est associé un attribut appelé « action » qui peut prendre les

valeurs «allow» ou «deny». Cet attribut définit l'action à effectuer au cas où le pare-feu trouve une correspondance entre l'entête du paquet reçu et le contenu de cet élément: Soit on permet au paquet de passer (allow), soit ce paquet est rejeté (deny).

La Figure 3.22 suivante est un exemple particulier d'un fichier XML d'une ACL validée par ce « XML schema ».

```
<?xml version="1.0" encoding="utf-8" ?>
<aclRoot xmlns="http://tempuri.org/aclSchema.xsd">
    <aclRule action="allow">
        <userGrp>ASWAusers</userGrp>
        <dstAddrRange>
                <fromAddr>10.10.10.11</fromAddr>
                <toAddr>10.10.10.23</toAddr>
        </dstAddrRange>
        <srcAddr>10.10.10.2</srcAddr>
        <protocol>udp</protocol>
        <srcPortRange>
                <fromPort>20</fromPort>
                <toPort>25</toPort>
        </srcPortRange>
        <containWord>maroun</containWord>
        <containWord>rima</containWord>
        <timeRange>
                <startTime>08:00:00</startTime>
                <endTime>12:00:00</endTime>
        </timeRange>
        <timeRange>
                <startTime>14:00:00</startTime>
                <endTime>18:00:00</endTime>
        </timeRange>
    </aclRule>
    <aclRule action= « deny »>
        <userGrp>others</userGrp>
        <srcAddrRange>
                <fromAddr>192.168.3.100</fromAddr>
                <toAddr>192.168.3.120</toAddr>
        </srcAddrRange>
        <srcAddr>10.10.10.2</srcAddr>
        <protocol>tcp</protocol>
        <dstPort>80</dstPort>
    </aclRule>
</aclRoot>
```

Figure 3.22 - Fichier XML servant comme ACL

Une interface Web graphique développée pour faciliter la gestion des ACLs (insertion, modification, suppression et consultation) a été créée pour être utilisée par un administrateur du réseau pour la création et la gestion de tels fichiers.

3.8.4. Mise en œuvre du pare-feu actif sous ASWA

L'interface Web qui permet de gérer le fonctionnement du Service d'installation de code et de fichiers a été décrite au niveau de la section 3.7. C'est cette interface Web (Figure 3.18) qui sera utilisée pour déployer le Web service du pare-feu actif et son ACL. Une fois le pare-feu actif installé, il pourra être sollicité par l'environnement d'exécution pour filtrer les paquets entrants et sortants en se basant sur les règles de filtrage déployées sur ce nœud actif.

Pour être reconnu et traité par notre environnement d'exécution, un paquet devra avoir la forme suivante (Figure 3.23):

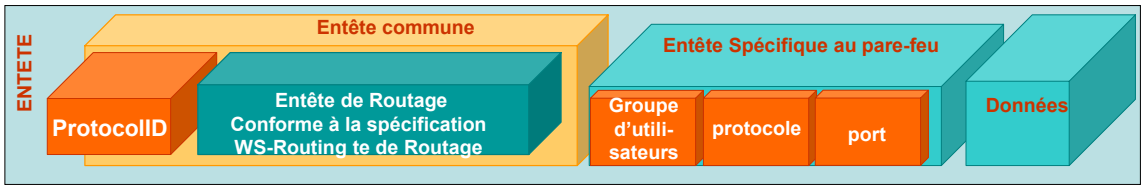

Figure 3.23 - Format d'un paquet du pare-feu actif

Un exemple de message SOAP encapsulant ce paquet est le suivant :

```
<soap:Envelope xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsse=http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
 <soap:Header>
    <wsa:Action>http://tempuri.org/HelloWorld</wsa:Action>
    <wsa:MessageID>uuid:28e5bc6b-4c68-4198-9926-cdd58f4687ff</wsa:MessageID>
    <wsa:ReplyTo>
       <wsa:Address>
         http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
       </wsa:Address>
    </wsa:ReplyTo>
    <wsa:To>http://localhost/aswa2/Test/Test.asmx</wsa:To>
    <!--Entête du paquet actif-->
    <aswa:ASWA aswa:ProtocolID="Test" xmlns:aswa="http://fi.usj.edu.lb/aswa">
     <ProtocolID>Test</ProtocolID>
    </aswa:ASWA>
    <ASWAFirewallHeader>
     <srcaddr>34.34.34.3</srcaddr>
     <dstaddr>192.168.0.128</dstaddr>
     <srcport>1234</srcport>
     <dstport>80</dstport>
     <protocol>Test</protocol>
     <UserGrp>ASWA</UserGrp>
    </ASWAFirewallHeader>
    <wsse:Security>
     <wsu:Timestamp wsu:Id="Timestamp-f61e9b92-a5f8-4b0e-8bf2-e7807a7cf784">
       <wsu:Created>2005-05-31T04:20:00Z</wsu:Created>
       <wsu:Expires>2005-05-31T04:25:00Z</wsu:Expires>
     </wsu:Timestamp>
    </wsse:Security>
  </soap:Header>
  <soap:Body>
   <HelloWorld xmlns="http://tempuri.org/" />
  </soap:Body>
 </soap:Envelope>
```

Figure 3.24 - Paquet ASWA encapsulant une entête spécifique au pare-feu distribué

Une fois ce paquet reçu par le module de filtrage de l'environnement d'exécution, le fichier ACL sera utilisé pour vérifier si ce paquet, provenant d'un utilisateur appartenant à un groupe donné, devra être filtré ou routé vers le nœud suivant, et ceci selon les étapes suivantes:

La fonction de filtrage effectue l'analyse des balises XML de ce fichier ACL présent sur le nœud, à la recherche du groupe auquel appartient l'utilisateur. Une fois le groupe trouvé, les autres champs de l'entête du paquet spécifique à l'application active du pare-feu, seront comparés un à un avec les éléments de l'ACL correspondant (au niveau du tag « aclRule »). Dans le cas d'une correspondance entre un champ de l'entête et un élément, il faut agir en fonction de

l'attribut « action » de cet élément :

- Si la valeur de l'action est « deny », le pare-feu prépare un message SOAP spécifiant la cause du blocage avant de retourner « FALSE » et bloquer le passage du paquet.
- Si la valeur de l'action est « allow », deux cas se présentent en fonction de la donnée « allowDirectly » :
  - Si elle vaut « TRUE », il faut alors arrêter le filtrage et permettre au paquet de franchir le nœud.
  - Si elle vaut « FALSE », il faut continuer la comparaison.

3.9. CONCLUSION DU CHAPITRE

L'utilisation des services Web dans la modélisation du nœud actif (Composante NodeOS, Composante EE et composante AA) ainsi que dans le chargement dynamique du code des Applications Actives qui se fait à partir d'une interface Web et à travers des systèmes et des réseaux hétérogènes, donne à cette approche un aspect innovateur :

– L'architecture de ASWA permet aux Applications Actives d'être automatiquement déployées aux nœuds du réseau qui en ont besoin en vue de modifier dynamiquement le comportement de ces nœuds.

– Aucun consensus n'est requis au préalable à propos du type et de la définition de l'application active à déployer.

– ASWA offre la transparence et l'interopérabilité vis-à-vis de systèmes et de réseaux hétérogènes. Ceci est assuré par la possibilité de choisir n'importe quel langage de programmation au niveau du développement du code ainsi que par l'utilisation du protocole SOAP comme protocole de communication.

– L'architecture et l'implémentation de ASWA permettent d'offrir l'interopérabilité avec d'autres plates-formes en particulier avec ANTS, par simple composition.

– ASWA profite de la spécification « WS-Security » pour la mise en œuvre d'une authentification automatique et transparente.

– ASWA profite de la spécification « WS-Routing » de W3C pour la mise en œuvre d'un protocole de routage applicatif. Cette spécification permet d'utiliser l'entête du message SOAP pour définir le chemin entre un nœud source et un nœud destination en passant par des nœuds intermédiaires. Cette spécification a l'avantage d'une part d'être indépendante de la plate-forme et du protocole utilisé et d'autre part d'alléger notre architecture du réseau actif à base de Web services.

– Deux applications ou services actives ont été implémentés et déployés sur la plateforme ASWA essentiellement pour sa validation :
  - Un pare-feu distribué. Comme on a vu dans ce chapitre, ce service ainsi qu'une liste de contrôle d'accès doivent être déployés à partir du nœud de contrôle, sur chaque nœud actif du domaine. Une fois le pare-feu actif installé, il pourra être sollicité par le service de routage de l'environnement d'exécution pour effectuer le filtrage des paquets reçus.
  - Un protocole de gestion de réseaux par politiques basé sur le protocole COPS, et où la communication se fait à l'aide de Web Services fournis par ASWA, la définition des règles de politiques et la représentation de la base des informations de politiques se fait en utilisant le langage OWL ce qui améliore la représentation sémantique de ces informations. Les détails de ce protocole seront explicités au Chapitre 5.

– Le développement de la plateforme ASWA fut effectué sur une plateforme Windows en tenant compte de la portabilité sur d'autres plateformes. La portabilité ainsi que l'interopérabilité avec Linux devront être testées.

Ayant vu tous les détails de l'architecture de la plateforme ASWA au niveau de ce chapitre, nous expliciterons dans le chapitre suivant, tous les aspects liés à l'implémentation de cette plateforme.

Chapitre 4

IMPLEMENTATION DE ASWA

4.1. INTRODUCTION

La conception des réseaux actifs permet le déploiement du code à distance. ASWA, s'applique dans le cas des réseaux formés de plusieurs domaines donc à des réseaux à large échelle. Ayant donc besoin d'un environnement d'exécution distribué et sachant que la plupart des services doivent être fournis à la demande, nous nous demandons qu'est ce qu'il y a de mieux qu'un environnement basé sur des Web services (section C.4) qui s'échangent des messages SOAP (section C.4.8) « un langage accepté dans le monde d'Internet ». Ainsi, les composants de notre environnement assureront la portabilité au niveau des systèmes d'exploitation ainsi qu'au niveau des langages de programmation.

D'autre part, pour être réussite, toute implémentation de réseau actif devrait satisfaire les conditions suivantes :

- Le réseau doit être utilisable: le réseau doit présenter le moins de contraintes possible vis-à-vis de l'utilisateur.
- Le réseau doit être flexible: la flexibilité est la raison primaire derrière les recherches dans ce domaine. Le système doit pouvoir s'adapter à une grande variété de tâches.
- L'implémentation doit être sécurisée: la sécurité est l'obstacle majeur devant le déploiement des réseaux actifs.
- L'implémentation doit permettre la distribution des charges et doit assurer l'intégrité du code déployé.
- Le réseau doit assurer une haute performance: les réseaux actifs ne doivent pas créer de nouveaux rétrécissements dans l'infrastructure. En particulier, l'implémentation devrait assurer un acheminement rapide des paquets traditionnels.

Pour répondre à ces besoins :

- Nous avons créé des Web services de routage applicatif (section C.5.3) qui assurent une distribution de charges.
- Nous avons ajouté à ces Web services une fonction de filtrage (section C.5.2) qui arrête un paquet ou le transmet au nœud suivant de son chemin après avoir comparé son entête SOAP aux règles de filtrage dans une base de données locale.
- Enfin, nous avons éliminé nos soucis d'intégrité, de confidentialité et de non répudiation des données utiles en ajoutant des fonctionnalités de signature et de chiffrement utilisant des certificats X.509 (section C.5.4).

Pour la réalisation de la plateforme ASWA, notre choix s'est porté sur les outils Microsoft (section C.1). Ce choix se justifie essentiellement par la convivialité des outils de développements des « Web services ». Notre objectif n'est absolument pas celui de faire la promotion de tel ou tel environnement. Mais bien celui de prouver la faisabilité de ASWA telle que décrit dans

le chapitre de conception.

Par ailleurs pour peut être répondre à des exigences qui consistent à travailler dans des environnements ouverts, nous pouvons signaler l'existence de la plateforme .NET dans un contexte non Microsoft. En effet le projet « Mono » prend de plus corps et réalité. Le projet Mono [113] ouvre la voie à une implémentation de l'architecture .NET sur des OS concurrents de Microsoft comme Linux, Solaris, FreeBSD et Mac OS X. Cette implémentation inclut CLR, C# et VB, les librairies de base, ADO.NET, ASP.NET. Ceci permet à .NET de devenir de plus en plus indépendant de la plateforme et de Microsoft.

Dans la suite de ce chapitre nous allons commencer par décrire les fonctionnalités du NodeOS dans notre implémentation assurées par des couches pré installées. Ensuite nous allons décrire l'environnement d'exécution que nous avons implémenté à l'aide de la plateforme .Net, des Web Services et du Web Services Enhancements 2.0. Enfin, nous allons conclure en résumant les caractéristiques de ASWA.

4.2. NODEOS

Le Nœud Actif est généralement architecturé autour d'un Système d'Exploitation du Nœud (Node Operating System ou NodeOS) offrant le support système en matière d'allocation et de gestion des ressources du noeud actif (l'ordonnancement de tâches, la bande passante des liens, la mémoire, le CPU, le stockage).

Le NodeOS fournit un ensemble de capacités de base utilisées par des Environnements d'Exécution (EEs) pour implémenter une machine virtuelle. Le NodeOS isole un EE de la gestion des ressources et de l'effet du fonctionnement des autres EEs. De leur côté les EEs cachent au NodeOS les détails de l'interaction avec les utilisateurs.

Les ressources du nœud contrôlées par un NodeOS sont les suivantes :

- Temps CPU : A chaque code actif est associé un temps d'exécution maximal (défini dans ses politiques de sécurité). Lors de son exécution, un timer est lancé par le nœud. A son expiration, le thread dans lequel le code s'exécute est terminé.
- La mémoire: ceci inclut la gestion du code actif dans la mémoire cache sur un nœud.
- Les processus et les threads : un service est lancé comme étant un processus et peut utiliser plusieurs threads.
- Les ressources réseau sont contrôlées par exemple en limitant le nombre de capsules pouvant être émises durant l'exécution d'un code actif. Il est possible d'utiliser le mécanisme de type TTL (Time To Live). Les accréditations associées au code permettent de spécifier le nombre de capsules autorisées pour un service donné.
- Canaux : un service peut demander la création de canaux spécifiques entre plusieurs nœuds.
- Les flux sont des ressources créées par un service. La modification des paramètres d'un flux est soumise au contrôle d'accès. Le NodeOS ou l'EE peuvent imposer des restrictions sur le flux (par utilisateur, par service, …).

Dans notre implémentation, les fonctionnalités d'un NodeOS sont assurées par les couches pré-installées sur le nœud actif ASWA suivantes (Figure 4.1):

- La plateforme .NET (section C.2)
- Le serveur d'application ASP.NET (section C.3.2)
- Le serveur Web IIS (section C.3.1)

En effet, le contrôle des ressources est assuré comme suit :

- Temps CPU : Le serveur Web IIS limite le temps CPU d'une application en utilisant le mécanisme nommée *process accounting interval*.
- Mémoire: Ce n'est pas parce qu'un élément peut être mis en cache qu'il faut l'y mettre: la gestion de l'élément et de la mémoire qu'il consomme n'est pas sans coût. C'est pourquoi IIS utilise une fonction heuristique pour déterminer quels éléments mettre en cache en fonction de

la distribution des requêtes reçues par une application spécifique. Ceci signifie que la tenue du serveur Web s'améliore lors d'une montée en charge car il fait un meilleur usage des ressources dont il dispose, tout en maintenant un haut niveau de performance sur les requêtes fréquentes.

- Les processus et les threads: Le Thread Pool est un composant .Net permettant une gestion efficace des threads multiples au sein d'une application .Net.
  D'autre part, les applications Web s'exécutent dans un processus isolé. Les applications Web gérées par IIS ne partagent pas de ressources communes. Ainsi, le dysfonctionnement d'une application ne peut affecter l'exécution d'une autre application. Il y a un confinement des ressources au sein de chaque processus.
  Comme il n'y a plus de ressource partagée, le serveur IIS se concentre à contrôler et surveiller ses applications Web. Un contrôle très fin peut aussi être réalisé afin de tester le bon fonctionnement d'une application Web. En cas de faille de l'une d'entre elles, le serveur IIS est même capable d'en relancer une nouvelle instance.
- Les ressources réseau de notre plateforme active et le flux sont contrôlées par le système d'exploitation Windows lui-même. En effet, le pilote en mode noyau, HTTP.SYS, constitue un point de contact unique pour toutes les requêtes HTTP entrantes. Ceci permet d'obtenir des performances élevées des applications serveur HTTP. Le pilote est installé au-dessus du protocole TCP/IP. Il ne s'intéresse qu'aux combinaisons adresses IP/numéros de ports TCP pour lesquelles il a été configuré. HTTP.SYS est également chargé de la gestion globale des connexions, de la limitation de la bande passante et de la gestion des sessions sur les serveurs Web.
- Canaux : Dans le .NET Framework, quand un client appelle une méthode définie dans un objet distant, l'infrastructure distante sérialise automatiquement les données associées à la demande et transporte les données vers l'objet distant par le biais d'un canal. Le .NET Framework fournit les canaux HTTP et TCP mais des tiers peuvent écrire et brancher leurs propres canaux. Dans le cas du protocole HTTP, l'infrastructure utilise le protocole SOAP pour transporter des données au format XML du client vers le serveur et inversement. Le formateur de sérialisation par défaut pour HTTP est un formateur SOAP. Étant donné que les programmeurs peuvent créer des formateurs personnalisés pour les utiliser avec un canal quelconque, l'infrastructure distante peut être configurée pour fonctionner avec un .NET Framework externe quelconque sur d'autres plateformes. Le canal TCP utilise des sockets simples et la sérialisation binaire par défaut et peut être utilisé pour communiquer avec n'importe quel objet sur un serveur distant.

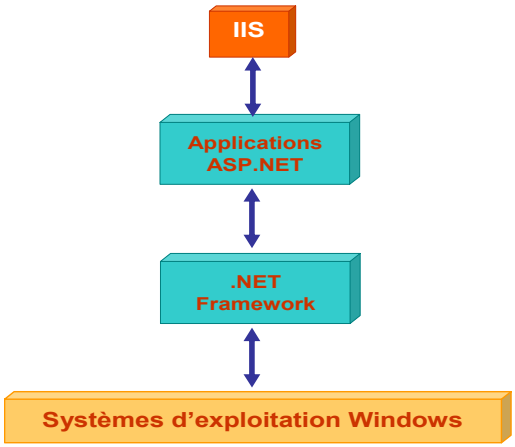

Figure 4.1 - Les couches du NodeOS dans ASWA

4.3. ENVIRONNEMENT D'EXECUTION D'UN NŒUD ACTIF

Un environnement d'exécution (EE) doit fournir des fonctionnalités qui permettent aux services actifs de s'exécuter sur un nœud actif. Le EE permet également de filtrer les paquets et/ou de router les requêtes vers d'autres nœuds actifs.

Même s'il existe beaucoup de plateformes pour exécuter des services actifs dans un réseau, la programmation des services qui travaillent sur des données de différents niveaux de protocole reste encore difficile. Les services actifs sont souvent programmés dans des langages comme Java (ALAN, ANTS), C (Router Plugins, LARA++) ou TCL (AS1). Les codes d'analyse et de formatage des paquets actifs, souvent complexes, source de beaucoup d'erreurs et demandant beaucoup de temps de test, sont combinés au code de traitement des données. Par conséquent, quand la structure de données ou PDU est modifiée, tout doit être recompilé et rechargé. En fait, les utilisateurs ou les programmeurs sont seulement intéressés par connaître comment les données doivent être traitées.

Dans le cadre de cette thèse, la plateforme ASWA offre un EE capable de traiter les requêtes ASWA qui ne sont autre que des pages Web d'extension « aswa ». Cet environnement d'exécution est implémenté sous forme d'un gestionnaire de requêtes HTTP (section C.3.3 et section C.3.4). Il aide les programmeurs et les utilisateurs des applications actives:

- en construisant automatiquement les paquets actifs.
- en générant d'une façon transparente le proxy qui joue le rôle d'interface obligatoire entre un client et un service actif (sous forme d'un Web service).
- en s'occupant du routage des paquets entre les différents nœuds actifs.
- En assurant les services de sécurité et de filtrage.

Pour ce faire, trois modules ont été développés. La Figure 4.2 résume l'architecture de ces trois modules formant l'environnement d'exécution de la plate-forme ASWA.

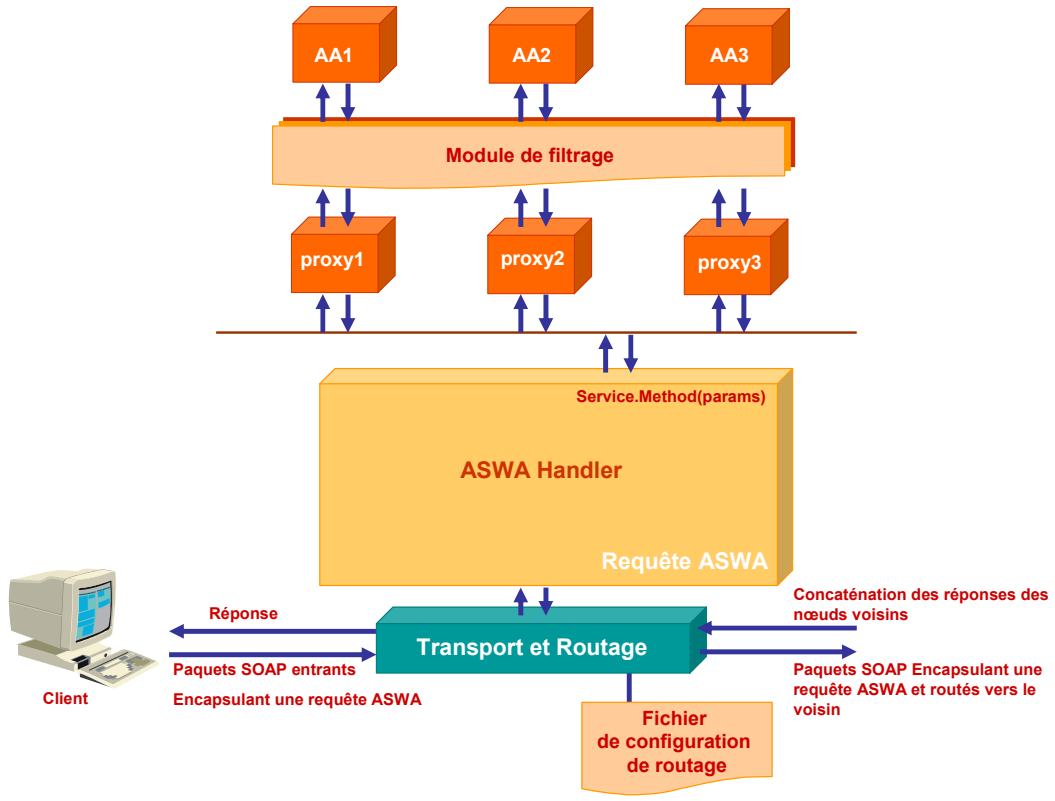

Figure 4.2 - L'environnement d'exécution de ASWA

- Le module de transport et de routage est un Web service qui reçoit une requête ASWA encapsulée dans un message SOAP. La requête est servie localement par le gestionnaire de requêtes HTTP appelé ASWAHandler. Par la suite, cette requête est envoyée vers une fonction de routage qui, en fonction du mode de routage défini au niveau du paquet entrant, choisit le nœud suivant et envoie cette requête vers ce dernier.
- ASWAHandler est un gestionnaire de requêtes HTTP, son rôle principal est de vérifier

l'existence du Web service en question sur la machine locale. Si le service n'est pas installé, il est téléchargé à l'aide du nœud de contrôle, puis un proxy est créé. ASWAHandler utilisera ce proxy pour appeler la méthode Web de l'application active à l'aide du proxy.

- Un ou plusieurs module(s) de filtrage sont installés soit entre le client et le module de transport, soit entre le proxy et l'application active. Le rôle de ces modules est d'appliquer les services de sécurité et de filtrage sur les messages échangés.

4.3.1. Le gestionnaire de requête ASWAHandler

Un gestionnaire HTTP est une classe capable de traiter un fichier avec une certaine extension. ASWAHandler est une classe qui traite les fichiers ayant l'extension « .aswa ». Cette classe implémente l'interface IHttpHandler :

```
public interface IHttpHandler
{
    bool IsReusable {get;}
    void ProcessRequest(HttpContext context);
}
```

Elle expose les deux membres suivants:

IsReusable	Retourne si une autre requête HTTP peut utiliser la même instance du handler. L'instance sera alors mise en cache. Cette propriété peut par exemple être mise à **false** si le traitement personnalisé n'est pas thread-safe ou bien si l'on ne souhaite pas mettre en place un pool. Elle peut être mise à **true** également pour signifier que le gestionnaire supporte un mode asynchrone.
ProcessRequest	Méthode dans laquelle a lieu le traitement personnalisé

ASWAHandler a la forme suivante :

```
namespace aswaHandler
{
    …
    public class aswaHandler:IHttpHandler
    {
        …
        bool System.Web.IHttpHandler.IsReusable {
            get { return false;}
        }
        void IHttpHandler.ProcessRequest(HttpContext context)
        {
            …
            cheminAswaHandler =  (string)(System.Configuration.ConfigurationSettings.
                                        AppSettings.Get("AswaHandlerPath"));
            cheminAswa = (string)(System.Configuration.ConfigurationSettings.
                                    AppSettings.Get("AswaPath"));
            string[] splitUrl = ParseUrl(context);
            if ( WSExist(splitUrl[0])==false) return;
            AppelWS(context,splitUrl);
        }
        …
    }
```

La méthode de traitement *ProcessRequest()* fait appel à trois fonctions principales :

- La fonction *ParseUrl()* : Cette fonction joue le rôle d'analyseur de requête, elle reçoit une requête de la forme :
  http://Serveur/aswa/WebService.aswa?method=NomMéthode,param=param1,...
  Exemple :
  http://Piano/aswa/Calculatrice.aswa?method=Add,param=10,param=-23

Après l'analyse de cette requête, celle-ci est transformée en un tableau de string contenant :
- Le nom du Web service. Exemple : Calculatrice.asmx
- La méthode Web qu'il faut appeler. Exemple : Add
- La liste des paramètres. Exemple : 10, -23

- La fonction *WSExist()* dont le rôle principal est le déploiement dynamique des applications actives sous forme de Web services.
  En effet, cette fonction reçoit le tableau contenant le nom du Web service, le nom de la méthode et les paramètres.
  - Elle commence par vérifier l'existence du Web service sur la machine elle-même. Si ce service n'existe pas, une requête est envoyée au nœud de contrôle. Ce dernier retourne l'emplacement de ce service permettant ainsi le téléchargement de ce service.
  - L'étape suivante consiste à vérifier l'existence du proxy. Si ce dernier n'existe pas, il est créé automatiquement pour permettre l'appel des différentes méthodes Web du service.
- Enfin, une fois le proxy est installé, la méthode Web est appelée de façon sychrone à l'aide de la fonction *AppelWS()* et en utilisant la « reflection ».

La Figure 4.3 illustre le fonctionnement du gestionnaire ASWA.

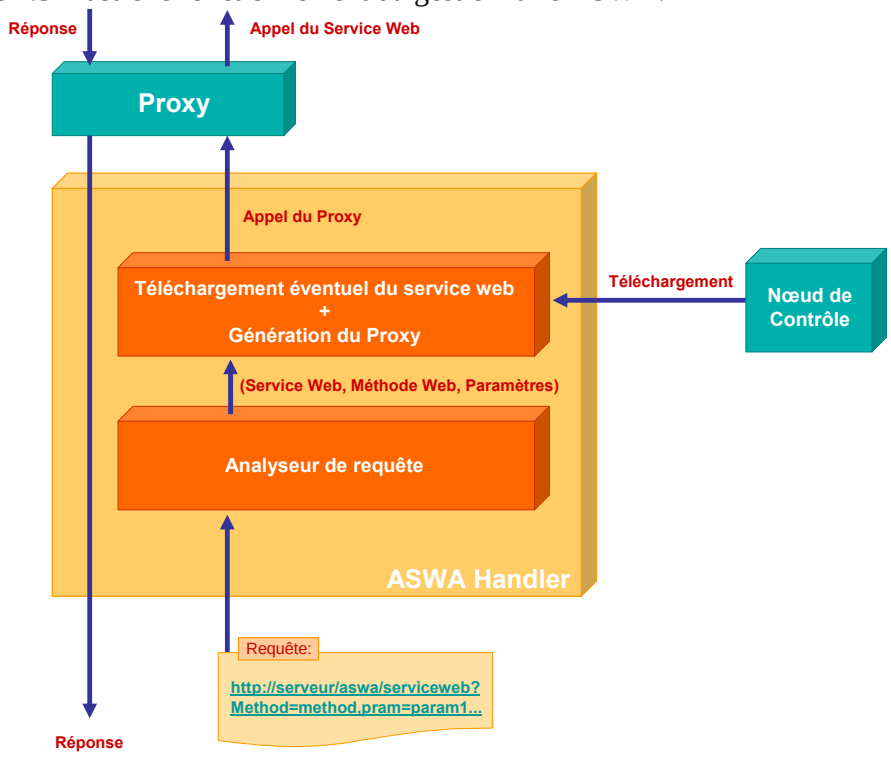

Figure 4.3 - Architecture de ASWA Handler

Avant de pouvoir utiliser ce gestionnaire, plusieurs tâches de configuration sont requises:
- Créer une application Web sous le nom : aswa. Ceci résultera en la création d'un répertoire qui contiendra toutes les applications actives sous forme de Web services.
- Créer une application Web sous le nom : aswa2. Ceci résultera en la création d'un répertoire qui contiendra les proxies de tous les Web services.
- Configurer IIS de façon à dérouter les requêtes portant sur les fichiers d'extension « .aswa » vers ASP.NET.
- Personnaliser le fonctionnement du module ASP.NET suite à la réception de ce type de fichier.

Les étapes à suivre pour effectuer les 2 dernières tâches sont les suivantes :
- Lancer le gestionnaire du service IIS à l'aide de la commande *inetmgr* (Figure 4.4) pour pouvoir modifier les propriétés de l'application aswa déjà créée (Figure 4.5)

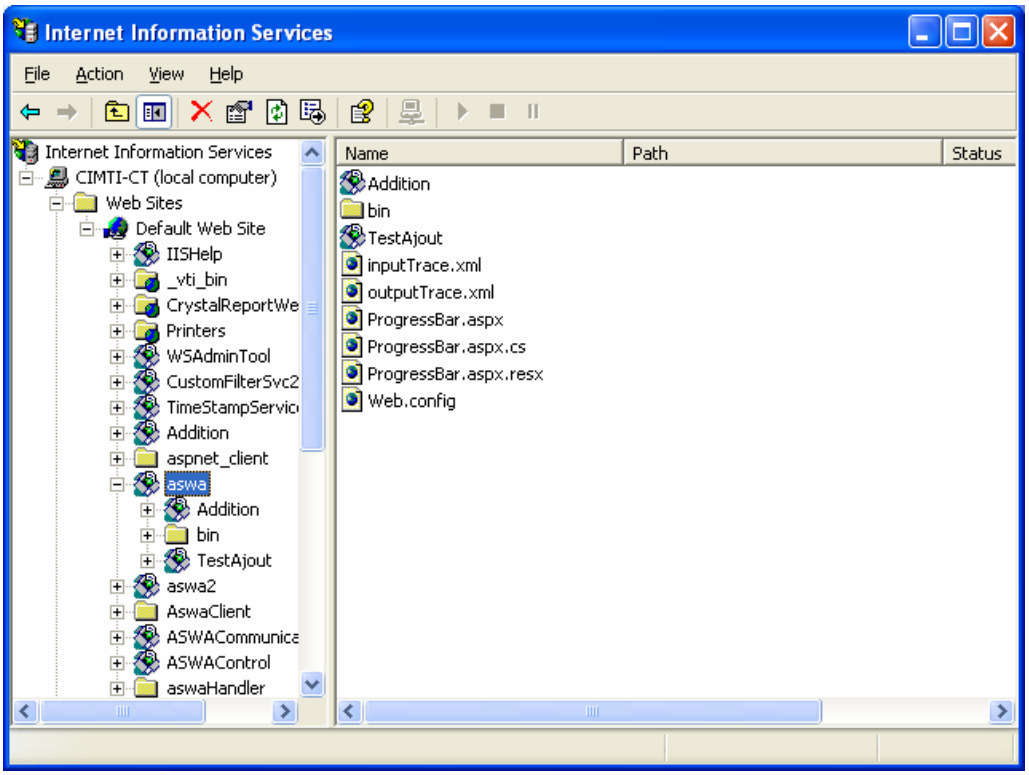

Figure 4.4 - Gestionnaire du serveur Web IIS

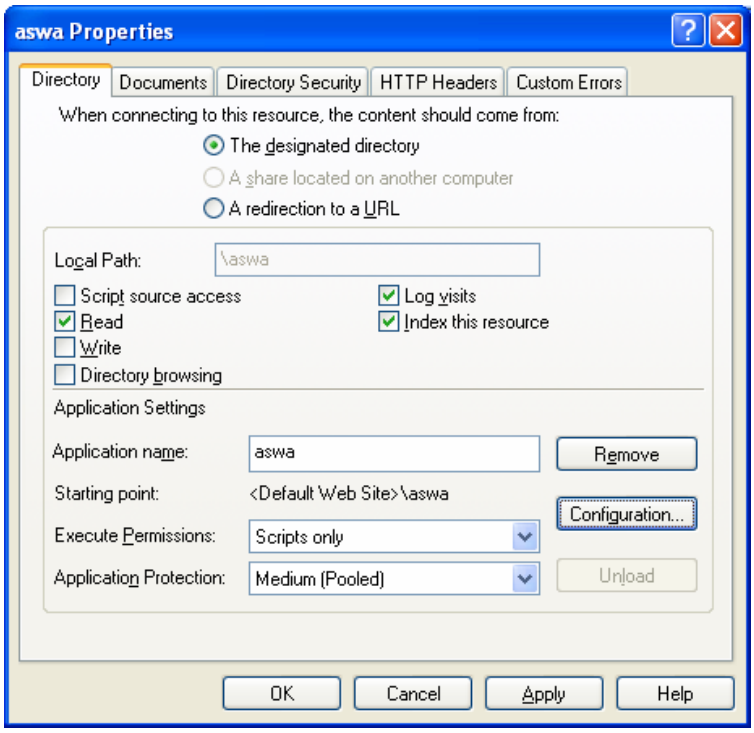

Figure 4.5 - Menu "properties" de l'application Web aswa

- Dérouter les fichiers d'extension «.aswa» vers ASP.NET (Figure 4.6) en associant un nouveau gestionnaire HTTP à ces fichiers (Figure 4.7). Pour cela il faut saisir les données suivantes:
  - Dans « Executable » saisir le chemin complet vers le filtre ASP.NET : aspnet_isapi.dll, dont l'adresse officielle pour le framework 1.1 est:

    <lecteur>:\%windows%\Microsoft.NET\Framework\v1.1.4322\aspnet_isapi.dll
  - Dans la case *extension* taper « .aswa »

- Décocher l'option « Check that file exists », vu que L'URL d'une requête aswa contient le nom du Web service en question auquel on a ajouté l'extension .aswa. Par conséquent le filtre ASP.NET ne doit pas vérifier l'existence du fichier avant d'appeler le gestionnaire HTTP.

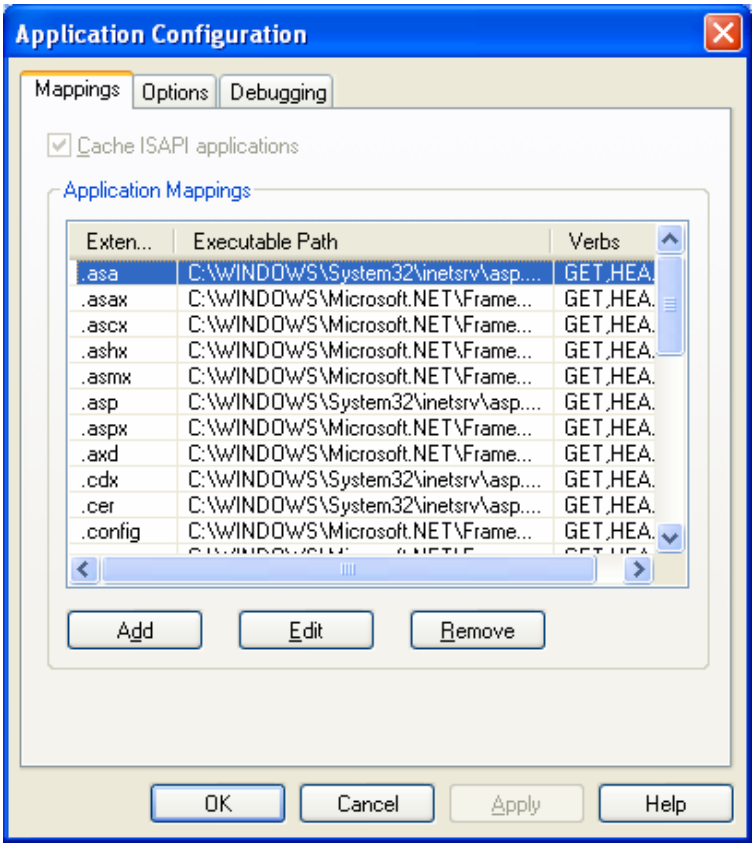

Figure 4.6 - Configuration d'un gestionnaire HTTP

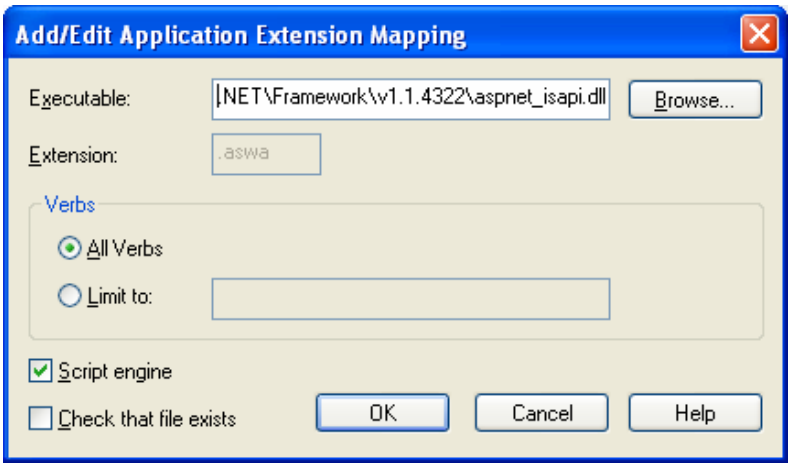

Figure 4.7 - Définition d'un nouveau gestionnaire HTTP

- Compléter enfin le fichier de configuration *web.config* de l'application *aswa* en y ajoutant la section <*httpHandlers*>. Cette section permettra d'associer le type de fichier ayant une extension .aswa au handler HTTP aswaHandler que nous avons implémenté.

```
<httpHandlers>
    <add verb="GET" path="*.aswa" type="aswaHandler.aswaHandler, aswaHandler" validate="false"/>
</httpHandlers>
```

Cette section peut se trouver dans *web.config* ou bien dans *machine.config* si l'on souhaite que ses effets s'appliquent à tous les sites ASP d'un serveur. Sa syntaxe est la suivante:

```
<httpHandlers>
    <add verb="liste de verbes http" path="chemin d'une url"
         type="classe implémentant le httphandler" validate="true|false" />
</httpHandlers>
```

Le tableau suivant explique ses différents attributs :

verb	Liste de verbes HTTP séparés par une virgule. On peut utiliser * pour mapper le handler pour tous les verbes HTTP. Par exemple: GET,POST,HEAD
path	Format de l'URL à laquelle s'applique le mappage. On peut utiliser le joker *. Par exemple: *.aspx DownloadFile.down
type	Désigne la classe et l'assemblage qui implémentent le handler. La syntaxe a la forme suivante : « namespace.classe, assemblage »
validate	Spécifie à quel moment le handler est chargé. **False** signifie que ASP.NET charge le handler uniquement au moment où c'est nécessaire « lazzy loading ». **True** implique un chargement au moment où le fichier de configuration est traité pour la première fois.

4.3.2. Le module d'interception de paquets et le service de routage : ASWA Transport

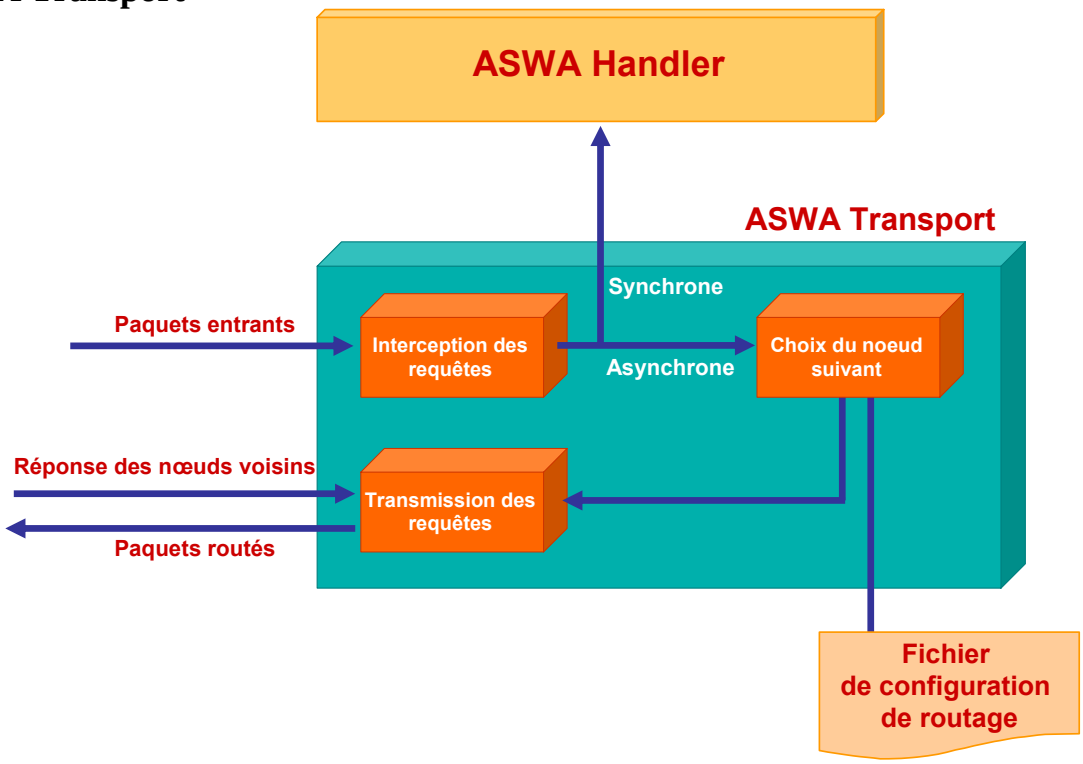

Figure 4.8 - Interception des paquets et routage

ASWA Transport est un Web service qui assure les fonctions suivantes (Figure 4.8) :

- La communication avec le Noeud de Contrôle : Grâce à ce service, l'administrateur d'un nœud actif peut, à n'importe quel instant, connaître l'adresse IP du nœud de contrôle qui lui est attribué, et s'attribuer un autre au besoin.
- La réception des paquets provenant d'un client : Ce service joue le rôle d'interface entre un client et une application active. Il permet au nœud actif de recevoir une requête ASWA encapsulée dans un message SOAP (voir Figure 4.9) contenant entre autre le mode de routage que le service de routage devra appliquer à ce message.

- Le service de routage : Il reçoit les messages SOAP du service d'interception de paquets. Il choisit le nœud suivant auquel il envoie la requête encapsulée dans un message SOAP, sachant que ce choix dépend du mode de routage encapsulé dans la requête. L'envoi se fait, après la mise à jour de l'entête de routage, à l'aide d'un appel asynchrone. Cette requête est alors servie localement en appelant le gestionnaire ASWAHandler à l'aide d'une requête synchrone. Une fois les réponses reçues du nœud courant et du nœud suivant, un message SOAP, encapsulant la concaténation de ces réponses, est envoyé au nœud précédent.

L'exemple de la Figure 4.9 fournit un message SOAP émis par un client ASP.NET et reçu par le module d'interception des paquets:

```
<soap:Envelope
    xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xmlns:xsd=http://www.w3.org/2001/XMLSchema
    xmlns:wsa=http://schemas.xmlsoap.org/ws/2004/03/addressing
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
  <soap:Header>
    <wsa:Action>http://tempuri.org/GetService</wsa:Action>
    <wsa:MessageID>uuid:88acbea1-5ef1-4b01-9452-0537dfe54222</wsa:MessageID>
    <wsa:ReplyTo>
      <wsa:Address>
          http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
      </wsa:Address>
    </wsa:ReplyTo>
      <wsa:To>http://192.168.2.41/ASWACommunicator/ASWATransport.asmx</wsa:To>
      <wsse:Security>
          <wsu:Timestamp wsu:Id="Timestamp-089626d5-99bb-4d4c-b03a-b6be9a686179">
              <wsu:Created>2005-06-29T13:22:43Z</wsu:Created>
              <wsu:Expires>2005-06-29T13:27:43Z</wsu:Expires>
          </wsu:Timestamp>
      </wsse:Security>
      <!--Entête du paquet actif ASWA-->
      <aswa:ASWA aswa:ProtocolID="Addition" xmlns:aswa="http://fi.usj.edu.lb/aswa">
          <ProtocolID>Addition</ProtocolID>
          <RoutingMethod> Integrated</RoutingMethod>
          <path xmlns:m="http://schemas.xmlsoap.org/rp/">
              <action>http://schemas.xmlsoap.org/soap/fault</action>
              <!—Adresse destination -->
              <to>http://192.168.2.41/aswa/Addition.asmx</to>
              <!—Adresse source -->
              <from>http://192.168.2.40/ASWACommunicator/PacketGenerator.aspx</from>
              <!— Champ ID -->
              <id>uuid:88acbea1-5ef1-4b01-9452-0537dfe54222</id>
              <!—Champ REF -->
              <relatesTo>uuid:20020802-1403-4351-b623-5dsf35sgs5d6</relatesTo>
          </path>
      </aswa:ASWA>
  </soap:Header>
  <soap:Body>
      <GetService xmlns="http://tempuri.org/">
          http://localhost/aswa/Addition.aswa?method=Add,param=1022,param=4232
      </GetService>
  </soap:Body>
</soap:Envelope>
```

Figure 4.9 - Requête ASWA émise par un client et reçue par le module ASWATransport

4.3.3. Implémentation des modes de routage

C'est le nœud de contrôle qui fournit le chemin traversé par un paquet actif, ceci à l'aide de l'une des deux méthodes suivantes :

- « GetFullRoute(Url[] node) » qui reçoit en paramètre un tableau de type Url qui contient l'adresse du nœud source et l'adresse du nœud destination et retourne ce même tableau en le complétant par les URLs des nœuds intermédiaires par lesquels le paquet doit passer pour atteindre sa destination.
- « GetRoute(Url[] node) » qui reçoit comme paramètre un tableau de type Url qui contient l'adresse du nœud source et l'adresse du nœud destination et retourne l'url du premier nœud intermédiaire que le paquet doit traverser.

Dans ce paragraphe nous allons décrire comment nous avons implémenté les trois modes de routage supporté par notre plateforme ASWA en se basant sur ces deux fonctions, sachant que nous avons déjà détaillé leur spécification précédemment (section 3.5.5.2). Ces trois modes de routage sont :

- Le mode intégré.
- Le mode progressif.
- Le mode séquentiel.

4.3.3.1. Le mode intégré

Dans ce mode, le nœud actif source qui reçoit un paquet, demande au nœud de contrôle, le chemin complet permettant au paquet reçu d'atteindre sa destination. Cette demande est effectuée en appelant la méthode « GetFullRoute» auprès du nœud de contrôle en fournissant comme paramètre le nœud source et le nœud destination. Un nouveau paquet sera construit par le nœud de contrôle et qui contiendra le chemin complet selon la définition de la table de routage du nœud de contrôle. Ce chemin complet sera ajouté à l'entête du message SOAP partant du nœud actif source. Ainsi chaque nœud intermédiaire, en recevant ce message SOAP, va retirer son adresse de l'entête et ajouter un acquittement de réception jusqu'à l'arrivée du message à sa destination finale. Notons qu'un chemin de retour peut être imposé.

L'entête de routage du message SOAP dans un nœud intermédiaire est décrite dans la Figure 4.10. Il faut surtout remarquer la présence de la balise <fwd>. Notons que le nœud source reçoit un message SOAP sans cette balise (voir Figure 4.9) et qu'au noeud destination, cette balise sera vide, ne contenant donc aucune balise <via>.

```
<RoutingMethod>Integrated</RoutingMethod>
<path xmlns:m="http://schemas.xmlsoap.org/rp/">
        <action>http://schemas.xmlsoap.org/soap/fault</action>
        <!—Adresse destination -->
        <to>http://orgue/aswa/Addition.asmx</to>
        <!—Adresse source -->
        <from>http://piano/ASWACommunicator/PacketGenerator.aspx</from>
        <fwd>
            <via>http://flute/aswa/Addition.asmx</via>
            <via>http://violon/aswa/Addition.asmx</via>
        </fwd>
        <!— Champ ID -->
        <id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</id>
        <!—Champ REF -->
        <relatesTo>uuid:20020802-1403-4351-b623-5dsf35sgs5d6</relatesTo>
</path>
```

Figure 4.10 Entête de routage dans un nœud intermédiaire en mode intégré

4.3.3.2. Le mode progressif

Dans ce mode, à la réception d'un paquet, chaque nœud actif envoie au nœud de contrôle, une requête de demande du nœud suivant sur le chemin à suivre, et ceci par l'appel de la méthode

« GetRoute » en fournissant comme paramètre le nœud source et le nœud destination. Le nœud de contrôle retournera alors l'adresse du nœud suivant sur le chemin à parcourir par le paquet actif et ceci en consultant sa table de routage. Cette adresse sera ajoutée à l'entête du message SOAP. Dans ce cas, la balise « path » du paquet construit au niveau de chaque nœud actif contiendra une seule adresse intermédiaire qui est celle du nœud suivant. Ainsi chaque nœud intermédiaire, en recevant le message SOAP, va modifier l'entête de ce message en remplaçant son adresse par l'adresse du nœud suivant jusqu'à l'arrivée du message à la destination finale.

Par exemple, le message SOAP reçu par le nœud « flute » contient l'entête de routage décrit dans la Figure 4.11 :

```
<RoutingMethod>Progressive</RoutingMethod>
<path xmlns:m="http://schemas.xmlsoap.org/rp/">
        <action>http://schemas.xmlsoap.org/soap/fault</action>
        <!—Adresse destination -->
        <to>http://orgue/aswa/Addition.asmx</to>
        <!—Adresse source -->
        <from>http://piano/ASWACommunicator/PacketGenerator.aspx</from>
        <fwd>
            <via>
                http://flute/aswa/Addition.asmx
            </via>
        </fwd>
        <!— Champ ID -->
        <id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</id>
        <!—Champ REF -->
        <relatesTo>uuid:20020802-1403-4351-b623-5dsf35sgs5d6</relatesTo>
</path>
```

Figure 4.11 – Entête de routage en mode progressif à l'entrée du nœud « flute »

En sortant du nœud « flute », le message SOAP contiendra l'entête de routage décrit dans la Figure 4.12 :

```
<RoutingMethod>Progressive</RoutingMethod>
<path xmlns:m="http://schemas.xmlsoap.org/rp/">
        <action>http://schemas.xmlsoap.org/soap/fault</action>
        <!—Adresse destination -->
        <to>http://orgue/aswa/Addition.asmx</to>
        <!—Adresse source -->
        <from>http://piano/ASWACommunicator/PacketGenerator.aspx</from>
        <fwd>
            <via>
                http://violon/aswa/Addition.asmx
            </via>
        </fwd>
        <!— Champ ID -->
        <id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</id>
        <!—Champ REF -->
        <relatesTo>uuid:20020802-1403-4351-b623-5dsf35sgs5d6</relatesTo>
</path>
```

Figure 4.12 – Entête de routage en mode progressif à la sortie du nœud « flute »

4.3.3.3. Le mode séquentiel

Dans le routage en mode séquentiel, chaque nœud actif route le paquet en consultant sa table de routage définie selon les spécifications WS-Referal.

Dans ce mode, chaque nœud actif, lors de la réception d'un paquet, cherche dans son fichier de routage « Referral Cache » le nœud suivant vers lequel il envoie le paquet. Ce fichier est

mis a jour dynamiquement à travers un Web service.

L'entête de routage du message SOAP ne contient pas la balise <fwd> et il est décrit dans la Figure 4.13 :

```
<RoutingMethod>Sequential</RoutingMethod>
<path xmlns:m="http://schemas.xmlsoap.org/rp/">
        <action>http://schemas.xmlsoap.org/soap/fault</action>
        <!—Adresse destination -->
        <to>http://orgue/aswa/Addition.asmx</to>
        <!—Adresse source -->
        <from>http://piano/ASWACommunicator/PacketGenerator.aspx</from>
        <!— Champ ID -->
        <id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</id>
        <!—Champ REF -->
        <relatesTo>uuid:20020802-1403-4351-b623-5dsf35sgs5d6</relatesTo>
</path>
```

Figure 4.13 – Entête de routage en mode séquentiel

4.4. GESTION GRAPHIQUE D'UN NŒUD ACTIF

Une interface graphique a été implémentée pour faciliter les tâches d'administration et de gestion d'un nœud actif. Cette interface permet:

- A l'administrateur d'un nœud actif de choisir le domaine auquel appartiendra ce nœud, en lui attribuant le noeud de contrôle de ce domaine (Figure 4.14 et Figure 4.15).

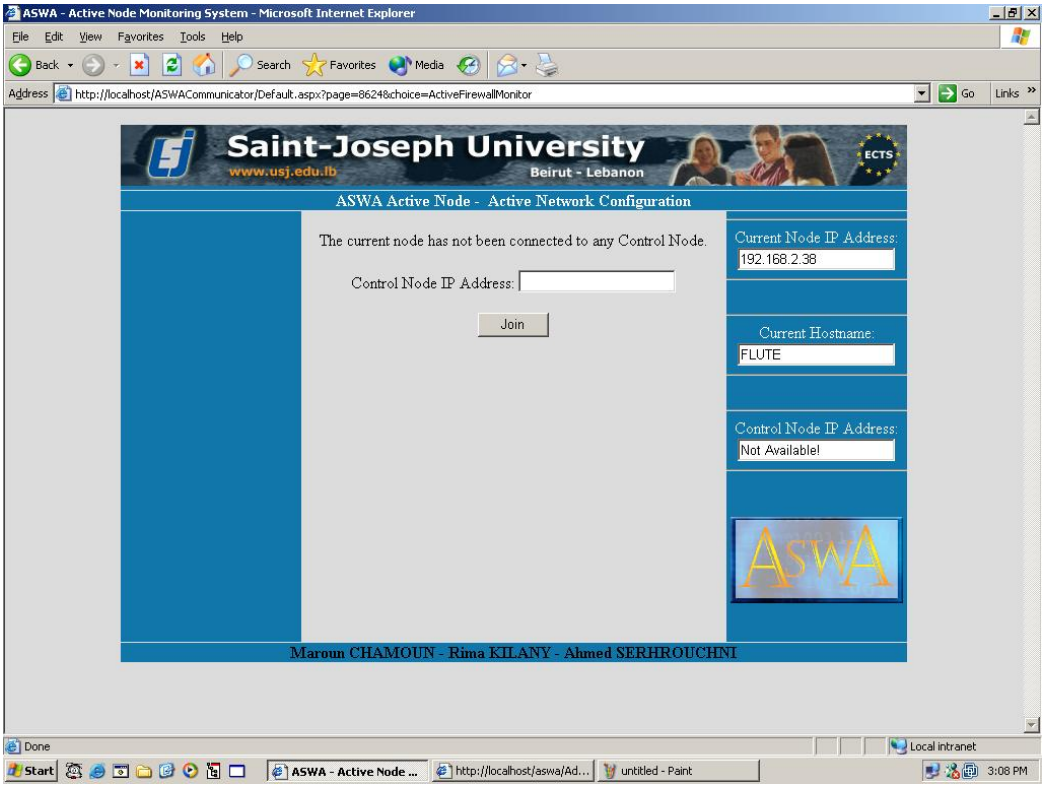

Figure 4.14 – Interface de configuration du noeud de contrôle

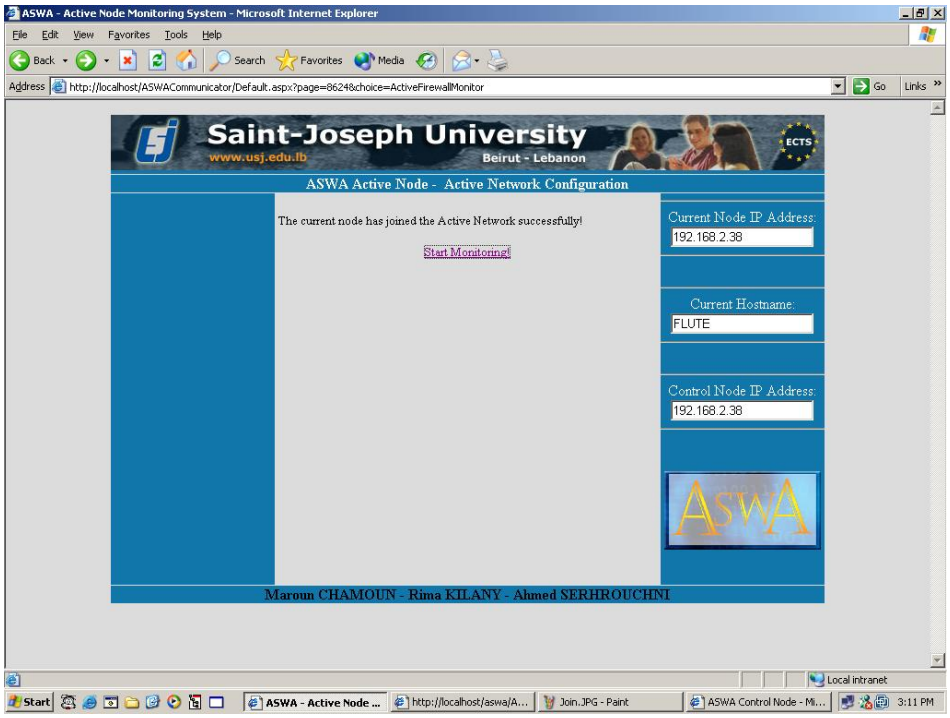

Figure 4.15 - Définition d'un noeud de contrôle

- De lancer un service actif en précisant le mode de routage des paquets et le chemin à parcourir par ces paquets à l'aide d'un générateur de paquet (Figure 4.16).

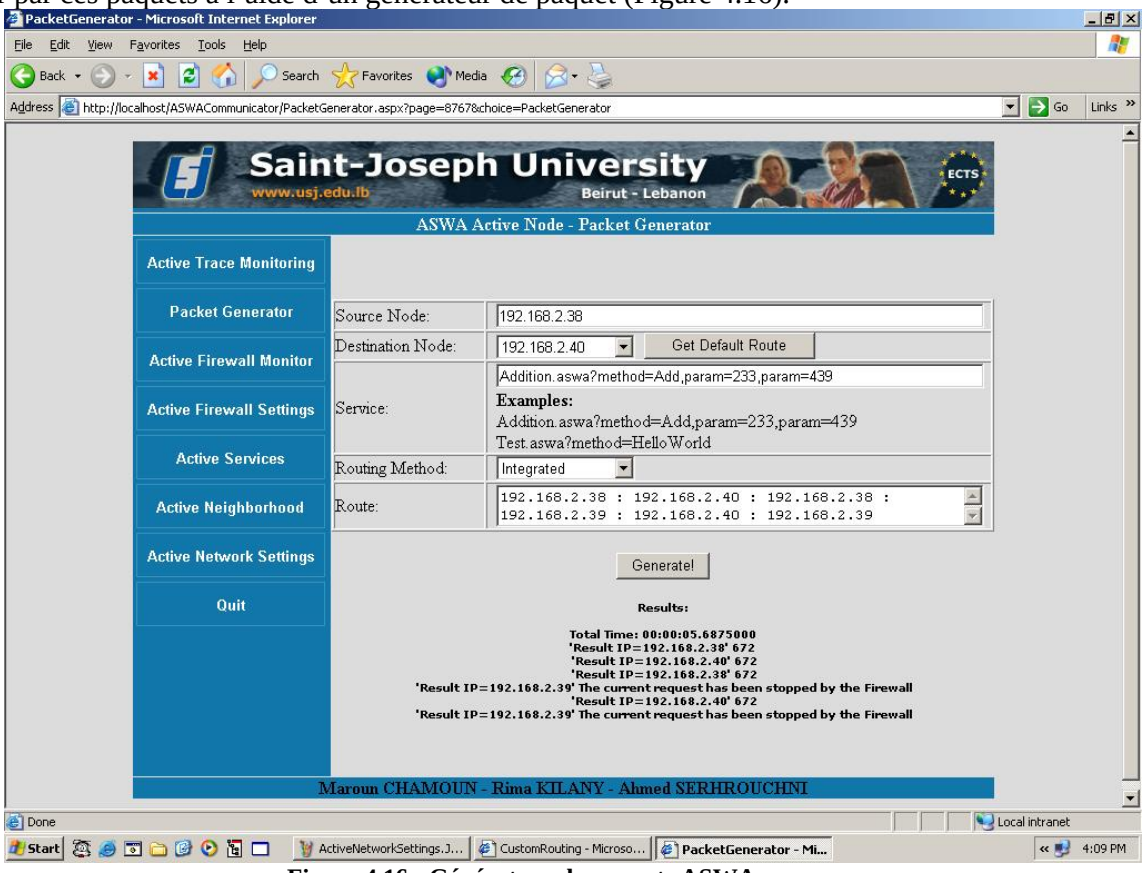

Figure 4.16 - Générateur de paquets ASWA

- De consulter la liste des services disponibles sur les nœuds actifs appartenant au même domaine et accessible par le nœud actif courant (Figure 4.17).

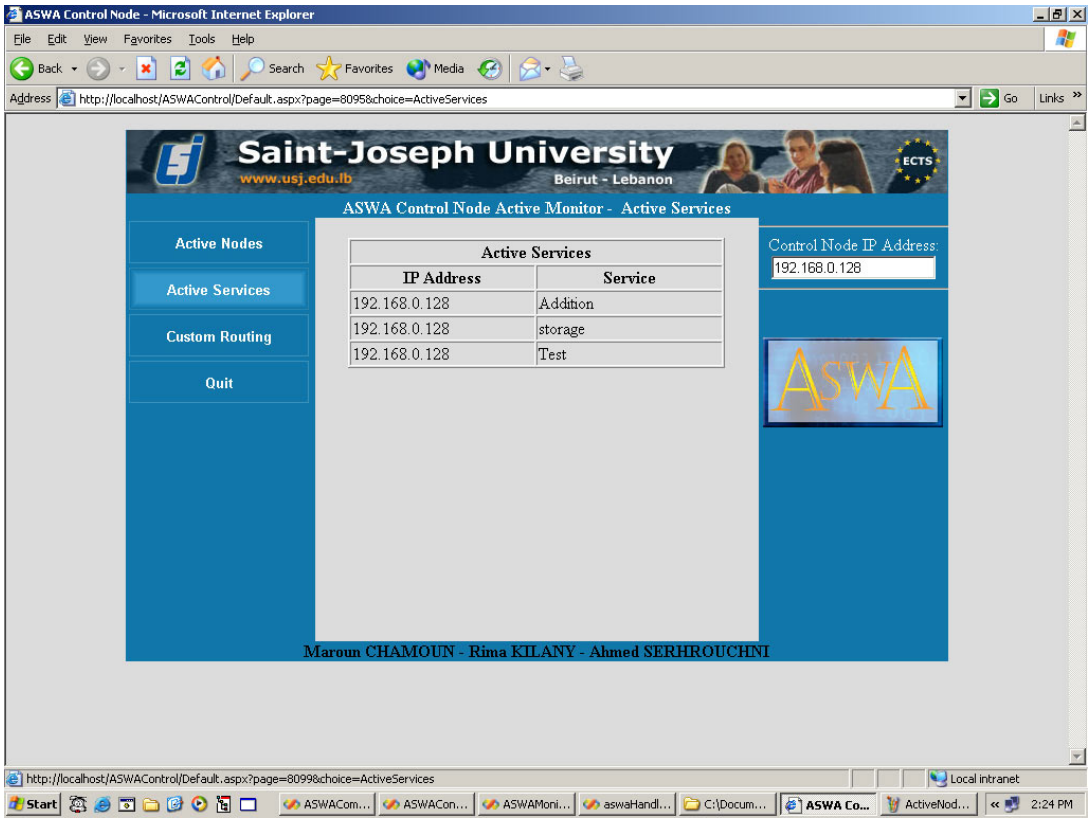

Figure 4.17 - Liste des services accessibles par le noeud actif

- De consulter et de modifier l'adresse du nœud de contrôle et celle du nœud voisin. Un voisin est défini dans le cas du mode de routage séquentiel comme étant le nœud suivant (Figure 4.18).

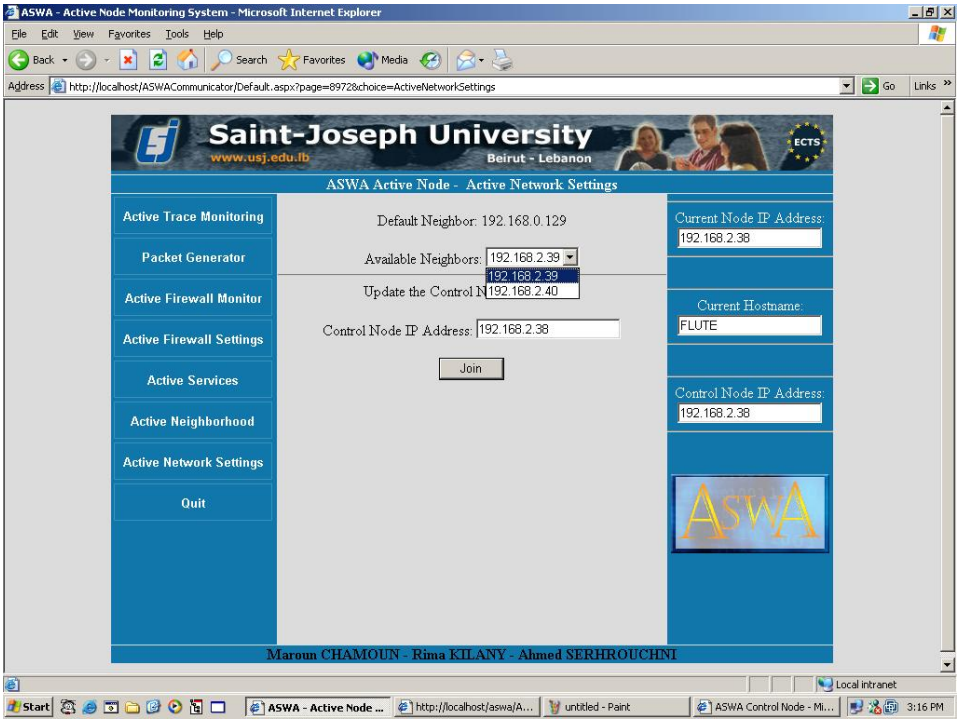

Figure 4.18 - Ecran de paramétrage

- De surveiller le nœud actif en affichant des messages tout au long du passage du paquet actif à travers les différents modules de l'environnement d'exécution (Figure 4.19). Ces messages nous permettent :

- De consulter le contenu de la requête tels que le corps du message, le mode de routage et le chemin à parcourir par le paquet.
- De vérifier si le Web service requis est installé sur la machine. Sinon, toute la procédure de demande du service à partir du nœud de contrôle jusqu'à l'installation sur la machine est affichée.
- De vérifier si le proxy existe. Sinon, toute la procédure de création du proxy ainsi que les méthodes Web disponibles sont affichées.
- De surveiller éventuellement le passage du paquet par un pare-feu et le traitement qui lui est appliqué (paquet permis ou refusé).

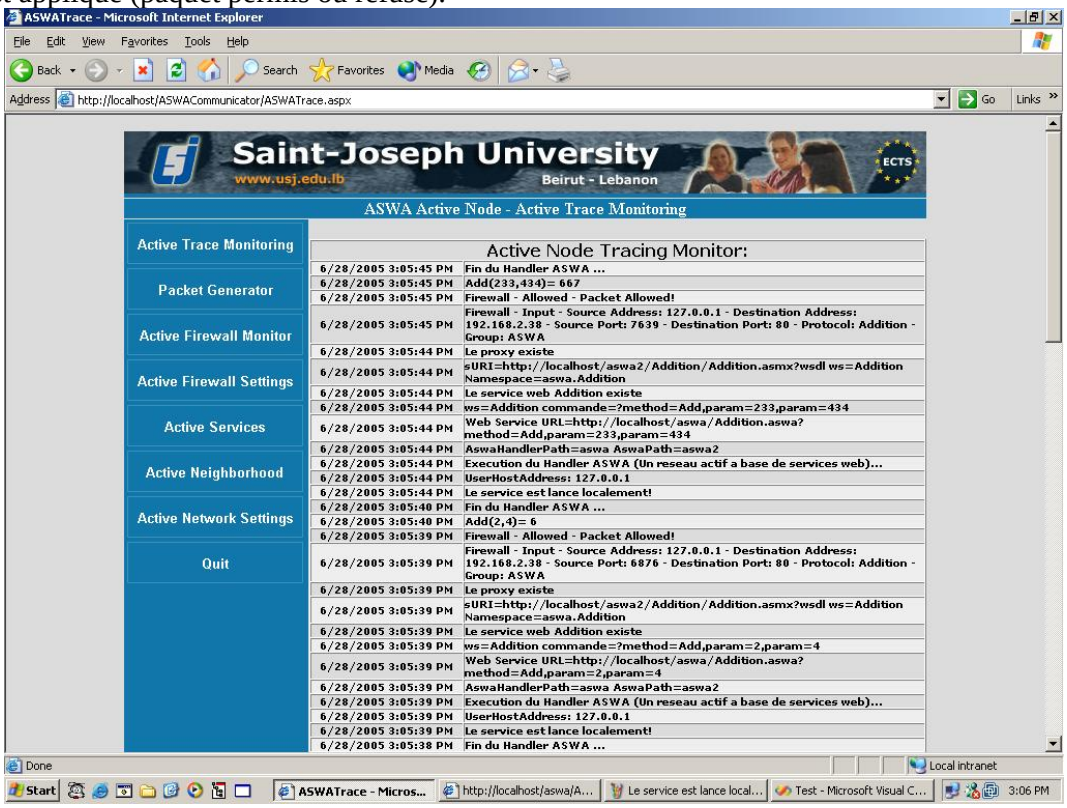

Figure 4.19 - Ecran de surveillance

4.5. NŒUD DE CONTROLE

Un nœud de contrôle est un nœud actif qui assure de plus la gestion et le contrôle des nœuds actifs définis dans son domaine. Il assure les tâches suivantes :

- Gestion des nœuds appartenant à son domaine de contrôle (ajouter, enlever, fournir la liste de tous les nœuds,…).
- Gestion des services actifs :
  - Ajouter un service avec sa référence (Adresse IP du nœud à partir duquel il faut télécharger le service) à la table des services du nœud de contrôle.
  - Mettre à jour l'annuaire du nœud d'administration à partir de la table de services locale, à l'aide d'une requête UDDI.
  - Enlever un service de la table des services.
  - Fournir la liste des services disponibles dans son domaine de contrôle.
  - Fournir à un nœud actif l'adresse IP du nœud contenant un service bien déterminé. On distingue deux cas:
    - Le service est fourni par un nœud appartenant au même domaine, le nœud de contrôle peut répondre immédiatement.
    - Le service est fourni par un nœud appartenant à un autre domaine, le nœud de contrôle émet alors une requête UDDI vers le nœud d'administration pour la recherche du service dans son annuaire.

- Gestion de la table de routage (Figure 4.20) :
  - Gestion de la base des données de toutes les routes définies dans le domaine.
  - Ajouter un nœud à un chemin.
  - Enlever un nœud d'un chemin.
  - Fournir un chemin complet.
  - Fournir le nœud suivant dans un chemin en fonction du mode de routage et du nœud courant.

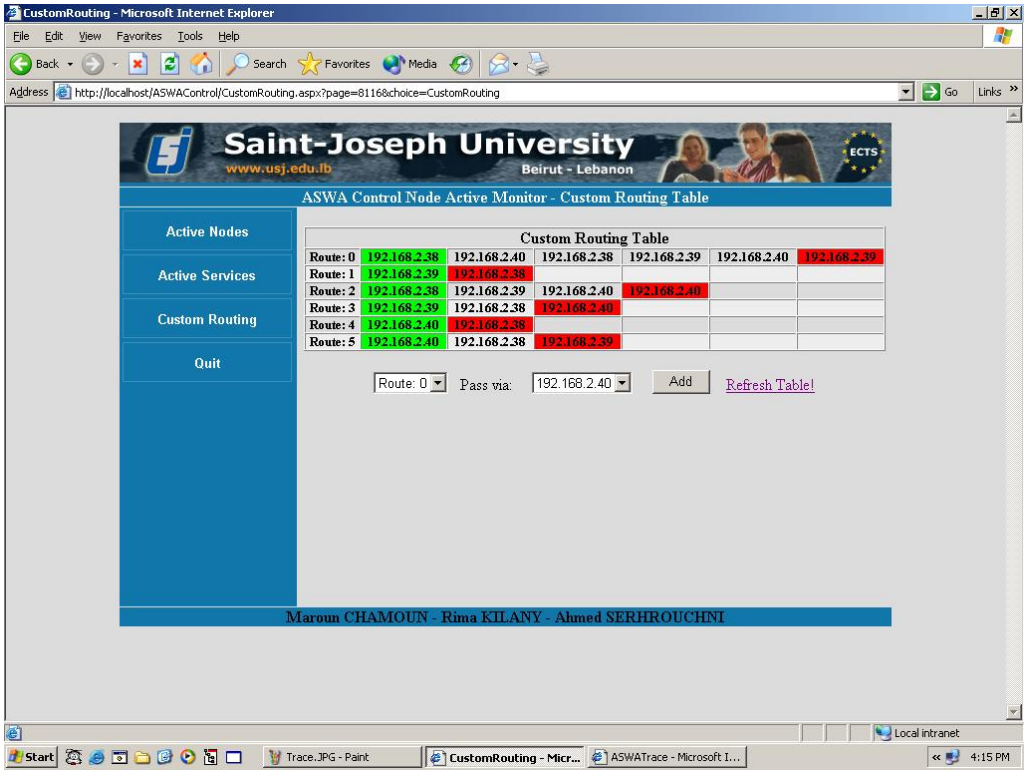

Figure 4.20 – Interface graphique pour la gestion de la table de routage

4.6. NŒUD D'ADMINISTRATION

Le nœud d'administration est un nœud de contrôle qui a de plus deux rôles principaux :

– Il accepte l'enregistrement par des requêtes UDDI de tous les services supportés par les nœuds de contrôle. Ceci nécessite l'utilisation d'un annuaire.
– Il fonctionne en tant qu'autorité de certification dont la clé publique doit être distribuée par face à face à tous les nœuds actifs du réseau voulant recevoir du code authentifié. Ce qui nécessite l'utilisation d'une infrastructure à clé publique (PKI).

Notre solution consiste à considérer un nœud d'administration comme étant une plateforme *Windows 2003 Server* sur laquelle sont installés les deux services Windows : *PKI* et *UDDI*.

4.7. SECURITE DANS ASWA

Les problèmes de sécurité des réseaux actifs sont déjà bien identifiés (voir section 3.6.1). Les paquets actifs peuvent mal utiliser les nœuds actifs, les ressources réseau ainsi que d'autres paquets actifs. De même que les nœuds actifs peuvent mal utiliser les paquets actifs. Protéger donc les nœuds et les paquets dans un environnement aussi flexible est une tâche difficile mais nécessaire. Dans cette section nous décrivons les détails de l'implémentation de la solution que nous avons proposée au niveau de la section 3.6.3, pour sécuriser les transferts et assurer une haute disponibilité des services actifs.

4.7.1. Implémentation de la sécurité dans ASWA

Pour l'implémentation de notre solution, nous avons utilisé la notion de filtres de sécurité (« Security Filter ») du pipeline WSE (section C.5). Chaque filtre pourrait être implémenté pour

supporter un ou plusieurs services de sécurité et être placés à l'entrée et/ou à la sortie du module d'interception de paquets (Figure 4.21) sur tous les nœuds actifs. Il suffirait alors à un administrateur de préparer un fichier de configuration (Figure 4.22) et de le déployer avec les filtres à partir d'un nœud de contrôle sur tous les nœuds actifs du domaine. Le fichier de configuration indique au module d'interception de paquets les services de sécurité qu'il faut appliquer à chaque paquet traversant le nœud courant. Cette solution est transparente pour l'utilisateur final puisque c'est l'EE du nœud actif qui s'occupe d'appliquer les services de sécurité.

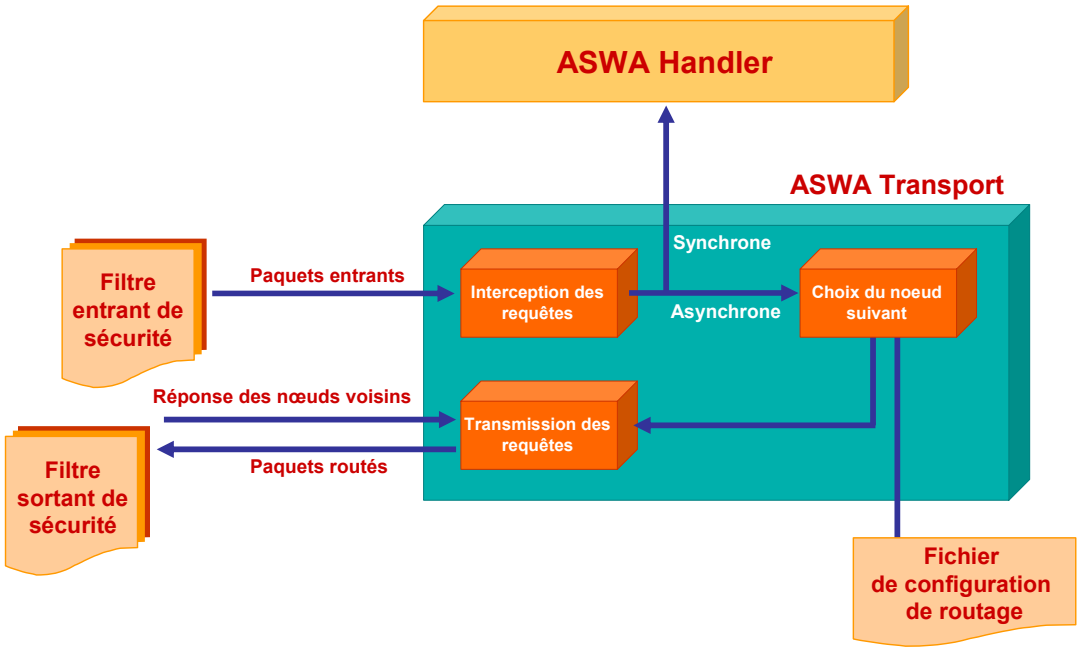

Figure 4.21 - Filtres de sécurité

```
<filters>
    <input>
        <add type="CustomSecurityFilter.CustomSecurityInputFilter, CustomSecurityFilter,
                Version=1.1.1.1, Culture=neutral, PublicKeyToken=817f3074faacffe6" />
    </input>
    <output>
        <add type="CustomSecurityFilter.CustomSecurityOutputFilter, CustomSecurityFilter,
                Version=1.1.1.1, Culture=neutral, PublicKeyToken=817f3074faacffe6" />
    </output>
</filters-->
```

Figure 4.22 - Exemple de définition des filtres dans le fichier de configuration Web.config

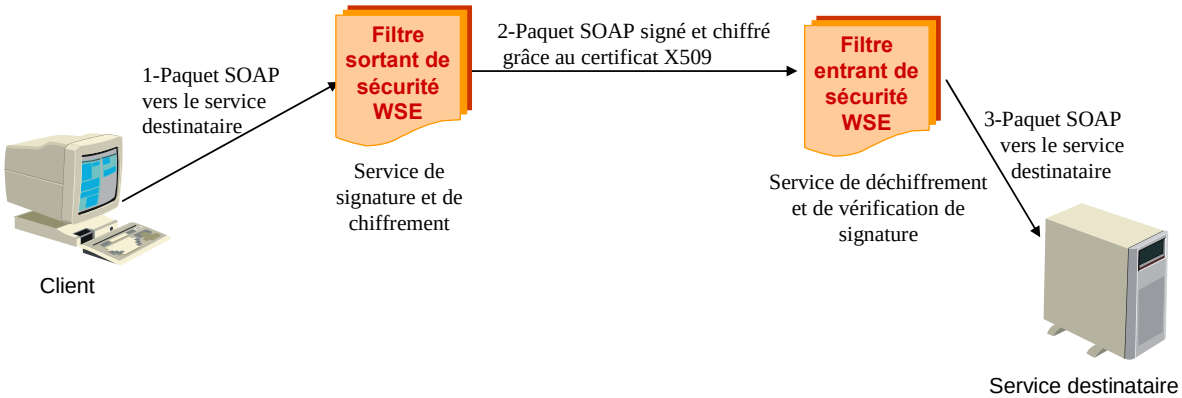

Figure 4.23 - Echange sécurisé des paquets SOAP

L'exemple de la Figure 4.23 illustre l'utilisation des filtres de sécurité, où le paquet SOAP émis en clair et sans signature par le client est intercepté par le filtre sortant qu'on a développé

pour être transmis après lui avoir appliqué les services de signature et de chiffrement correspondant. Le paquet chiffré et signé continue son chemin vers sa destination. Avant d'atteindre l'application réceptrice, le paquet est intercepté par le filtre entrant de sécurité. Ce dernier vérifie la signature et déchiffre le paquet. Par la suite, le paquet atteint l'application réceptrice avec le même format qu'à l'émission. Si la signature n'est pas vérifiée au niveau du filtre entrant de sécurité, une exception est retournée au client. Le même principe s'applique au retour.

Le fichier dans la Figure 4.24 représente un extrait d'un paquet SOAP signé et chiffré dans ASWA, suite au passage par le filtre sortant de la Figure 4.23 (Le paquet complet est présenté dans la section C.5.7 de l'Annexe C):

```
<soap:Header>
    <wsse:Security soap:mustUnderstand="1">
        <wsse:BinarySecurityToken
                                                    …
        </wsse:BinarySecurityToken>
                                                    …
        <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
                                                    …
        </xenc:EncryptedKey>
        <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
                                                    …
      </Signature>
     </wsse:Security>
    </soap:Header>
    <soap:Body wsu:Id="Id-114c8848-78e1-47eb-b5ce-19353c51004c">
        <xenc:EncryptedData Id="EncryptedContent-ed7286ea-667c-4af0-9a82-613f3eebec33"
            Type=http://www.w3.org/2001/04/xmlenc#Content
            xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
        <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc" />
        <xenc:CipherData>
            <xenc:CipherValue>
                YOpl9ot+rvRDsd0cIp4y/vrxQo9dGKbf6EweZeEUV3Vq
                +rOhTY8QbuxmTHDm4rWlmxQ/k/e1Oh+WbXG6GhEb
                7DlmMP8v56Y916DUzsfGdJurkMJvtv3UIFgXxv16MrjU
            </xenc:CipherValue>
        </xenc:CipherData>
    </xenc:EncryptedData>
    </soap:Body>
```

Figure 4.24 - Paquet chiffré et signé

4.7.2. Haute disponibilité des services actifs

Comme déjà décrit dans la section 3.6.3, la haute disponibilité des services actifs est assurée par l'implémentation d'une topologie de réseau transparente en utilisant la technique de redirection qui permet à un paquet de changer sa route en cas de coupure de lien. Cette technique est assurée en utilisant la notion de routeur intermédiaire supportant la spécification WS-Routing proposée par Microsoft WSE.

L'implémentation de la technique de redirection est effectuée par la définition d'un gestionnaire de routage (« Routing Handler »). Un gestionnaire de routage est une instance d'une classse dérivée de la classe « Microsoft.Web.Services.Routing.RoutingHandler », et qui peut redéfinir sa méthode « ProcessRequestMessage ».

```
protected override void ProcessRequestMessage( SoapEnvelope message, Path outPath )
```

Si la fonction « ProcessRequestMessage » n'est pas redéfinie, le Routing Handler s'exécutera simplement en tant que Forwarder. Ceci signifie qu'à la réception d'un message le Routing Handler transmettra le message au nœud suivant tel que spécifié dans son « Referral

Cache ». Sinon, c'est le traitement défini par la fonction « ProcessRequestMessage » qui sera exécuté.

Pour associer un gestionnaire de routage à un service, il suffit d'ajouter les lignes suivantes au fichier Web.config du routeur intermédiaire qui recevra toutes les requêtes des clients:

```
<httpHandlers>
   <add verb="*" path="*.ashx" type="RouterA.RoutingHandler, RouterA" />
</httpHandlers>
```

ashx représente l'extension des requêtes vers les services auxquels sera appliqué le type de redirection voulu, voire le traitement défini au niveau de la méthode « ProcessRequestMessage » de la classe définie: RouterA.RoutingHandler, ou par simple lecture du « referral cache » de ce routeur intermédiaire.

Si un service référencé par un gestionnaire de routage tombe en panne, il suffit de mettre à jour le « referral cache » sur le routeur intermédiaire (voir Figure 3.17), sans changer les tables de routage du réseau actif et conserver ainsi la transparence de la topologie du réseau et assurer une grande flexibilité.

4.8. IMPLEMENTATION DU PARE-FEU DISTRIBUE SOUS ASWA

L'architecture adoptée dans ASWA pour le filtrage distribué consiste à centraliser la création et le contrôle des politiques de sécurité et déployer les règles de politiques définies ainsi que le mécanisme de filtrage.

Cette architecture est en parfaite harmonie avec l'architecture en trois plans des réseaux actifs (Figure 3.2) qu'on a adoptée pour l'implémentation de ASWA. Il suffit donc de permettre la création et la gestion de toutes les règles de politique de sécurité correspondante pour chaque machine ou pour chaque utilisateur sur les noeuds de contrôle, et d'effectuer le filtrage localement sur chaque machine du réseau ASWA.

Le principe même des réseaux actifs donne une très grande flexibilité à cette solution par le fait qu'il permet le déploiement des règles de politiques définies et des mécanismes de filtrage. Ceci a facilité l'implémentation et l'installation d'une interface de gestion et d'administration sur les nœuds de contrôle pour qu'ils fonctionnent en tant que serveurs de politique. Les règles de politiques elles, ont été décrites sous forme de fichiers XML conforme à un « XML schema » déjà défini, pour permettre leur déploiement sous forme de listes de contrôle d'accès (ACL) sur tous les nœuds actifs.

La partie fonctionnelle du pare-feu a été implémentée en se basant sur la notion de filtres personnalisés (« Custom Filter ») de Microsoft WSE, ce qui permet le pré et post traitement des entêtes des messages SOAP de façon systématique suite à l'installation de deux filtres : le « FirewallInputFilter» et le « FirewallOutputFilter».

La Figure 4.25 schématise le fonctionnement d'un pare-feu ASWA:

- A sa réception par l'application active, le paquet SOAP est analysé par le «FirewallInputFilter» du pare-feu. En fait cette analyse n'est autre qu'une comparaison avec l'ACL distribuée et déployée sous forme de fichier XML et qui contient les paramètres de filtrage. Si le paquet peut continuer son chemin il sera envoyé vers l'application active sinon une exception sera signalée.
- A la sortie de l'application active, le paquet est intercepté par le pare-feu, mais cette fois c'est le «FirewallOutputFilter» qui va ajouter à l'entête du paquet SOAP les paramètres spécifiques au pare-feu.

Un filtre personnalisé doit dériver de la classe SoapInputFilter ou SoapOutputFilter et doit obligatoirement redéfinir la méthode virtuelle *ProcessMessage(SoapEnvelope envelope)*. C'est cette méthode qui sera appelée à chaque fois qu'un paquet traverse le filtre.

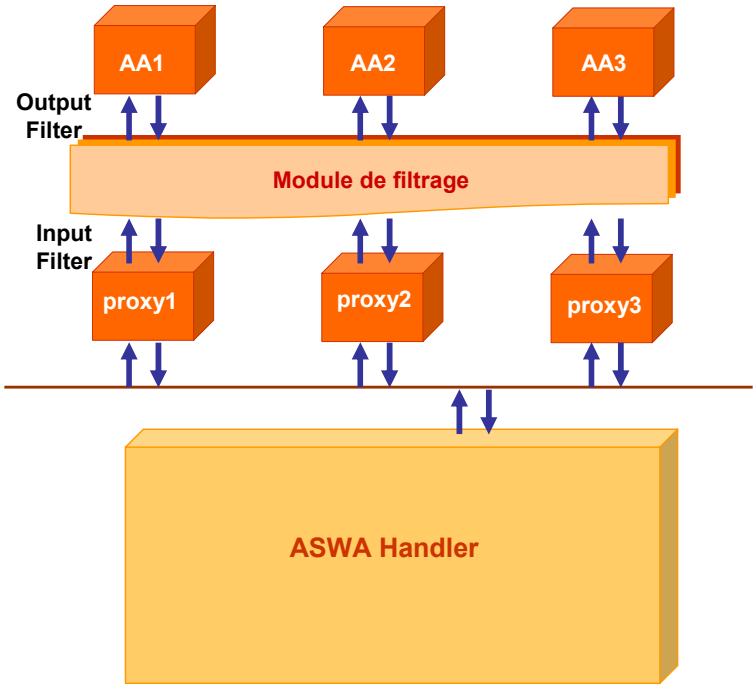

Figure 4.25 - Pare-feu ASWA

4.8.1.1. La classe d'analyse de paquets : FirewallInputFilter

Cette classe est un filtre d'entrée personnalisé. Elle dérive de la classe SoapInputFilter et a la forme suivante :

```
namespace ASWAFilterLibrary
{
	public class FirewallInputFilter : SoapInputFilter
	{
		const string CustomHeaderName = "ASWAFirewallHeader";
		ACL acl=null;
		public FirewallInputFilter()
		{
		}
		public override void ProcessMessage(SoapEnvelope envelope)
		{
			acl=new ACL();
			ACLParam param=new ACLParam();
			XmlElement soapHeader = envelope.Header;
			…
		}
	}
}
```

Le rôle de cette classe est d'extraire les paramètres du pare-feu à partir de l'entête du paquet reçu et de vérifier si ce paquet a le droit de traverser le filtre pour atteindre l'application active.

4.8.1.2. La classe de construction du paquet

Cette classe est un filtre de sortie personnalisé. Elle dérive de la classe SoapOutputFilter et a la forme suivante :

```
namespace ASWAFilterLibrary
{
	public class FirewallOutputFilter : SoapOutputFilter
	{
		public FirewallOutputFilter()
```

```
                {
                }
                public override void ProcessMessage(SoapEnvelope envelope)
                {
                        …
                        XmlElement node =envelope.CreateElement("ASWAFirewallHeader");
                        XmlElement nodeSrcAdd=envelope.CreateElement("srcadd");
                        …
                        nodeSrcAdd.InnerXml=SrcAdd;
                        …
                        node.AppendChild(nodeSrcAdd);
                        …
                        envelope.CreateHeader().AppendChild(node);
                }
        }
}
```

Le rôle de cette classe est d'ajouter les paramètres du pare-feu à l'entête du paquet SOAP sortant d'une application active (voir Figure 3.23).

4.8.1.3. Exemple d'exécution

La capture d'écran de la Figure 4.26 montre un paquet rejeté par le pare-feu.

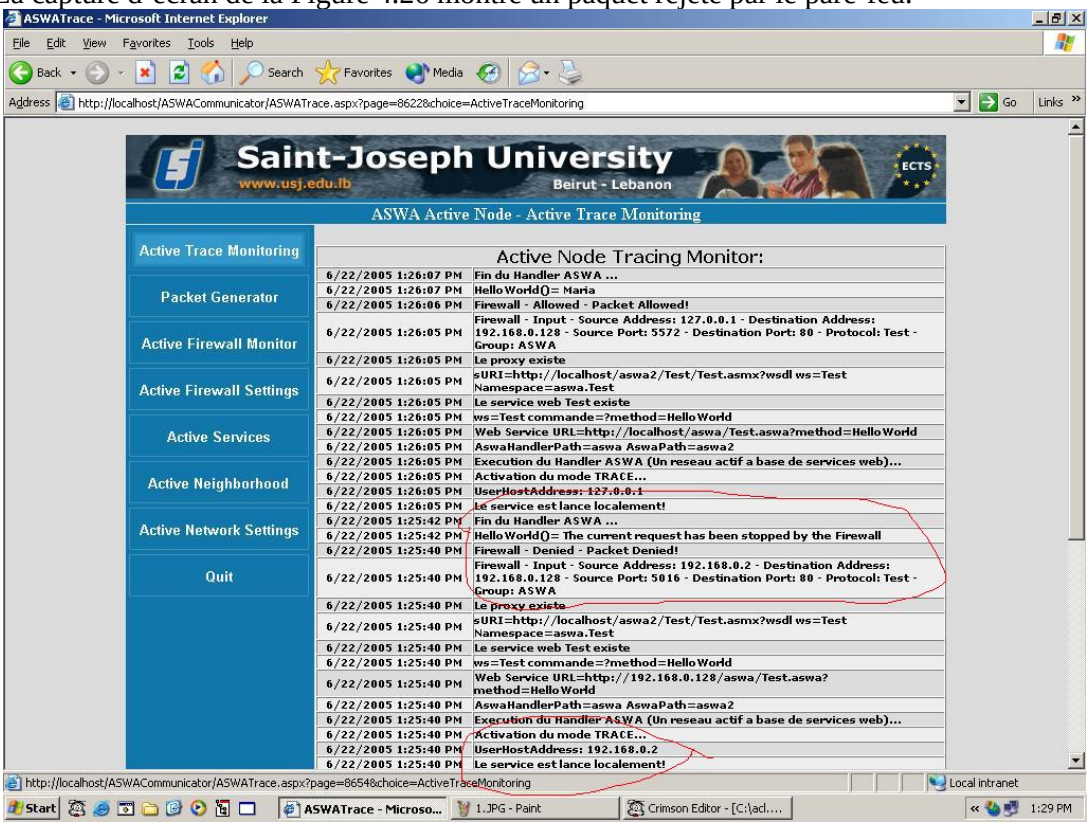

Figure 4.26 - Trace d'un paquet rejeté par le pare-feu

4.8.2. Administration du pare-feu

Une interface graphique a été réalisée pour être installée sur tous les nœuds actifs dans le but de faciliter l'administration du pare-feu.

Un premier écran permet de consulter l'état de chaque paquet reçu par le nœud courant :

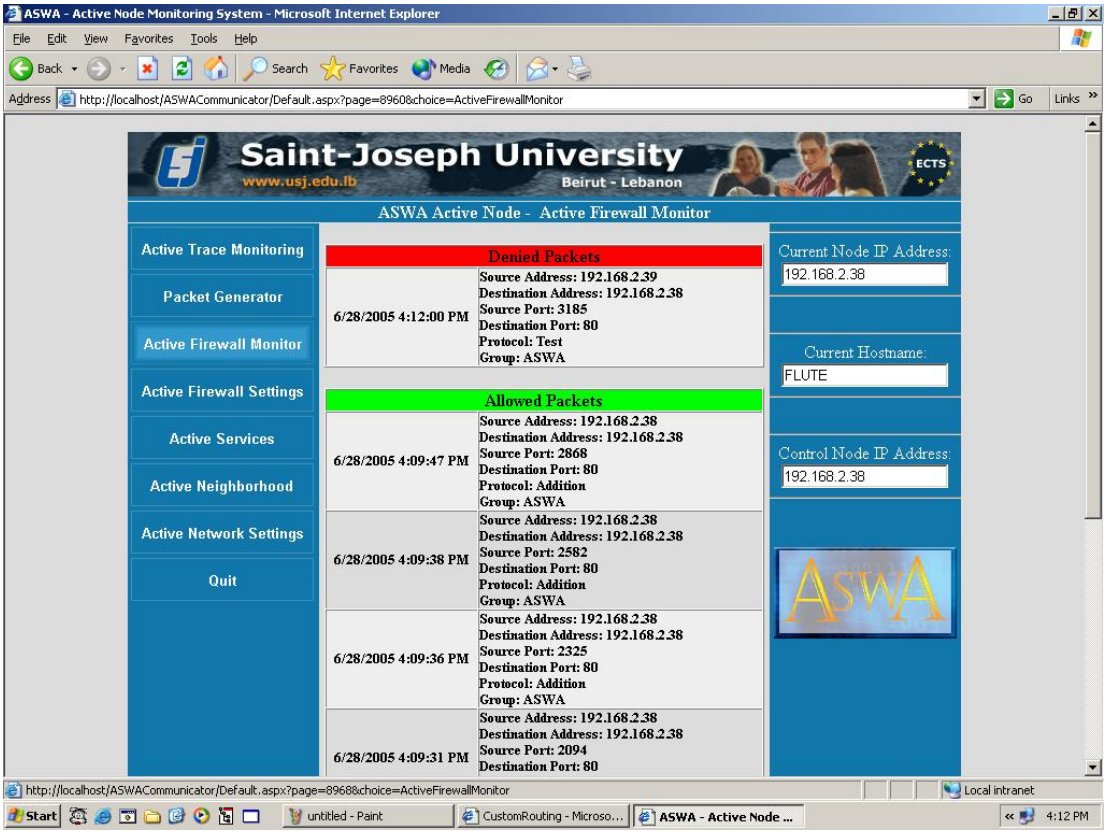

Figure 4.27 - Consultation de l'état des paquets reçus

L'écran suivant permet la consultation et la modification des règles de filtrage du pare-feu, à partir du nœud de contrôle :

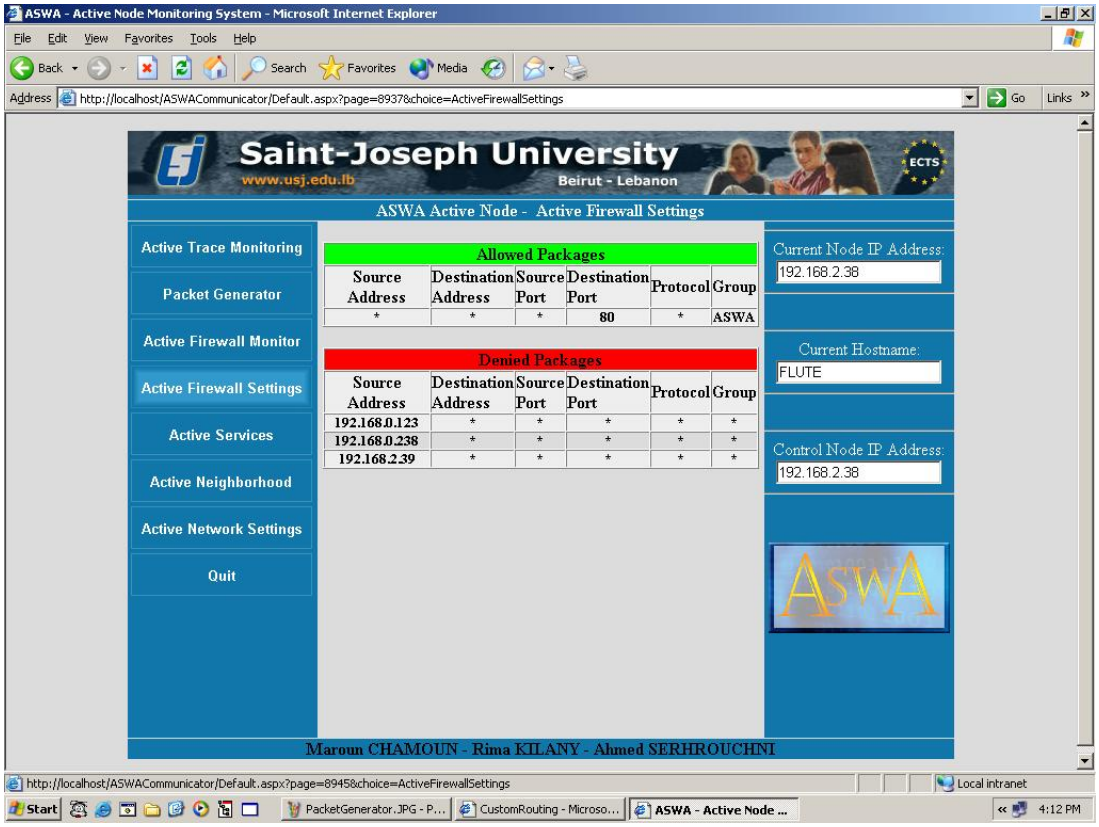

Figure 4.28 - Consultation et modification des règles de filtrage

4.9. CONCLUSION DU CHAPITRE

ASWA est une plateforme de réseau actif à base de Web services. Dans ce paragraphe, nous allons décrire quelques unes de ses caractéristiques :

- ASWA se caractérise par une architecture modulaire. En effet, on peut ajouter et retirer des modules (sous forme de Web services (section C.4) et/ou des filtres personnalisés (section C.5.2)) à l'environnement d'exécution selon le besoin. Quand un service n'a pas besoin d'être présent sur un nœud actif, il peut être désinstallé par l'administrateur pour que les paquets passent sans rencontrer de délais additionnels, ce qui réduit la latence au niveau du noeud.
- Contrairement aux autres plateformes de réseaux actifs, aucune API particulière n'est nécessaire pour le développement des applications actives sur ASWA. En effet :
  - Les paquets ASWA sont transportés au niveau de messages SOAP (section C.4.8) construits implicitement par la plateforme .NET (section C.2). Par la suite, un entête ASWA de base (voir section 3.5.1) est encapsulé dans ces messages SOAP. Cette tâche est aussi effectuée de façon implicite par la plateforme elle-même.
  - La recherche des voisins et l'envoi des paquets au nœud suivant sont assurés par la plateforme elle-même.
  - L'appel d'une application active se fait à l'aide d'une requête HTTP sous la forme : http://AdresseServeur/WebService.aswa?method=WebMethod,param=param1,... Cette requête est reçue par le gestionnaire de requêtes HTTP (section C.3.4) que nous avons implémenté. Ce gestionnaire prépare l'environnement d'exécution nécessaire pour l'activation du Web service et l'invocation de la méthode Web correspondante.
- ASWA se caractérise par l'indépendance des applications actives du langage de programmation grâce aux Web services (section C.4).
- ASWA peut être considérée comme étant une amélioration de la plateforme .NET (section C.2) concernant l'appel des Web services. En effet, chaque client .NET ayant besoin d'appeler des méthodes d'un Web service bien déterminé, doit commencer par créer un proxy (section C.4.6) dont le rôle est d'émuler les appels des méthodes distantes par des fonctions locales. Cette génération de proxy est obligatoire et sa difficulté varie, sur la même plateforme, d'un langage à un autre. Cette génération est automatique et immédiate pour les langages C# et Visual Basic et manuelle pour les autres langages et parfois difficile surtout pour les langages non orientés objet. D'autre part, chaque langage génère son propre proxy. Dans ASWA, la génération et l'appel du proxy sont transparents pour le client et sont assurés par le gestionnaire de requêtes HTTP (section C.3.4) personnalisé que nous avons implémenté.
- L'utilisation des services de sécurité est transparente pour le développeur des applications actives. En effet, les services de sécurité sont implémentés sous forme des filtres personnalisés (section C.5.2) et peuvent être ajoutés et retirés par l'administrateur par simple configuration. Ces filtres seront donc traversés par tous les paquets actifs échangés entre l'application cliente et l'application active. Ceci permettra de les contrôler à l'envoi et à la transmission.
- Bien que théoriquement ASWA est indépendante de la plateforme grâce à l'‘utilisation des Web services, son implémentation dépend des services fournis par la plateforme .NET de Microsoft. Ce dernier est implémenté actuellement sous Windows, mais une implémentation sur différents systèmes d'exploitation comme Linux, FreeBSD, Solaris et Mac OS X a été déclarée par le projet Mono [113] sponsorisé par Novell. Ceci assure la portabilité de notre architecture sur toutes ces plateformes.

Dans ce chapitre, nous avons explicité tous les détails de l'implémentation de la plateforme ASWA, ainsi que l'implémentation d'un pare-feu distribué sous forme d'un module actif au niveau de son environnement d'exécution. Dans le chapitre suivant, nous décrirons l'implémentation d'une application active représentant un protocole de gestion de réseaux à base de politiques basé sur le protocole COPS, où la communication se fait à l'aide des Web Services fournis par ASWA, la définition des règles de politiques et la représentation de la base des informations de politiques se fait en utilisant le langage OWL en vue d'en améliorer la sémantique.

Chapitre 5

APPROCHE ACTIVE ET SEMANTIQUE POUR LA GESTION DES RESEAUX PAR POLITIQUE

5.1. INTRODUCTION

Pour pouvoir gérer la plateforme active ASWA, et par extension d'autres réseaux, nous avons défini et déployé un protocole de gestion à base de politiques conformément à COPS pour l'échange des informations entre le serveur de politiques et ses clients.

L'avantage principal reconnu dans l'utilisation de la technologie des réseaux actifs est la vitesse du déploiement du service et de la distribution des tâches à travers le réseau. Seulement, la technologie doit pouvoir présenter un certain niveau de compétitivité par rapport aux réseaux existants en termes de sécurité et de robustesse, chose difficile à gérer quand on a dans ce type de réseau un transport fréquent d'information et une exécution de code dans un vaste système hétérogène. Des solutions à ces problèmes sont apportées avec l'utilisation du protocole SOAP pour les communications notamment dans le cas de l'implémentation de COPS dans ASWA. Les bénéfices de cette implémentation et plus précisément l'implémentation de l'extension COPS-PR sur ASWA sont de:

- Permettre aux services de définir leurs propres mécanismes de traitement des données dans les nœuds intermédiaires et avoir accès à certaines informations sur ces nœuds.
- Profiter des facilités des réseaux actifs pour proposer une solution de gestion et de contrôle flexible.
- Prendre avantage de cette plateforme à base de Web services qui offre une solution assurant interopérabilité et transparence dans un environnement hétérogène. Ceci est assuré en encapsulant les messages COPS dans des messages SOAP.
- Déployer dynamiquement et automatiquement des PEPs et échanger des messages sécurisés.

La base de données des politiques (ou PIB) étudiée dans la section A.4.2, et plus précisément dans le cadre de l'implémentation de COPS-PR définit les informations dans le domaine de gestion de manière plus ou moins formelle. Pourtant, la sémantique de cette base de données est limitée puisqu'elle n'inclut aucune restriction. Nous proposons dans ce chapitre l'utilisation d'une ontologie pour la définition de la PIB. Ceci permet d'avoir des axiomes et contraintes qui peuvent être interprétés et automatiquement lus par le système de gestion puisqu'ils seront définis de manière formelle.

Nous verrons également dans ce chapitre que l'utilisation de SWRL (qui est une extension

de OWL), comme langage de spécification des règles de politiques, marque un grand avantage puisqu'il permettra d'éviter une étape de traduction entre les règles de politiques à installer et la base de données de ces politiques.

5.2. MODELE CONCEPTUEL

Nous proposons dans ce chapitre une solution au problème de contrôle dynamique et intelligent dans la gestion des réseaux et ceci en développant un modèle en trois plans basé sur la synergie entre les 4 paradigmes: Réseaux à base de politique, Réseaux actifs, Web services et Ontologie.

La Figure 5.1 illustre cette architecture à trois plans qui est définie ainsi:

- L'infrastructure de cette architecture est un plan de déploiement dynamique de services réalisé sous forme d'un réseau actif à base de Web services. Pour permettre de contrôler et de sécuriser le déploiement des services, nous avons adopté une autre architecture à trois plans dont laquelle on trouve le plan de transport, le plan actif et le plan de contrôle actif (Chapitre 3).
- Au niveau le plus haut se trouve le plan de signalisation dont le rôle est de contrôler intelligemment les services. Le contrôle est assuré en définissant des méta-données sémantiques qui sont utilisées pour qualifier et décrire sans ambiguité la sémantique des données. Ce plan permet ainsi d'enrichir l'exploration des informations définies au niveau du plan de service et donner la capacité d'automatiser leur traitement.
- Au milieu de cette architecture se trouve le plan de services. C'est un plan générique qui représente tout service qui a besoin d'une part d'un déploiement dynamique et d'autre part d'un contrôle intelligent.

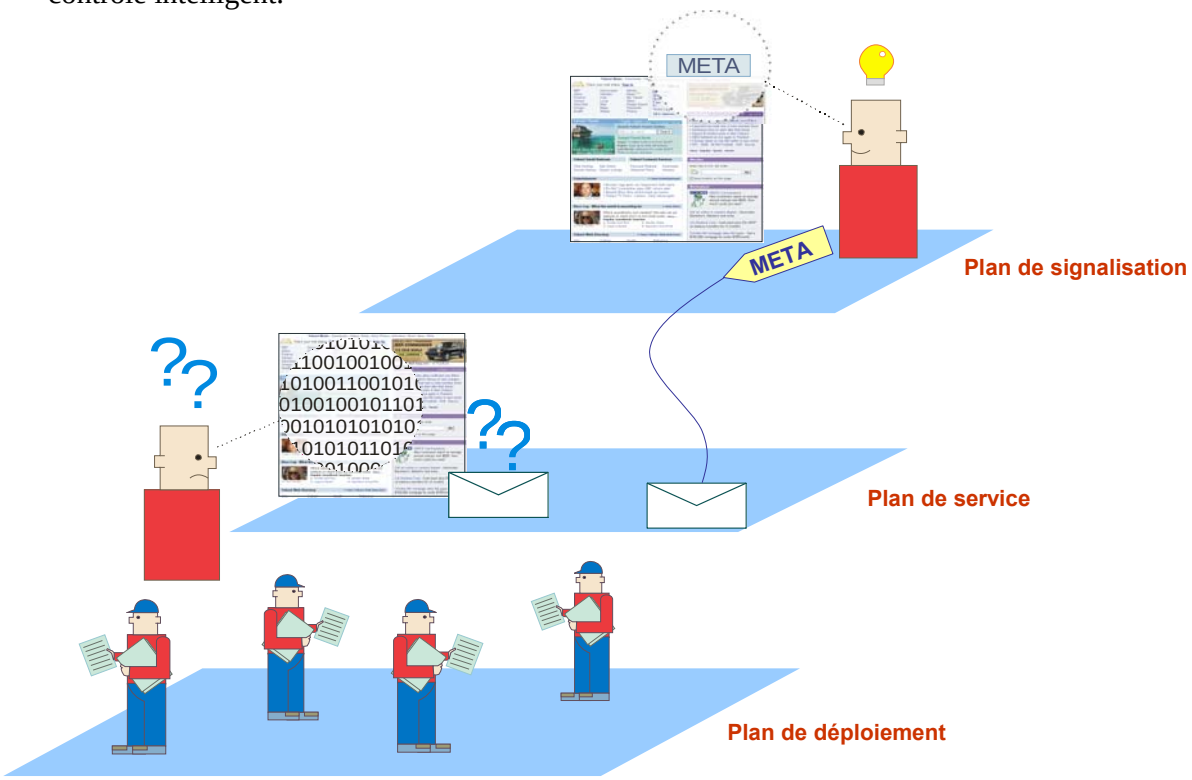

Figure 5.1 – Modèle conceptuel de l'approche active et sémantique pour la gestion par politique

Pour valider ce modèle conceptuel, nous l'avons appliqué au domaine de la gestion des réseaux. Dans cette application, les trois plans ont été réalisés ainsi :

- Le réseau actif à base de Web services qu'on a appelé ASWA a été implémenté sur une plateforme .NET de Windows (Chapitre 4).

- Dans le plan de service nous avons implémenté le protocole COPS-PR pour la gestion des réseaux sous forme de Web services. Dans cette implémentation les messages COPS sont encapsulés dans des messages SOAP, ce qui améliore la mobilité de ces messages.
- Le plan de signalisation a été réalisé en utilisant le langage d'ontologie OWL et de son extension SWRL. Ceci permet de rendre les informations de gestion des réseaux à base de politiques plus expressives dans leurs sémantiques.

5.3. IMPLEMENTATION DE COPS DANS ASWA

Nous avons vu que COPS est l'une des approches de mise en œuvre de la gestion des réseaux par politique. COPS est un protocole standardisé qui va permettre aux routeurs et aux différents équipements du réseau d'échanger des informations avec un serveur centralisé qui stocke l'ensemble des politiques. Mais l'échange de messages COPS n'est pas en mesure de franchir les barrières d'Internet. Pour contourner ce problème de mobilité, nous avons encapsulé ces messages COPS dans des messages SOAP. Tout en conservant l'interopérabilité avec les réseaux COPS existants standard, nous avons implémenté l'échange et le format des messages conformément à COPS.

L'implémentation a été faite en langage C# sur la plateforme .Net. Les éléments de cette implémentation se résument par les points suivants:

- L'implémentation du PDP et du PEP sous la forme des Web services.
- Le PDP traite les politiques et les envoie aux PEPs qui les intègrent.
- L'implémentation d'un gestionnaire de politique pour déployer dynamiquement les PIBs aux nœuds actifs à partir d'une simple interface Web située sur le nœud de contrôle du domaine.
- L'implémentation d'un sous-ensemble de messages COPS-PR encapsulés dans des messages SOAP pour permettre à ces messages COPS de traverser les pare-feu.
- L'implémentation d'un proxy convertisseur de messages SOAP (encapsulant des messages COPS) en messages COPS purs, pour permettre la communication inter-plateforme de gestion COPS. Par exemple, en permettant à un PDP sur ASWA de communiquer avec un PEP d'une autre plateforme. Cette conversion est immédiate car les messages SOAP contiennent toutes les données nécessaires à la création des objets COPS et par suite de messages COPS.

5.3.1. L'implémentation des Web services PEP et PDP

Pour respecter l'architecture ASWA décrite précédemment nous avons créé deux Web services. Le premier représente un PEP et le second un PDP. Nous avons aussi créé une application cliente qui demande des services du PEP. Ceci permet de mettre en œuvre un processus de communication entre le PEP et le PDP à l'aide du protocole COPS.

Pour chacun de ces Web services, nous avons défini les classes nécessaires à l'établissement de la communication en conformité avec le protocole COPS : le format des messages et des objets, l'établissement de la communication entre PEP et PDP, et enfin le traitement des messages au niveau du PEP et du PDP.

5.3.1.1. Entête du message COPS

L'entête des messages COPS (section A.3.2) est défini par la classe CopsHeader qui hérite de la classe SoapHeader. Ceci permet de l'insérer dans la partie « entête » du message SOAP durant l'échange des paquets COPS entre les PEPs et le PDP.

```
public class CopsHeader : SoapHeader
{
        public byte Vers_Flag;
        public Int16 Opcode;
        public byte Client_Type;
        public int Length ;
}
```

Figure 5.2 - Classe définissant l'entête du message COPS

Le champ « Length », qui représente la taille du message COPS, sera calculé pour chaque message envoyé dépendamment du nombre d'objets COPS présents dans le message et de la longueur de chacun de ces objets.

5.3.1.2. Entête des objets COPS

Un objet contient un entête de taille fixe et un contenu dont la taille dépend de chaque objet. L'entête de l'objet COPS (section A.3.2) est défini à l'aide de la classe CopsObject suivante :

```
public class CopsObject
{
        public Int16 Length;
        public byte CNum;
        public byte CType;
}
```

Figure 5.3 - Classe définissant l'entête d'un objet COPS

Comme chaque objet possède un contenu de type variable (Object Contents) qui dépend de la nature de l'objet, le champ « Length » de la classe CopsObject, qui représente la taille de l'objet, sera calculé de façon spécifique pour chaque objet.

5.3.1.3. Les objets COPS

Chaque objet COPS est une instance d'une classe spécifique qui hérite de CopsObject et lui ajoute ses données spécifiques. Par exemple, l'objet « error » a, en plus de l'entête de l'objet COPS, deux champs qui sont :

- Error Code dont la taille est de deux octets.
- Error Sub-Code dont la taille est de deux octets.

La classe qui la définit est la suivante :

```
public class Error : CopsObject
{
        public Int16 Code;
        public Int16 SubCode;
        public Error( )
        {
                this.CNum = 8 ;
                this.CType = 1 ;
        }
}
```

Figure 5.4 - Classe définissant l'objet COPS "error"

L'objet KeepAlive envoyé avec le message ClientAccept est un autre exemple. Dans ce message, le PDP envoie au PEP son accord sur l'ouverture de la connexion et un objet KeepAlive dans lequel il impose un temps maximal avant lequel le PEP doit envoyer un signe de vie pour que le PDP garde la connexion.

```
public class KeepAlive : CopsObject
{
        public Int16 Comment;
        public Int16 KATimer;
        public KeepAlive( )
        {
                this.CNum = 10 ;
                this.CType = 1 ;
        }
}
```

Figure 5.5 - Classe définissant l'objet COPS "KeepAlive"

L'envoi de l'objet KeepAlive se fait après son instanciation et l'affectation de ses attributs comme le montre la Figure 5.6:

```
KA = new KeepAlive( );
KA.KATimer = 3200;
KA.Comment = 0;
```

Figure 5.6 - Instanciation d'un objet de type KeepAlive

5.3.2. Implémentation de la communication entre PEP et PDP

Après avoir décrit le format d'un message COPS, nous décrivons au niveau de cette section les détails de l'implémentation de la communication entre PDP et PEP (section A.3.3).

5.3.2.1. L'établissement du contact

L'établissement du contact entre PEP et PDP se fait à l'aide des deux méthodes Contact-PDP et ContactPEP.

En effet, le PEP envoie son message au PDP en remplissant un « CopsHeader » et en appelant la méthode « ContactPDP » (définie dans la Figure 5.7) qui va s'occuper du traitement du message.

```
 [WebMethod]
[SoapHeader("MyHeader", Direction=SoapHeaderDirection.In)]
public void ContactPDP( CopsObject[])
{
                //Traitement au niveau du PDP
}
```

Figure 5.7 - Envoi d'un message au PDP

De même, le PDP communique avec le PEP en remplissant un « CopsHeader » et en appelant la méthode « ContactPEP » (définie dans la Figure 5.8) qui va s'occuper du traitement du message.

```
[WebMethod]
[SoapHeader("MyHeader", Direction=SoapHeaderDirection.In)]
public void ContactPEP( CopsObject[])
{
                //Traitement au niveau du PEP
}
```

Figure 5.8 - Envoi d'un message au PEP

Comme chaque message COPS contient un nombre indéterminé d'objets, ces méthodes admettent comme paramètre un tableau d'objets COPS et reçoivent un message SOAP.

5.3.2.2. Traitement au niveau du PEP et du PDP

Le traitement d'un message COPS est régi par deux valeurs :

- Le type de client (*Client-Type*).
- Le code de l'opération (OpCode) qui permet à un acteur COPS de déduire les Objets qui suivent et le traitement à appliquer.

Ce traitement se fait au niveau de la méthode Web ContactPDP (cas où les messages sont en provenance d'un PEP) ou au niveau de la méthode Web ContactPEP (cas où les messages sont en provenance d'un PDP). Il consiste à :

- décortiquer le message,
- lire le Client-Type et le OpCode
- appeler la méthode locale chargée de traiter ce type de message.

```
[WebMethod]
[SoapHeader("MyHeader", Direction=SoapHeaderDirection.In)]
public void ContactPEP( CopsObject[] coarr)
{
        if (MyHeader.OpCode == 2)
        {
```

```
                // Fill the headers
                // Treat the message
                // call the appropriate method
                UnsollicitedDecision(Handle, Context, Desicion, NamedData, XmlData) ;
        }

        if (MyHeader.OpCode == 7)
        {
                // Fill the headers
                // Treat the message
                // call the appropriate method
                KeepAlive ccol = new KeepAlive( );
                ccol = (KeepAlive)coarr[0];
                ClientAccept(ccol);
        }

        if (MyHeader.OpCode == 8)
        {
                // Fill the headers
                // Treat the message
                // call the appropriate method: ClientClose
        }

}
```

Figure 5.9 - Traitement des messages au niveau du PEP

```
[WebMethod]
[SoapHeader("MyHeader", Direction=SoapHeaderDirection.In)]
public void ContactPDP( CopsObject[] coarr)
{
        if (MyHeader.OpCode == 6)
        {
                // Fill the headers
                // Treat the message
                // call the appropriate method: ClientOpen
        }

        if (MyHeader.OpCode == 8)
        {
                // Fill the headers
                // Treat the message
                // call the appropriate method: ClientClose
        }

}
```

Figure 5.10 - Traitement des messages au niveau du PDP

5.3.3. Exemple d'un dialogue COPS

Soit, par exemple le scénario suivant : un pare-feu jouant le rôle d'un PEP (tel que nous l'avons implémenté) est installé au périmètre d'un réseau bien déterminé. Son rôle est de filtrer tous les paquets entrants et sortants. A son démarrage ce pare-feu contacte le PDP de son domaine. Ce dernier lui envoie alors les politiques de filtrage. Pour compléter ce scénario, nous supposons que plus tard, le PDP et suite à une modification au niveau des règles de filtrage par l'administrateur du réseau, envoie à ce pare-feu un message non sollicité contenant la mise à jour de ses politiques. Les détails de ce dialogue (Figure 5.11) suivent les étapes suivantes :

- Pour commencer un dialogue COPS, le PEP doit demander l'ouverture d'une connexion et ceci par l'envoi d'un message « **ClientOpen** ». Pour ce faire, il prépare un « CopsHeader »,

qui représente l'entête du message COPS, et qui contient un opcode de valeur 6 (celle de l'opération open-client) et un Client-Type de valeur 0x8888 (celle du filtrage).
Dans le corps du message SOAP s'ajoute les objets COPS suivants :
- PEPID qui contient un identificateur du PEP et du Client-Type,
- ClientInfo qui contient l'URL du PEP et qui sera utile au cas où le PDP aura à traiter avec plusieurs PEPs.

Ces objets sont envoyés dans un tableau en tant que paramètres lors de l'appel de la fonction « ContactPDP ».

- A l'appel de la méthode « ContactPDP », le service PDP reçoit le message SOAP et commence son **traitement**. Comme l'OpCode correspond à un « ClientOpen », le PDP appelle une fonction « ClientOpen » locale qui vérifie que l'identificateur fourni par le PEP est unique. Cette vérification se fait à l'aide d'une table contenant les identificateurs déjà alloués, les URL correspondantes et le type de client.
  - Si l'identificateur existe déjà, le PDP renvoie un message d'erreur.
  - Sinon, il l'ajoute dans sa table et appelle une fonction qui va préparer le message « Client Accept » pour l'envoyer au PEP.
- La fonction « **ClientAccept** » prépare le message en mettant dans l'entête le même client-type reçu et 7 pour valeur d'OpCode (celle de l'opération Client-Accept) et l'envoie au PEP (par l'appel de la méthode « ContactPEP ») qui, à son tour, traite le message en suivant le même principe.
- Le but de ce dialogue est surtout la communication de règles de politiques entre le PEP et le PDP. Ceci se fait par exemple, par l'envoi d'un « **UnsollicitedDecision** » de la part du PDP vers le PEP.
  Ce message est complexe. Il comporte en effet au moins 6 objets, le dernier objet étant de type « NamedData », et comportera dans son contenu une chaîne de caractères :
  - Si le champ C-Type de l'entête de cet objet est égale à 1, cette chaîne correspondera à des données XML qui impliquent une ou plusieurs règles que le PDP veut imposer au PEP.
  - Si C-Type est égale à 2, cette chaîne correspondera à des données codées en BER.
  - Nous avons choisi d'affecter la valeur 3 à C-Type pour indiquer que la chaîne sera codée en OWL.
- Quand le PDP reçoit du PEP un « **clientClose** », l'identificateur du PEP est libéré par le PDP. La table du PDP sera alors mise à jour pour qu'un autre PEP puisse utiliser ce même identificateur.

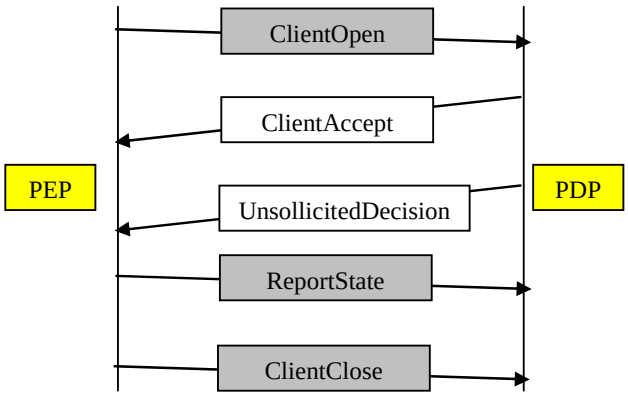

Figure 5.11 - Exemple d'un dialogue

5.3.4. Exemple de messages SOAP échangés

Une fois la connexion est ouverte, le PEP peut transmettre sa requête par l'envoi d'un message « Request ». La Figure 5.12 illustre l'exemple d'une telle requête (OpCode == 1) transmise au sein de la plateforme ASWA.

```
<soap:Envelope xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
```

```
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsse=http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
<soap:Header>
    <!--Entête du paquet actif-->
    <aswa:ASWA aswa:ProtocolID="COPS" xmlns:aswa="http://fi.usj.edu.lb/aswa">
     <ProtocolID>COPS</ProtocolID>
    </aswa:ASWA>
    <ASWAFirewallHeader>
     <srcaddr>34.34.34.3</srcaddr>
     <dstaddr>192.168.0.128</dstaddr>
     <srcport>1234</srcport>
     <dstport>80</dstport>
     <protocol>COPS</protocol>
     <UserGrp>ASWA</UserGrp>
    </ASWAFirewallHeader>
    <CopsHeader xmlns="http://tempuri.org/">
        <Vers_Flag>17</Vers_Flag>
        <Opcode>1</Opcode>
        <Client_Type>0</Client_Type>
        <Length>28</Length>
    </CopsHeader>
   <wsse:Security>
     <wsu:Timestamp wsu:Id="Timestamp-f61e9b92-a5f8-4b0e-8bf2-e7807a7cf784">
      <wsu:Created>2005-05-31T04:20:00Z</wsu:Created>
      <wsu:Expires>2005-05-31T04:25:00Z</wsu:Expires>
     </wsu:Timestamp>
   </wsse:Security>
  </soap:Header>
  <soap:Body>
   <ContactPDP xmlns="http://tempuri.org/">
        <coarr>
            <CopsObject xsi:type="ClientHandle">
                <Length>12</Length>
                <CNum>0</CNum>
                <CType>0</CType>
                <HandleValue>ABCD0000</HandleValue>
            </CopsObject>
            <CopsObject xsi:type="context">
                <Length>8</Length>
                <CNum>0</CNum>
                <CType>0</CType>
                <RType>1</RType>
                <MType>1</MType>
            </CopsObject>
        </coarr>
   </ContactPDP>
  </soap:Body>
 </soap:Envelope>
```

Figure 5.12 - Message COPS de type "Request" (REQ), encapsulé dans un message SOAP

Comme illustré dans la Figure 5.12, l'objet « *ClientHandle* », défini dans le corps du message SOAP, contient une valeur unique générée par le PEP permettant de référencer les différentes configurations ou états qu'il manipule pour un même type de client. Cet objet est ensuite utilisé par le PDP dans toutes les informations échangées et décisions communiquées.

5.4. REPRÉSENTATION SÉMANTIQUE DES INFORMATIONS DE GESTION

Dans cette section nous proposons une nouvelle approche pour la définition des informations de gestion des réseaux à base de politiques afin de rendre celles-ci plus expressives dans leurs sémantiques (valeur ajoutée par rapport aux langages habituels). Pour cela, nous avons fait le choix de OWL (section B.2.7), le langage d'ontologies Web conçu par le Consortium WWW et permettant la définition d'ontologies (section B.2.4) basées Web.

Les principaux avantages de notre proposition sont :

- Mis à part les avantages de la syntaxe de type XML, les langages d'ontologies sont formels et leurs sémantiques est donc complète, d'autant plus que des moyens permettent de les tester et les valider.
- Un autre avantage se base sur la définition interne qui n'est pas faite à l'aide de langages habituels difficiles à lire pour les machines. Les axiomes et contraintes des ontologies peuvent être aisément interprétés par le système de gestion par politiques.
- Notons également la possibilité pour un administrateur de spécifier le comportement des ressources et plus précisément des règles de politiques à utiliser. Ainsi le langage de spécification (SWRL qui est une extension de OWL) et celui de représentation (OWL) est le même (l'interpréteur de langage est donc unique).
- La PIB est décrite en ASN.1 de manière abstraite mais pratiquement chaque implémentation de système de gestion de réseau utilise ses propres méthodes pour la définition et l'utilisation de la structure arboresente de cette base de données. Définir des instances individuelles des classes dans une ontologie permet d'étendre l'arbre de représentation de l'information et de représenter cette PIB avec ce même langage.

5.4.1. Définition sémantique de la SPPI

La SPPI (section A.4.3) est la spécification de la structure de donnée de la PIB (l'arbre des PRCs et PRIs) (section A.4.2). Il est donc utile de pouvoir associer une sémantique à la SPPI par le concept d'ontologie à l'aide par exemple d'une structure OWL qui engloberait l'essentiel des propriétés nécessaires à cette spécification:

- Les clauses (ou champs) peuvent être spécifiées en utilisant owl:DatatypeProperty.
- Les classes peuvent être définies avec la balise owl:Class et leurs héritages avec la balise rdfs:subClassOf.
- Les clauses définies dans les macros peuvent être utilisées pour la définition de la syntaxe et la sémantique des PRCs et des attributs. Ces clauses sont juste descriptives et ne sont donc pas utiles pour les PRIs. Pour éviter l'héritage de ces clauses à tous les PRIs, la solution est de définir les macros ASN.1 comme étant des méta-classes OWL et non pas des classes OWL. Une méta-classe OWL étant une classe dont les instances sont aussi des classes.

Ces propriétés nous permettent de structurer la SPPI en utilisant une ontologie OWL (L'éditeur choisi pour effectuer cette tâche étant Protégé).

La Figure 5.13 illustre la structure des méta-classes OWL qui représente la SPPI. Les méta-classes définies sont, par analogie à la structure de la SPPI:

- ObjectIdentity représentant la macro OBJECT-IDENTITY. Elle enregistre les valeurs des différents OBJECT-IDENTIFIER.
- ObjectGroup représentant la macro OBJECT-GROUP qui définit l'ensemble des objets ayant une certaine relation.
- ModuleCompliance pour la macro MODULE-COMPLIANCE qui spécifie les modules requis pour une implémentation conforme.
- ModuleIdentity pour la macro MODULE-IDENTITY décrivant des catégories de sujets, des informations sur les contacts et l'historique de révision de la PIB.
- Enfin, la macro OBJECT-TYPE est représentée par la méta-classe OT_Base et toutes ses mé-

ta classes dérivées. Cette macro ayant été utilisée pour définir les tables, les entrées des tables (PRCs) et les attributs des entrées, nous avons défini toutes les propriétés communes à ces éléments dans la méta-classe de base OT_Base. Pour les clauses spécifiques, on dérive de OT_Base les méta-classes OT_Table, OT_Entry et OT_Attribute.

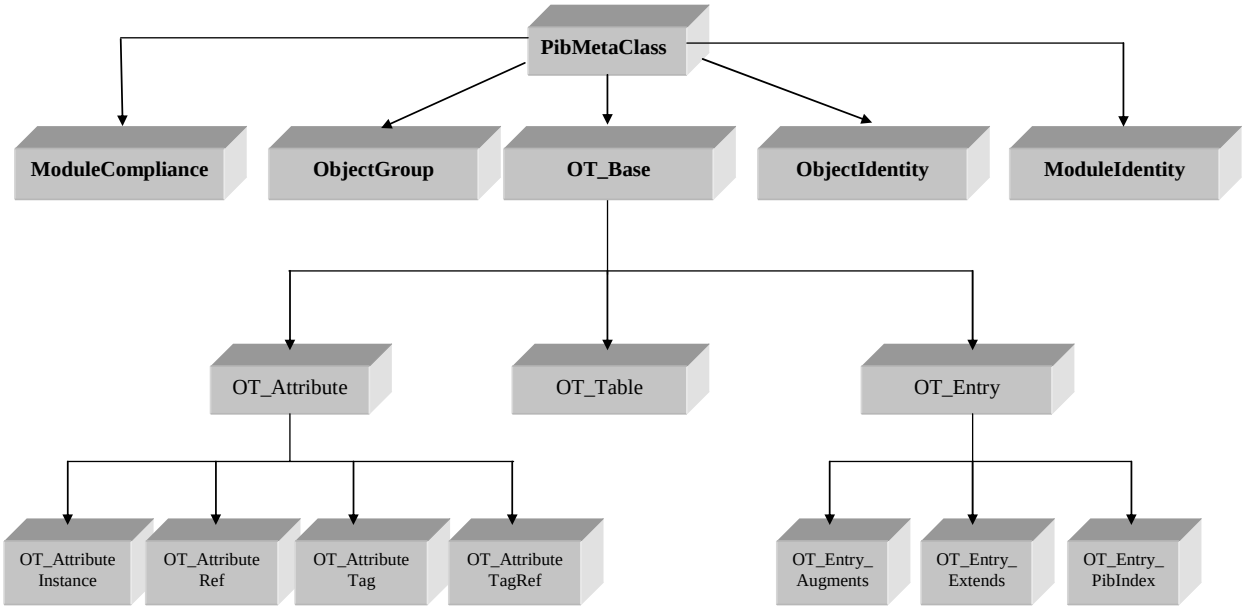

Figure 5.13 - Méta-Classes représentant les macros

La Figure 5.14 montre comment ces méta-classes ont été créées dans l'éditeur. Elles sont toutes dérivées de PibMetaClass qui contient le DataProperty OID, unique propriété commune à toutes les macros représentées par ces méta-classes.

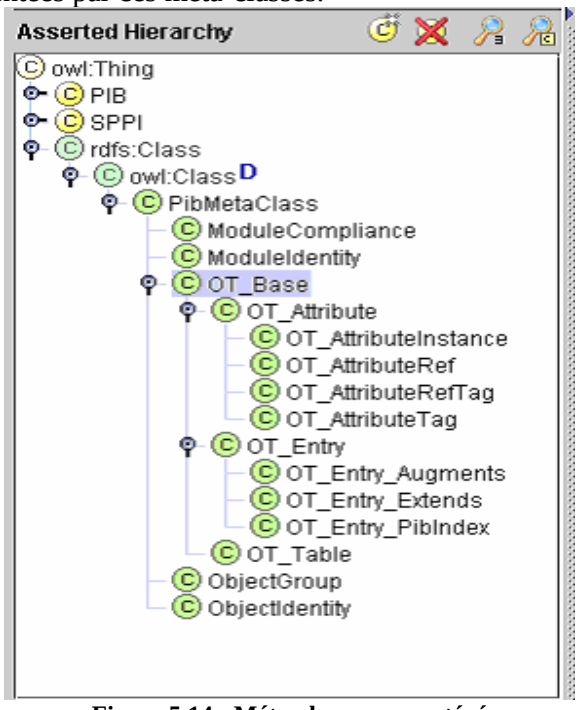

Figure 5.14 - Méta-classes en protégé

Notons que certaines clauses de la macro OBJECT-TYPE peuvent exclusivement être associées à une PRC particulière. Tel est le cas de TagReferenceID qui ne doit être utilisé que par une PRC ayant TagReferenceId comme valeur d'attributs. Autant que ceci est difficile à modéliser en ASN.1, autant que c'est facile à faire en utilisant les méthodes sémantiques et des restrictions OWL dans la définition des classes ou des méta-classes. Ceci est bien illustré dans la Figure

5.15 :

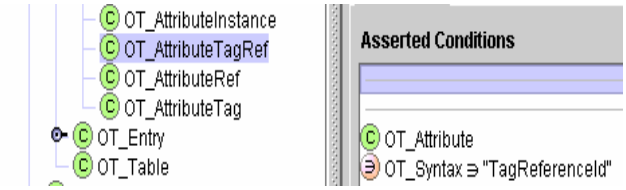

Figure 5.15 - Restrictions dans OWL

Etant donné que certains attributs de ces méta-classes nécessitent une définition particulière, des classes ont été définies afin de pouvoir en instancier ces attributs.

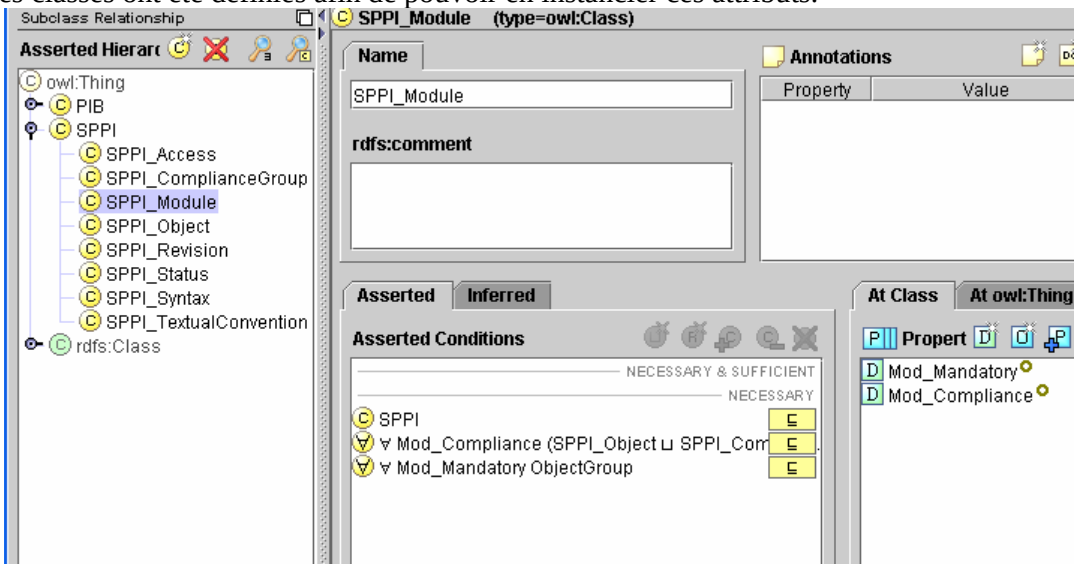

Figure 5.16 - Attributs de SPPI

Par souci de structure, ces classes ont été dérivées d'une classe de base: SPPI. Elles sont représentées dans la Figure 5.16:

Remarquons par exemple qu'un attribut qui doit obligatoirement être une instance de SPPI_Module doit avoir deux attributs Mod_Mandatory et Mod_Compliance qui eux-mêmes ont des restrictions sur leurs instanciations.

Pour mettre en relief encore plus l'aspect sémantique et verbeux de l'utilisation de OWL vis-à-vis du caractère syntaxique dans l'utilisation de ASN.1, nous montrons dans la Figure 5.17 la définition en ASN.1 de la méta-classe OT_Entry_PibIndex ainsi que cette définition en OWL dans la Figure 5.18. On voit bien, dans cette dernière figure, l'héritage de la classe OT_Entry ainsi que la restriction dans les valeurs d'attributs.

```
OBJECT-TYPE MACRO ::=
BEGIN
   TYPE NOTATION ::=
              "SYNTAX" Syntax
              UnitsPart
              "PIB-ACCESS" Access     -- modified
              PibReferencesPart              -- new
              PibTagPart              -- new
              "STATUS" Status
              "DESCRIPTION" Text
              ReferPart
              IndexPart              -- modified
              MibIndexPart          -- modified
…
IndexPart ::=
```

```
              "PIB-INDEX"          "{" Index "}"      -- new
            | "AUGMENTS"           "{" Entry "}"
            | "EXTENDS"            "{" Entry "}"      -- new
            | empty
Index ::=
               -- the correspondent OBJECT-TYPE invocation
              value(ObjectName)
Entry ::=
               -- use the INDEX value of the
               -- correspondent OBJECT-TYPE invocation
MibIndexPart ::=
              "INDEX"   "{" IndexTypePart "}"
            | empty
Attr ::=      -- specifies an attribute
              value(ObjectName
…
END
```

Figure 5.17 - Définition ASN.1 de OT_Entry_PibIndex

```
<owl:Class rdf:ID="OT_Entry_PibIndex">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="OT_Entry"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom>
        <owl:DataRange>
          <owl:oneOf rdf:parseType="Resource">
            <rdf:rest  rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
            <rdf:first  rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
      Cls(OT_Attribute,FrameID(1:10086))</rdf:first>
          </owl:oneOf>
        </owl:DataRange>
      </owl:allValuesFrom>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:ID="OT_MibIndex"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
 </owl:Class>
```

Figure 5.18 - Représentation OWL de la méta-classe OT_Entry_PibIndex

5.4.2. Définition sémantique de la PIB

5.4.2.1. Définition d'un modèle générique pour la PIB

En utilisant les définitions sémantiques de la SPPI décrites dans la section précédente, on peut à présent construire un modèle sémantique de l'ontologie de la PIB. La Figure 5.19 illustre ce modèle générique.

Les éléments de la PIB sont structurés de la manière suivante : Une PIB est défini dans un module (de type MODULE-IDENTITY). Le module peut être constitué d'un ou de plusieurs Groupes (pour lesquels on a besoin seulement de définir un OID). Ces groupes seront donc de type PibMetaClass. Chaque groupe est constitué d'une ou de plusieurs Tables (de type OBJECT-TYPE avec des clauses spécifiques pour les tables). Une table contient une et une seule Entrée (de type OBJECT-TYPE avec des clauses spécifiques pour les entrées) dans laquelle on peut créer plusieurs instances (qui seront les politiques). Chaque entrée est constituée d'un ou de plusieurs attributs (de type OBJECT-TYPE avec des clauses spécifiques pour les attributs).

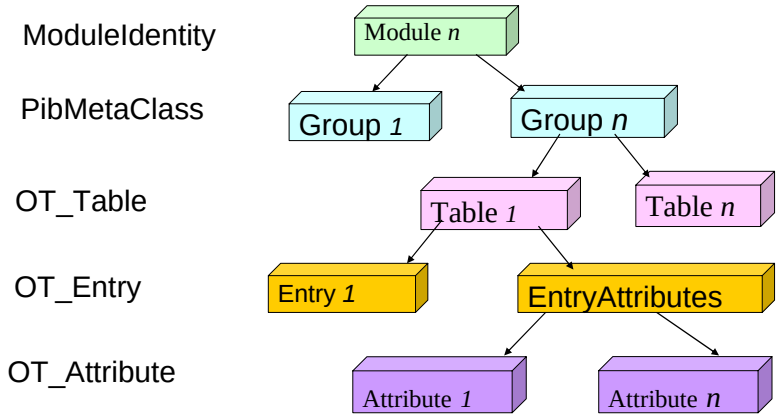

Figure 5.19 - Eléments de la PIB

Afin de valider cette structure, nous avons défini deux PIBs : une PIB DiffServ [148] et une PIB de filtrage [44], que nous allons détailler dans les sections qui suivent.

5.4.2.2. Exemple 1 : La PIB DiffServ

La Figure 5.20 montre une partie de la PIB DiffServ, tel que définie au niveau de l'éditeur protégé, tandis que la Figure 5.21 montre le modèle défini dans la Figure 5.19.

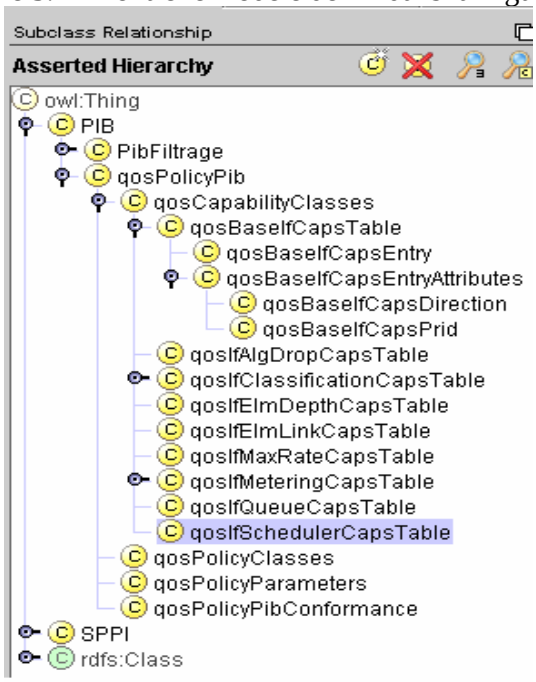

Figure 5.20 - Exemple de la PIB DiffServ

- Chaque PIB possède un *MODULE-IDENTITY*: La classe *qosPolicyPIB* est une instance de la méta-classe *ModuleIdentity* rattachée à la classe PIB par héritage.
- Un module PIB est divisé en des groupes qui ont chacun un OID : La classe *qosCapability-Classes* qui dérive de *qosPolicyPib* en est l'exemple.
- Un groupe possède différentes tables et les classes représentant les tables sont elles-mêmes dérivées d'instances de la méta-classe *OT_Table*.
- Les entrées sont des instances de la méta-classe *OT_Entry* et doivent être dérivées d'une instance de *OT_Table* : Par exemple, la classe *qosBaseIfCapsEntry* dérive de *qosBaseIfCapsTable*.
- Les attributs dérivent d'une classe instance de la méta-classe *OT_Attribute* : *qosBaseIfCapsAttributes* dérive de *qosBaseIfCapsEntry*.
- Les PRIs sont des instances des attributs et sont définies comme des individus (*individuals*)

dans *Protégé*, celles-ci étant les politiques à appliquer.

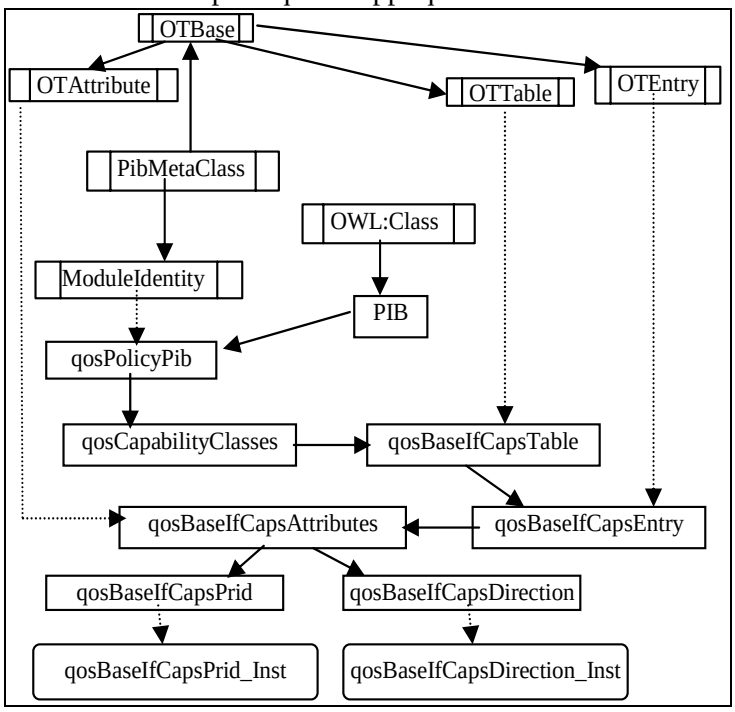

Figure 5.21 - Taxonomie de la PIB QdS de DiffServ

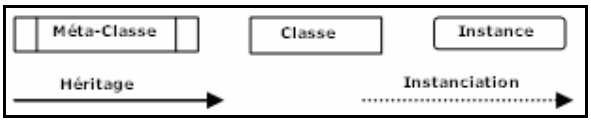

Figure 5.22- Convention adoptée par la Figure 5.21

La PIB n'est en définitive qu'un fichier OWL que nous avons créé en se basant sur la description abstraite de celle-ci en ASN1 et dont un extrait du groupe *qosCapabilityClasses* est illustré dans la Figure 5.23 suivante (Cf. la section D.1.1 de l'Annexe D pour la définition complète de ce groupe de la PIB DiffServ):

```
qosPolicyPib  MODULE-IDENTITY
 …
::= { pib xxx }

qosCapabilityClasses  OBJECT IDENTIFIER ::={qosPolicyPib 1}
qosPolicyClasses   OBJECT IDENTIFIER ::= {qosPolicyPib 2}
qosPolicyParameters OBJECT IDENTIFIER ::= {qosPolicyPib 3 }

qosBaseIfCapsTable OBJECT-TYPE
…
   ::= { qosCapabilityClasses 1 }
qosBaseIfCapsEntry OBJECT-TYPE
…
::= { qosBaseIfCapsTable 1 }
QosBaseIfCapsEntry ::= SEQUENCE {
       qosBaseIfCapsPrid InstanceId,
       qosBaseIfCapsDirection Integer32
}
qosBaseIfCapsPrid OBJECT-TYPE
…
   ::= { qosBaseIfCapsEntry 1 }
qosBaseIfCapsDirection OBJECT-TYPE
```

```
   …
   ::= { qosBaseIfCapsEntry 2 }
…
```

Figure 5.23 - Représentation en ASN.1 d'une partie de la PIB DiffServ

La définition détaillée du format OWL de cette représentation ASN.1 de la PIB DiffServ se trouve dans la section D.1.2 de l'Annexe D. Un extrait de cette définition est illustré dans la Figure 5.24 suivante:

```
<owl:ObjectProperty rdf:ID="qosBaseIfCapsDirection_Attribute">
  <rdfs:domain>
     <OT_Entry_PibIndex rdf:ID="qosBaseIfCapsEntry">
        <rdfs:subClassOf>
           <OT_Table rdf:ID="qosBaseIfCapsTable">
              <rdfs:subClassOf>
              <PibMetaClass rdf:ID="qosCapabilityClasses">
              <rdfs:subClassOf>
                 <ModuleIdentity rdf:ID="qosPolicyPib">
                    …
                 </ModuleIdentity>
              </rdfs:subClassOf>
              <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">2.1</OID>
           </PibMetaClass>
         </rdfs:subClassOf>
         …
</owl:ObjectProperty>
```

Figure 5.24 - Représentation en OWL d'une partie de la PIB DiffServ

5.4.2.3. Exemple 2: La PIB de Filtrage

Le deuxième exemple nous ayant permis de valider notre modèle sémantique de la PIB, n'est autre que celui de la PIB de filtrage dont un extrait de la définition en ASN.1 est présenté à la Figure 5.25 (Cf. la section D.2.1 de l'Annexe D pour la définition complète de cette PIB). La Figure 5.26 illustre la définition de la PIB de Filtrage au niveau de l'éditeur *protégé*:

```
ipv4FilterIpFilter OBJECT IDENTIFIER ::= { someExampleOID 1 }
            …
     ::= { ipv4FilterIpFilter 1 }

  ipv4FilterEntry OBJECT-TYPE
     …
     ::= { ipv4FilterTable 1 }
  Ipv4FilterEntry ::= SEQUENCE {
        ipv4FilterIndex       Unsigned32,
        …
  }
  ipv4FilterIndex OBJECT-TYPE
     …
     ::= { ipv4FilterEntry 1 }
…
```

Figure 5.25 - PIB de Filtrage en ASN.1

On voit bien que le module créé pour la PIB Filtrage est : *ipv4FilterPib*. *Ipv4FilterIpFilter* en dérive et définit le OID de base (1 par exemple). De cette classe dérive celle de la seule table de filtrage, ipv4FilterTable. Ipv4FilterEntry dérive de ipv4FilterTable et possède un certain nombre de propriétés chacune représentant un attribut de l'entrée et devant avoir pour valeur une instance d'une des classes dérivant de ipv4FilterEntryAttributes. En effet, les classes qui dérivent de cette dernière permettent d'attribuer un type spécifique pour la valeur des propriétés comme par exemple "boolean" pour la propriété "ipv4FilterPermit".

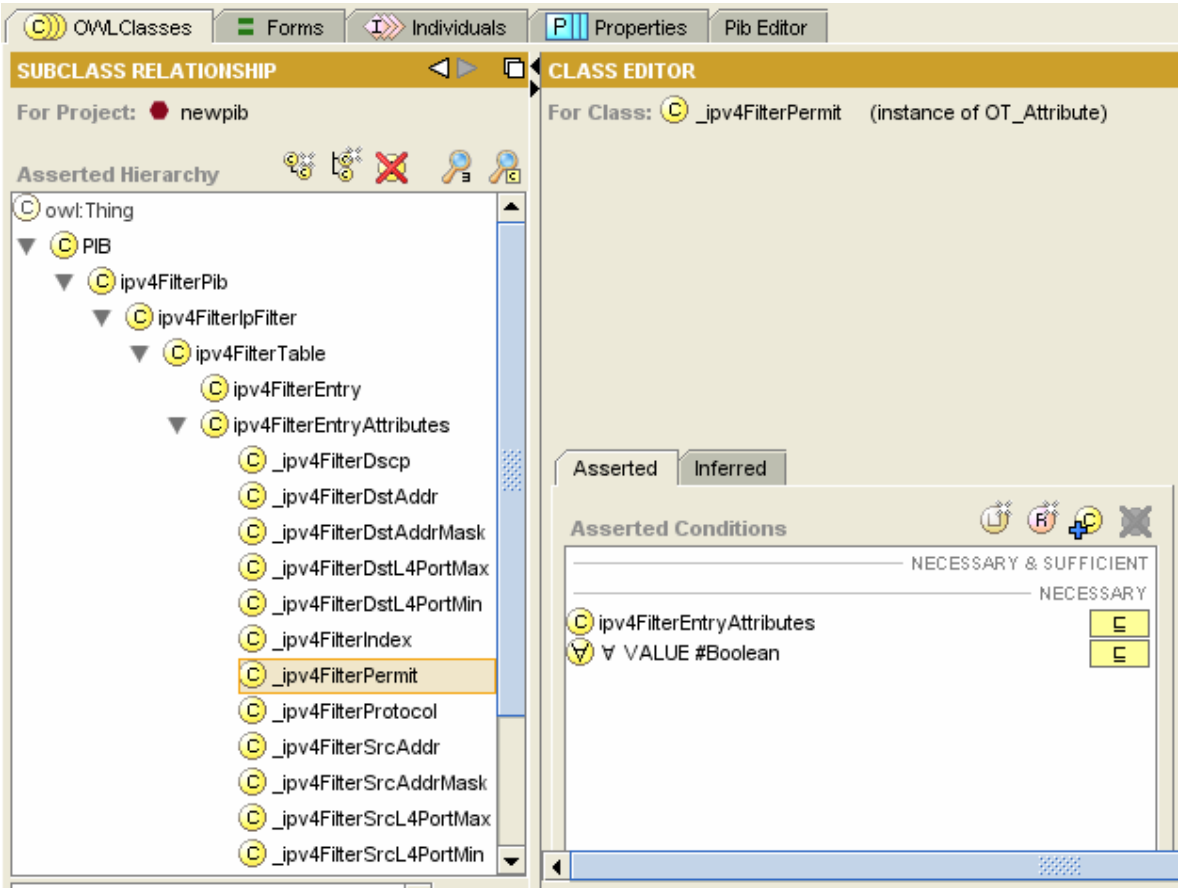

Figure 5.26 - PIB de filtrage en protégé

La représentation complète en format OWL de cette PIB de filtrage est définie dans la section D.2.2 de l'Annexe D et dont un extrait est illustré dans la Figure 5.27 suivante:

```
<OT_Table rdf:ID="ipv4FilterTable">
    <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.1.1</OID>
    <OT_Description rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
        Filter definitions.  A packet has to match all fields in a filter.
    Wildcards may be specified for those fields that are not relevant.
  </OT_Description>
        <rdfs:subClassOf>
         <PibMetaClass rdf:ID="ClassFiltrage">
                <rdfs:subClassOf>
            <ModuleIdentity rdf:ID="PibFiltrage">
            <rdfs:subClassOf rdf:resource="#PIB"/>
            <OID rdf:datatype=
         "http://www.w3.org/2001/XMLSchema#string">1.1.1</OID>
            </ModuleIdentity>
               </rdfs:subClassOf>
               <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.1</OID>
          </PibMetaClass>
        </rdfs:subClassOf>
        <OT_Syntax rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    SEQUENCE OF Ipv4FilterEntry  </OT_Syntax>
</OT_Table>
```

Figure 5.27 - Format OWL de la PIB de filtrage

Les politiques dans la PIB ne sont autres que les PRIs de celle-ci de manière générale ou encore des instances OWL de manière plus spécifique. Ainsi, on définit une politique de filtrage

en instanciant une entrée contenant toutes les propriétés qui lui ont été attribuées et ce, en vérifiant qu'elles sont bien du type qui leur a été spécifié dans les sous-classes de EntryAttributes. Les différentes entrées instanciées constituent les feuilles de la structure arborescente de la PIB.

Comme exemple d'application, la Figure 5.28 montre la définition sémantique d'une politique de filtrage sous forme d'une instance OWL et qui rejette tout paquet transmis de la machine ayant pour adresse 10.10.10.1 vers le réseau ayant pour adresse 192.168.2.0/24.

```
_ipv4FilterIndex rdf:ID="ipv4FilterIndex_r1">
    <VALUE rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</VALUE>
</_ipv4FilterIndex>
<_ipv4FilterSrcAddr rdf:ID="ipv4FilterSrcAddr_r1">
    <VALUE rdf:datatype="http://www.w3.org/2001/XMLSchema#string">10.10.10.1</VALUE>
</_ipv4FilterSrcAddr>
<_ipv4FilterDstAddr rdf:ID="ipv4FilterDstAddr_r1">
    <VALUE rdf:datatype= “http://www.w3.org/2001/XMLSchema#string>
        192.168.2.0
    </VALUE>
</_ipv4FilterDstAddr>
<_ipv4FilterDstAddrMask rdf:ID="ipv4FilterDstAddrMask_r1">
    <VALUE rdf:datatype=  "http://www.w3.org/2001/XMLSchema#string">
        255.255.255.0
    </VALUE>
</_ipv4FilterDstAddrMask>
<_ipv4FilterDscp rdf:ID="ipv4FilterDscp_Regle1">
    <VALUE rdf:datatype="http://www.w3.org/2001/XMLSchema#int">-1</VALUE>
</_ipv4FilterDscp>
<_ipv4FilterPermit rdf:ID="ipv4FilterPermit_r1">
    <VALUE rdf:datatype= “http://www.w3.org/2001/XMLSchema#boolean">
        False
    </VALUE>
</_ipv4FilterPermit>
<ipv4FilterEntry rdf:ID="ipv4FilterEntry_r1">
    <ipv4FilterDstAddr rdf:resource="#ipv4FilterDstAddr_r1"/>
    <ipv4FilterDstAddrMask rdf:resource= "#ipv4FilterDstAddrMask_r1"/>
    <ipv4FilterSrcAddr rdf:resource="#ipv4FilterSrcAddr_r1"/>
    <ipv4FilterPermit rdf:resource="#ipv4FilterPermit_r1"/>
    <ipv4FilterIndex rdf:resource="#ipv4FilterIndex_r1"/>
    <ipv4FilterDscp rdf:resource="#ipv4FilterDscp_Regle1"/>
</ipv4FilterEntry>
```

Figure 5.28 – Exemple d'une règle de filtrage en OWL

5.4.3. Ajout d'un Plugin à Protégé : Pib Editor

La structure de la PIB, arborescence utilisée pour la définition des politiques de gestion, a donc été convertie de ASN.1 en OWL. Ce modèle sémantique de la PIB a été validé par les deux exemples implémentés (Filtrage et Diffserv).

Mais afin de pouvoir utiliser ces définitions d'une façon plus pratique, il fallait pouvoir accéder à l'ontologie créée et de pouvoir modifier cette arborescence en connaissant juste la structure de la PIB, sans obligatoirement avoir des connaissances en Web Sémantique; d'où l'idée de l'implémentation d'une interface graphique dédiée à l'édition des PIBs et de l'intégration de cette interface au sein de l'éditeur d'ontologies *protégé*.

En effet, malgré les grands avantages des éditeurs d'ontologies comme Protégé, il n'est pas évident de créer les classes et les instances qui représentent la PIB. La possibilité d'ajout des extensions (*plugins* ou *widgets*) à *Protégé* nous a permis d'en créer une, sous la forme d'une interface graphique permettant de faciliter la création et la modification de la PIB. Cette *extension* n'est autre qu'un programme Java (*Protégé* étant basé sur Java). Cette interface permet à un utili-

sateur de saisir de nouvelles classes et de faire les liens implicites nécessaires avec les méta-classes appropriées. Elle propose l'ajout ou le choix de Modules, de Groupes, de Tables, d'entrées et d'Attributs en vue d'étendre ou de modifier la PIB. La Figure 5.29 présente l'aspect de cette interface telle qu'elle a été intégrée directement à *Protégé*.

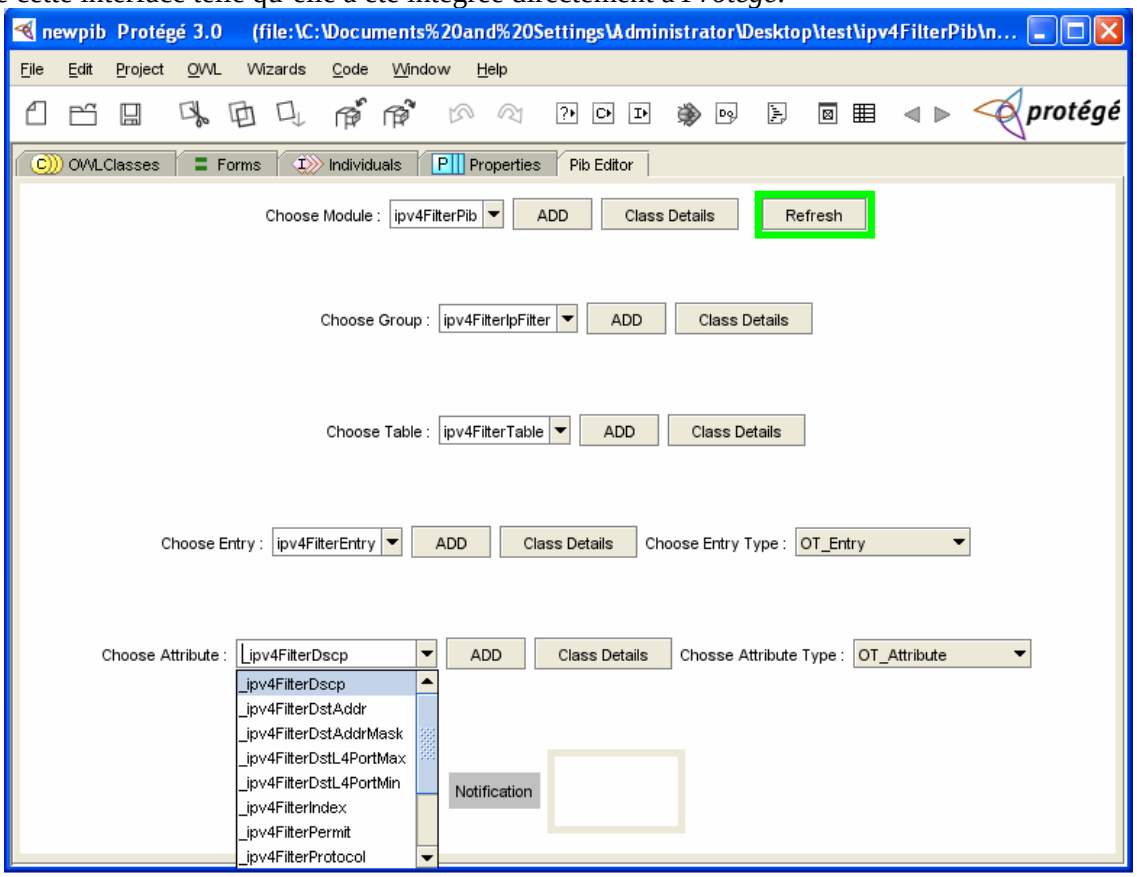

Figure 5.29- Widget PibEditor intégrée à protégé

5.5. APPLICATION DE TEST : GESTION PAR POLITIQUE D'UN PARE-FEU

Pour tester et valider notre proposition de gestion de politiques, nous avons implémenté une application de test que nous avons installée sur une plateforme Windows XP, où nous avons de même implémenté la PIB de filtrage en format OWL définie dans la section 5.4.2.3. Cette application de test consistait à mettre en place un PEP sous la forme d'un service actif au sein de la plateforme ASWA et jouant le rôle d'un pare-feu. Le rôle de ce PEP actif est de recevoir des décisions du PDP sous forme de requêtes COPS, de les traduire en règles de filtrage puis de configurer en conséquence le module de filtrage de Windows. Le filtrage se fera alors au niveau du noyau du système d'exploitation Windows.

En effet, Microsoft permet le filtrage des paquets entrants et sortants au niveau de la pile protocolaire TCP/IP d'un hôte Windows en intégrant le pilote de périphérique *IPFilterDriver* qu'on appelle module de filtrage ou *Packet Filter Driver* [150]. Ce dernier effectue le filtrage IP statique en se basant sur les critères suivants:

- Filtrage par interface, en entrée ou en sortie.
- Filtrage sur les adresses source et/ou destination.
- Filtrage sur les protocoles:
  - TCP et UDP (ports source et/ou destination)
  - ICMP (type et/ou code)
  - Tout protocole de niveau transport (numéro de protocole)
- Pseudo gestion des connexions TCP établies et de la fragmentation.

D'autre part, une interface de programmation (*Packet Filtering API)* [151] a été dévelop-

pée par Microsoft pour permettre la manipulation des filtres à partir d'une application.

Pour effectuer le filtrage au niveau du PEP, nous avons installé le package *PktFilter* [149]. *PktFilter* est un logiciel gratuit qui permet de configurer le module de filtrage IPv4 sur Windows 2000/XP/Server 2003. Il est constitué de deux programmes :

- pktfltSrv : un service Win32 qui configure le module de filtrage IPv4 en utilisant le *Packet Filtering API.*
- Pktctl : un utilitaire qui permet la saisie des règles de filtrage en utilisant une syntaxe bien déterminée. Il analyse chaque règle et l'envoie au service pktfltSrv à travers un tube nommé.

Dans notre implémentation (Figure 5.30), l'application des règles de filtrage sur le PEP passe par les étapes suivantes:

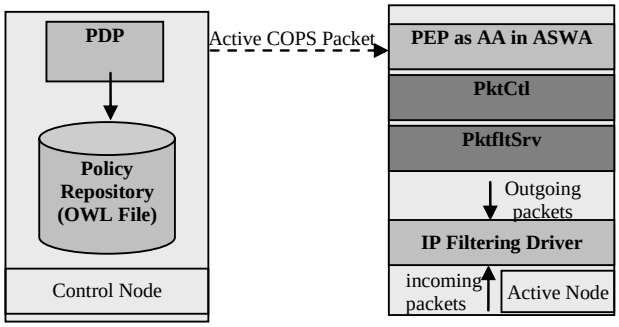

Figure 5.30 - Architecture de l'application de test

1. Le PEP se connecte au PDP qui gère le contenu de la PIB de filtrage.
2. Le PEP reçoit les règles de filtrage en format OWL encapsulées dans un message COPS-PR.
3. Le PEP extrait alors les règles de filtrage reçues à l'aide de l'analyseur VicSoft.RDF [152].
4. Les règles en format OWL sont converties en règles en format PktFilter et sauvegardées dans un fichier texte.
5. La configuration du module de filtrage de Windows se fait en appelant le programme pktctl. Le nom du fichier texte contenant les règles de filtrage en format PktFilter est passé comme argument sur la ligne de commande de ce programme.

Pour simplifier, on suppose que chaque règle contient seulement les attributs suivants (voir section D.2) : un index (*ipv4FilterIndex)*, une adresse destination (*ipv4FilterDstAddr*) et un attribut booléen nommé *ipv4FilterPermit* dont la valeur doit être *true* pour autoriser le trafic.

Comme exemple, on suppose qu'un administrateur réseau dans une société doit autoriser le trafic Internet durant les heures de travail de tous les jours ouvrables. Chaque matin à 08 heures, il doit autoriser le trafic Internet et à 16 heures il doit l'interdire. Nous réalisons ceci comme suit :

- Au démarrage, le PEP exécute la commande "pktctl –Fa" pour vider toutes les règles sur toutes les interfaces réseaux.
- Ensuite, le PEP se connecte au PDP et demande de télécharger les politiques initiales. Initialement, le PDP active la première politique en installant une PRI qui rejette tout le trafic Internet (PRID#1). Le PEP applique cette politique en exécutant la commande "pktctl –a block on eth0 all" pour rejeter tous les paquets entrants et sortants sur l'interface *eth0*.
- Au commencement d'une journée de travail, le PDP installe une PRI (PRID #2) qui accepte tout le trafic Internet. Le PEP applique cette politique en exécutant la commande "pktctl –a pass on eth0 all".
- A la fin d'une journée de travail, la première politique (PRID #1) est réinstallée pour rejeter de nouveau tout le trafic Internet.

Le message COPS de la Figure 5.31 illustre comment on autorise tout le trafic Internet.

```
<soap:Envelope xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsse=http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
 <soap:Header>
    <!--Entête du paquet actif-->
    <aswa:ASWA aswa:ProtocolID="COPS" xmlns:aswa="http://fi.usj.edu.lb/aswa">
        <ProtocolID>COPS</ProtocolID>      <!-- Active COPS -->
    </aswa:ASWA>
    <ASWAFirewallHeader>
        <srcaddr>34.34.34.3</srcaddr>
        <dstaddr>192.168.0.128</dstaddr>
        <srcport>1234</srcport>
        <dstport>80</dstport>
        <protocol>COPS</protocol>
        <UserGrp>ASWA</UserGrp>
    </ASWAFirewallHeader>
    <CopsHeader xmlns="http://tempuri.org/">
        <Vers_Flag>17</Vers_Flag>
        <Opcode>2</Opcode>                  <!--DEC -->
        <Client_Type>0x8888</Client_Type>  <-- IP Filtering -->
        <Length>118</Length>
    </CopsHeader>
    <wsse:Security>
        <wsu:Timestamp wsu:Id="Timestamp-f61e9b92-a5f8-4b0e-8bf2-e7807a7cf784">
            <wsu:Created>2005-05-31T04:20:00Z</wsu:Created>
            <wsu:Expires>2005-05-31T04:25:00Z</wsu:Expires>
        </wsu:Timestamp>
    </wsse:Security>
</soap:Header>
<soap:Body>
    <DecisionInstallResponse xmlns="http://tempuri.org/">
      <coarr>
        <COPSObject xsi:type= "DecisionObject">
            <Length>24</Length>
            <CNum>6</CNum>          <!-- Decision Object -->
            <CType>1</CType>                <!-- Decision Flags -->
            <contents>
                <command>1</command > <!--Install -->
                <flag>1</flag>
            </contents>
        </COPSObject>
        <COPSObject xsi:type= " PRID">
            <Length>19</Length>
            <CNum>1</CNum>          <!--PRID -->
            <CType>3</CType>                <!-- OWL Format -->
            <contents>
                <PRID>2</PRID >
            </contents>
        </COPSObject>
        <COPSObject xsi:type= "EPD">
            <Length>54</Length>
            <CNum>3</CNum>          <!--EPD -->
            <CType>3</CType>                <!-- OWL Format -->
            <!--The SEQUENCE of attributes-->
            <contents>
                <_ipv4FilterIndex rdf:ID="FilterIndex_Allow">
```

```
                    <VALUE rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
                        2
                    </VALUE>
                </_ipv4FilterIndex>
                <_ipv4FilterDstAddr rdf:ID="FilterDstAddr_All">
                    <VALUE rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                        *.*.*.*
                    </VALUE>
                </_ipv4FilterDstAddr>
                <_ipv4FilterPermit rdf:ID="FilterPermit_Allow">
                    <VALUE rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">
                        True
                    </VALUE>
                </_ipv4FilterPermit>
            </contents>
        </COPSObject>
    </coarr>
  </DecisionInstallResponse>
</soap:Body>
```

Figure 5.31 - Message COPS encapsulé dans SOAP pour autoriser tout le trafic Internet

5.6. REPRESENTATION SEMANTIQUE DES REGLES DE POLITIQUE

5.6.1. Introduction

Le modèle PCIM a pu être utilisé pour développer des langages de spécifications de règles de politiques ayant chacun leurs avantages et leurs inconvénients. Cependant c'est l'aspect de traduction de ces règles que nous avons abordé pour le choix d'un nouveau langage. En effet, les règles de politiques spécifiées au niveau business par l'administrateur doivent être traduites en politiques à installer ou à retrancher de la PIB. Ainsi, en considérant le modèle CIM (section A.5) et précisément la succession des niveaux dans ce modèle, nous notons l'importance de devoir passer par cette étape. Ce qui correspond en fait à passer du langage de spécification des politiques (PCIM, Ponder ou PFDL) au langage de représentation de la PIB, OWL dans notre cas.

Notre solution consiste à l'utilisation d'un langage qui permet d'économiser une étape qui peut paraître complexe quant à la différence entre les syntaxes de Ponder ou PFDL d'une part et OWL de l'autre. Ainsi, on préconise l'utilisation de SWRL (section B.3) qui permet d'une part la description de règles de politiques de manière très flexible et d'autre part l'utilisation d'une même ontologie pour la représentation des politiques et de leurs règles.

Comparé aux modèles PFDL et PCIM, SWRL a l'avantage de rester dans la même vision que celle utilisée pour la représentation de la PIB à l'aide de OWL. En effet, le fait que SWRL est une extension de OWL (donc de même sémantique), ceci permet d'utiliser le même type de langage pour la représentation de la PIB et la spécification des règles. Ainsi, en reprenant le modèle PCIM, on voit une avancé très importante puisque l'étape de traduction entre le langage de spécification et le langage de représentation de l'information (qui correspond à la PIB) est évitée. Tel est l'atout principal de notre solution par rapport à celles généralement utilisées.

5.6.2. Représentation des règles de politique en SWRL

Nous avons vu que les messages COPS-PR de configuration de la PIB peuvent être :

- Sollicités, dans ce cas le PEP envoie une demande au PDP qui répond par l'envoi de la décision ;
- Non sollicités, dans ce cas le PDP peut envoyer une décision mettant à jour la PIB du PEP.

L'utilisation de SWRL ne se fait que dans le cas de messages non sollicités puisque dans le cas de messages sollicités le PEP aura déjà reçu l'ensemble des règles à la suite de sa requête.

Une règle de politique est de la forme « **Si** *condition* **alors** *action* » (Figure A.10). Cette rè-

gle peut être représentée en SWRL par une relation d'implication de la forme :

antecedent ⇒ consequent

Supposons par exemple que l'administrateur veuille autoriser le trafic Internet entre l'instant T1 et l'instant T2. Pour ce faire, il doit installer la politique P1 dans la PIB du PEP à l'instant T1 et la retirer ensuite à l'instant T2. Cette règle va être introduite dans l'environnement de la manière suivante :

- La règle est d'abord définie en SWRL et stockée dans un fichier pouvant contenir d'autres règles.
- A l'instant T1 la règle est validée et le PDP force le PEP à installer la politique P1 en lui envoyant un message non sollicité qui, dans ce cas, peut contenir la commande *Install* et la propriété P1.
- Lorsque le PEP reçoit ce message, il installe cette politique dans sa PIB.
- A l'instant T2 la règle est également validée et le PDP force le PEP à retirer la politique P1 en lui envoyant un message non sollicité qui contient la commande *Remove* et la propriété P1.
- Le PEP reçoit le message et retire la politique de sa PIB.

La Figure 5.32 et la Figure 5.33 illustrent cela aux instants T1 et T2.

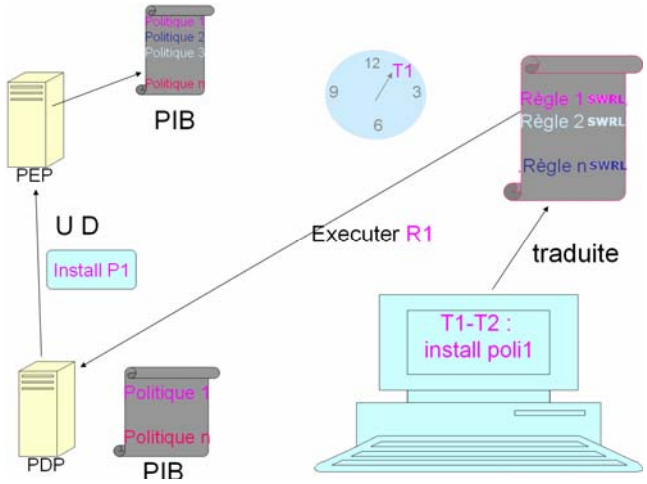

Figure 5.32 -Instant T1: Installation de la politique P1

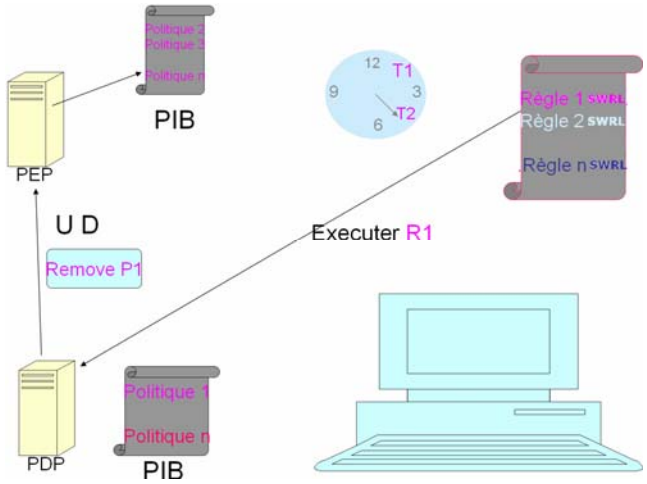

Figure 5.33 - instant T2: Retrait de la politique P1

La règle de politique du scénario précédent est définie par:
Si heure= T1 – T2 **Alors** autoriser le trafic Internet.

En SWRL, cette règle devient :
IsTime(?x) ^ Equal(?x, T1) ⇒ Send(P1)
IsTime(?x) ^ Equal(?x, T2) ⇒ Remove(P1)

Une interface graphique qui joue le rôle d'une console administrative a été développée. Cette interface permet à l'administrateur l'édition, la modification et la consultation des règles de politique ainsi que leur sauvegarde en format SWRL.

Le format SWRL des règles de politique du scénario précédent est illustré dans la Figure 5.34:

```
<swrl:Imp rdf:ID="Install_P1">
  <swrl:body>
  <swrl:datavaluedPropertyAtom swrl:property= send >
    <swrl:Variable swrl:name=  basecapsentry1 />
     <swrl:DataValue
          rdf:datatype= &xsd;boolean> true
    </swrl:DataValue>
   </swrl:datavaluedPropertyAtom>
  </swrl:body>
<swrl:head>
 <swrl:datavaluedPropertyAtom swrl:property= equal >
          <swrl:Variable swrl:name= pc_clock />
          <swrl:DataValue
                  rdf:datatype= &xsd;string> 08:00:00
          </swrl:DataValue>
   </swrl:datavaluedPropertyAtom>
  </swrl:head>
</swrl:Imp>
<swrl:Imp rdf:ID="Remove_P1">
   <swrl:body>
    <swrl:datavaluedPropertyAtom
          swrl:property=remove >
        <swrl:Variable swrl:name= basecapsentry1 />
<swrl:DataValue rdf:datatype= &xsd;boolean> true </swrl:DataValue>
   </swrl:datavaluedPropertyAtom>
  </swrl:body>
<swrl:head>
<swrl:datavaluedPropertyAtom  swrl:property= equal >
          <swrl:Variable swrl:name= pc_clock />
<swrl:DataValue rdf:datatype= &xsd;string> 14:30:00
</swrl:DataValue>
</swrl:datavaluedPropertyAtom>
   </swrl:head>
</swrl:Imp>
```

Figure 5.34 - Exemple de règles de politique en SWRL

On voit donc que le grand avantage dans le fait que SWRL soit une extension de OWL, réside dans l'utilisation du même type de langage pour la représentation de la PIB et la spécification des règles. Ceci permet d'éviter l'étape de traduction entre le langage de spécification des règles et le langage de représentation de l'information (qui correspond à la PIB).

5.7. CONCLUSION DU CHAPITRE

Cette architecture a été implémentée à l'aide de Web Services actifs grâce aux facilités d'utilisation de SOAP pour l'encapsulation des messages COPS, pour être validée au sein de notre plateforme de déploiement de services ASWA.

ASWA ayant été développée pour supporter tout type d'applications, s'est avérée être très

intéressante pour la gestion à base de politiques surtout avec l'utilisation des ontologies de type OWL pour la représentation des ressources et des politiques.

La partie la plus délicate concernait la manière de représenter les informations relatives aux ressources et plus précisément celles de la PIB utilisée dans COPS-PR. Ceci nous a poussés à penser à des langages évolués afin de permettre à cette représentation d'intégrer plus de sémantique et d'intelligence et de donner aux noeuds la possibilité de jouer un rôle plus important dans le processus de traitement des flux selon certains comportements prédéfinis.

Le Web Sémantique est donc un moyen de représenter les ressources afin que les machines puissent les comprendre et les manipuler intelligemment. Il est donc très utile d'utiliser cette précieuse qualité dans la gestion des politiques des réseaux afin de la rendre plus dynamique.

OWL est à la base de cette représentation des informations de manière formelle à l'aide de classes et de propriétés selon une certaine hiérarchie. L'évolution SWRL permet, avec les mêmes performances, de spécifier la structure des règles de politiques à utiliser.

Nous avons expliqué dans ce chapitre comment nous avons pu utiliser les caractéristiques des ontologies associés au langage Web sémantique dans la représentation de SPPI et la définition de différentes PIBs, ceci dans le cadre de l'implémentation de techniques de gestion à base de politiques utilisant COPS-PR pour la communication entre les nœuds du réseau (PEP) et l'élément décisionnel de celui-ci (PDP).

Tout ceci n'est qu'un premier pas vers la mutation des gestions de politiques en sémantique...

Enfin, il est intéressant de comparer la solution proposée dans ce chapitre aux architectures actuelles :

- La représentation des informations de la PIB à partir de la spécification SPPI se fait dans notre solution de manière sémantique avec OWL alors que les représentations des solutions actuelles restent syntaxiques en ASN.1.
- Des Méta-Classes OWL définissent la SPPI alors que celle-ci reste sous forme de Macros ASN.1 dans les standards.
- Notre modèle de description des politiques est plus structuré car il utilise des classes OWL, ces politiques étant encore spécifiées avec les PRCs en ASN.1 dans la PIB standard.
- Les politiques dans notre solution sont des instances des classes OWL au lieu de PRIs dont la représentation et le traitement dépendent de l'implémentation.
- L'annuaire de stockage des politiques (*Policy Repository*) est défini par une ontologie OWL, alors qu'il est la plupart du temps en LDAP, donc non sémantique.
- Les règles de politiques, et c'est à ce niveau que l'avantage de notre solution est le mieux perçu, sont définis à partir de SWRL (extension d'OWL), alors que les langages de spécifications sont en général différents des langages de représentation dans les implémentations actuelles. Ceci marque l'homogénéité de l'infrastructure choisie.

Chapitre 6

CONCLUSION ET PERSPECTIVES

La gestion des réseaux présente plusieurs problèmes dont les plus importants sont le **déploiement des services de gestion**, le **dimensionnement,** la **sécurité**, les **interactions complexes entre les composants appropriés** et enfin la **représentation ainsi que l'interprétation des politiques** qu'il faut appliquer au niveau des nœuds du réseau.

La gestion des réseaux à base de politiques (PBNM) tend à être de nos jours l'une des solutions les plus intéressantes pour une gestion et un contrôle d'accès aux ressources par les utilisateurs d'un réseau: L'administrateur se charge de définir des règles de politiques de manière centralisée au niveau du PDP et celles-ci pourront être communiquées aux différents nœuds PEP (à la demande ou non) à l'aide du protocole de transaction COPS.

D'autre part, le contrôle dynamique et intelligent dans la gestion des réseaux commence à intéresser les chercheurs de plus en plus, bien que ce soit toujours un sujet peu exploré dans la littérature.

Les travaux effectués dans le cadre de cette thèse montrent que l'utilisation du modèle à trois plans que nous avons défini et qui se base sur la synergie entre les 4 paradigmes: Réseaux à base de politique, Réseaux actifs, Web services et Ontologie, peut être une solution des plus intéressantes au problème de contrôle dynamique et intelligent dans la gestion des réseaux.

Pour illustrer l'intérêt de notre contribution, nous résumons dans la suite les réponses que fournit notre modèle aux différents problèmes cités ci-dessus:

- Au niveau du **déploiement des services de gestion**, les plateformes de réseaux actifs actuelles proposent des solutions architecturales offrant plus de flexibilité. Mais , ce qui nous a incités à proposer une nouvelle architecture à base de Web services sécurisée (ASWA), est le fait que la plupart de ces architectures de réseaux actifs restent des systèmes fermés. A ce niveau, et grâce au choix technologique que nous avons adopté dans l'utilisation des Web services pour l'implémentation des différents composants de notre plateforme, et du protocole SOAP pour l'échange de messages, ASWA permet l'interopérabilité avec d'autres architectures de réseaux actifs. D'autre part, ASWA utilise l'approche du nœud actif pour mieux contrôler le chargement de code, vu que cette approche offre des mécanismes de chargement et d'exécution de code séparés.
- Au niveau du **dimensionnement**, la notion de domaine introduite au sein de l'architecture ASWA permet une meilleure structuration et scalabilité du réseau : Chaque domaine regroupe un nœud de contrôle et plusieurs nœuds actifs. Chaque nœud de contrôle est autonome dans son domaine; il gère et contrôle les nœuds actifs de son domaine. Cette structuration en domaines limite la portée des services à un sous-ensemble des équipements, ce qui permet de favoriser le passage à l'échelle (scalabilité).
- Au niveau de la **sécurité**, ASWA profite de l'utilisation du protocole SOAP dans les échanges de messages, pour insérer des mécanismes de sécurité tel que l'authentification par secrets

pré partagés ou par certificats. ASWA utilise également le protocole HTTPS (HTTP over SSL) pour sécuriser le déploiement. Dans ASWA, les mécanismes de sécurité sont implémentés conformément à la spécification WS-Security des Web services qui permet d'améliorer la sécurité granulaire des systèmes basés sur le protocole SOAP. Cette spécification définit l'aptitude à échanger des justificatifs d'identification (par exemple, les certificats X.509 ou les tickets Kerberos), de vérifier l'intégrité d'un message et de mettre en vigueur la confidentialité de ce message. Grâce à la notion de filtres de sécurité (« Security Filter ») du pipeline WSE, l'utilisation des services de sécurité est transparente pour le développeur des applications actives ainsi que pour l'utilisateur de ces applications.

- Au niveau des **interactions entre les composants appropriés de la gestion de réseau** dans la technologie PBNM, ASWA permet de gérer leur complexité à trois niveaux :
  - D'une part, l'extensibilité dynamique de l'architecture de la gestion est assurée par les Web services. A partir d'un Web service, on peut construire des services ou des applications Web dérivés des précédents. En effet, le point crucial des Web services est l'imperméabilité entre les clients et les serveurs qui ignorent tout de l'implémentation, dite privée, des opérations respectives de leurs correspondants et ne se connaissent qu'à travers des descriptions publiques.
  - D'autre part, notre plateforme offre une solution assurant interopérabilité et transparence des interactions entre les entités de PBNM dans un environnement hétérogène et ceci grâce à l'encapsulation des messages COPS dans les messages SOAP.
  - Enfin, comme ASWA permet un déploiement actif de services, ceci permet de résoudre plusieurs des problèmes inhérents dans la technologie actuelle du PBNM. Ce dernier peut profiter des facilités des réseaux actifs pour proposer une solution de gestion et de contrôle flexible. En effet, les réseaux actifs permettent aux services de définir leurs propres mécanismes de traitement des données dans les nœuds intermédiaires et avoir accès à certaines informations sur ces nœuds. Un opérateur peut alors installer des mécanismes décentralisés et automatiques qui permettent de traiter localement des événements tels que la reconfiguration, les pannes et le déploiement de nouveaux services. Ceci permet facilement de traiter des fonctionnalités dynamiquement installées sur les noeuds actifs; aussi bien que contrôler les besoins croissants pour la personnalisation que les consommateurs de service imposent sur les réseaux actifs.
- Au niveau de la **représentation** et de **l'interprétation** des ressources et des règles de politiques ainsi qu'au niveau de leur**,** l'utilisation des ontologies de type OWL au niveau du plan de signalisation s'est avérée être très intéressante dans le cadre de la gestion à base de politiques. En effet, les langages d'ontologie peuvent être utilisés pour améliorer l'expressivité de la sémantique des spécifications de l'information de la gestion, rendant ainsi la gestion par politique plus dynamique. L'évolution SWRL permet, avec les mêmes performances, de spécifier la structure des règles de politiques à utiliser. Ceci permet d'éviter l'étape de traduction entre le langage de spécification des règles et le langage de représentation de l'information.

L'application de notre modèle de contrôle intelligent à trois plans ouvre de larges perspectives.

- L'une des perspectives à court terme peut se manifester par le projet d'intégration de notre architecture à trois plans avec le projet « *Déploiement contrôlé des services* » lancé par le laboratoire L2TI. Ce projet propose un modèle pour recenser tous les équipements hôtes potentiellement aptes au déploiement d'un service. Ce recensement est suivi d'une sélection des points d'installation, cette sélection étant couplée avec diverses stratégies d'optimisation et/ou politique. Enfin, une base de données contenant l'état actuel de tous les équipements (les points d'installation) est déployée et actualisée. Dans le projet d'intégration qu'on propose, ASWA se placera au niveau du plan de déploiement, tandis que les stratégies d'optimisation et/ou les politiques seront définis grâce à une ontologie OWL au niveau d'un plan de signalisation.

- D'autres scénarios d'application peuvent être proposés et implémentés. Nous citons par exemple un scénario dans lequel le plan de service est un réseau paritaire (« peer to peer »). Le déploiement des services dans ce réseau serait confié à ASWA. Le plan de signalisation, dans ce réseau, aurait alors comme tâche d'assurer une distribution équitable des ressources aux différents pairs.
- Enfin, au lieu d'ajouter la sémantique au protocole COPS, nous proposons de définir un nouveau protocole de gestion purement sémantique basé sur OWL. Dans ce protocole, une requête peut avoir par exemple la forme : *Recherche des règles de politiques à appliquer sur un pare-feu utilisant iptables sur une machine Linux ayant une carte ethernet à l'entrée du LAN des étudiants*. Ceci est possible, vu qu'avec les ontologies, on peut traiter les informations et non uniquement les présenter. Les opérations de gestion pourraient ainsi se baser sur des rôles, et non plus sur des entrées et des attributs définis dans la PIB, tel que préconisée par l'IETF. Ceci ouvrirait la voie à de nombreuses évolutions permettant de gérer les équipements selon leurs activités au sein du réseau.

Annexe A

GESTION DES RESEAUX PAR POLITIQUE

A.1. INTRODUCTION

De nos jours, le grand nombre d'utilisateurs des réseaux devient de plus en plus exigeant quand à la qualité du service auquel ils veulent avoir accès. Le service Best-Effort se voit être en voie de disparition du fait des grands enjeux commerciaux qui régissent le monde de l'informatique et des télécommunications. Il s'agit à présent de s'intéresser de plus en plus à la qualité de service, la sécurité, la disponibilité, etc.

Des solutions apparaissent régulièrement pour répondre aux contraintes liées à ces nouvelles demandes. Cependant, l'évolution exponentielle des réseaux révèle de nombreuses limitations des techniques de gestion habituelles en termes de facteur d'échelle et d'efficacité pour ces nouvelles techniques. L'abondance de ces nouvelles technologies comportant des échanges de flux d'informations importants et dynamiques, d'applications de plus en plus distribuées et donc de systèmes hétérogènes pose le problème majeur de la gestion et de l'administration de l'ensemble des équipements et systèmes d'informations mis en jeu. Il se doit donc d'avoir recours à des outils de gestion autonomes et dynamique.

C'est dans cette optique que fut élaboré le paradigme des réseaux à base de politiques pour la gestion et le contrôle des ressources réseaux. Ils sont couramment appelés PBNM [123] pour *Policy-Based Network Management*. Bien qu'originellement destinés à être déployés dans des réseaux pour gérer la qualité de service, ils ont été conçus pour être adaptable à d'autres besoins comme par exemple la sécurité ou encore l'adaptation de contenu. Le standard des PBNM préconise l'utilisation du protocole COPS [43] (*Common Open Policy Service*) pour l'échange des messages entre les entités de l'architecture. L'infrastructure comporte principalement des PEPs qui sont les nœuds clients ayant pour but d'exécuter un certain nombre de politiques concernant les différents domaines d'applications (QoS, sécurité …) et un serveur PDP qui se charge d'imposer aux PEPs ces règles de politiques à appliquer. Des connexions TCP s'ouvrent entre PEPs et le PDP pour l'échange des informations.

A.2. L'ARCHITECTURE POLICY-BASED NETWORKING (PBN)

On parle de gestion par politique ou PBM pour Policy-Based Management, technologie émergente qui présente de nombreuses différences par rapport aux anciennes techniques comme SNMP [124] qui concernait en grande partie la détection et réparation des problèmes dans un réseau. L'approche SNMP est efficace pour modéliser les ressources et avoir une vision de leur état, mais elle ne permet pas de gérer leur comportement. C'est là qu'est l'intérêt d'une gestion par politique: munir les ressources de comportements fondés sur la définition d'une politique de gestion. En effet, la nouvelle optique consiste à considérer le réseau fonctionnel et à permettre à l'administrateur de le régler pour atteindre certains objectifs définis à l'avance.

D'une manière très générale, toute mise en œuvre d'une architecture de gestion par politiques doit définir :

- Un modèle d'information pour les éléments du réseau, les services, les réseaux et les clients du réseau.

- Un langage de spécification de politiques qui puisse représenter les besoins d'une manière indépendante de leur implémentation.
- Une structure supportant le facteur d'échelle d'administration, de gestion, de distribution et de vérification de politiques.
- Un moyen de traduire les spécifications de politiques en commandes de configurations compréhensibles par un équipement.

Le groupe RAP [42] (*Ressource Allocation Protocol*) de l'IETF [126] est à l'origine d'une architecture générale d'implantation de gestion de réseau par politique, en accord avec les quatre besoins fonctionnels définis précédemment et qui se scinde en deux niveaux:

– un niveau *business* : qui représente le niveau fonctionnel des politiques. Il est muni d'un modèle d'information et d'un langage intelligibles et indépendants de l'implémentation. Il permet ainsi l'administration et la gestion des politiques à mettre en œuvre.
– un niveau *technologique* : qui vérifie, distribue et applique les politiques définies au niveau business. Il comporte un modèle d'information, un protocole de communication et une structure hiérarchique.

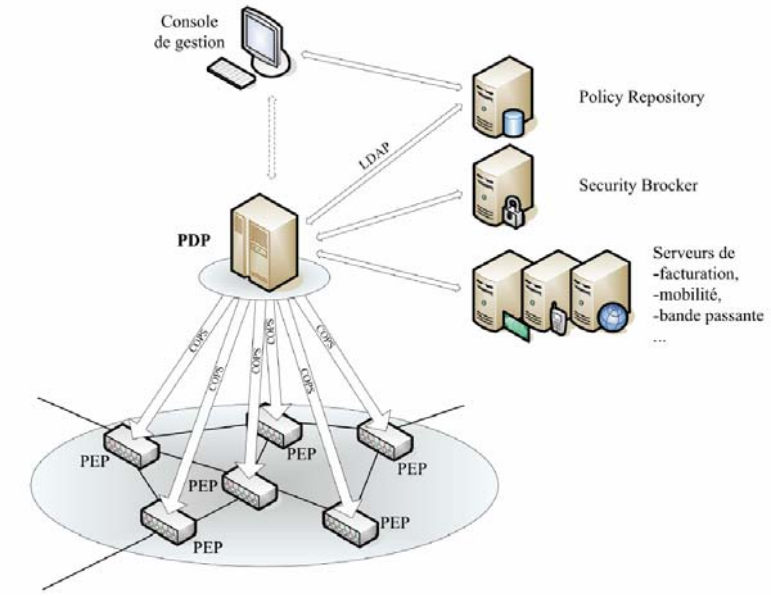

Figure A.1 – Entités fonctionnelles d'une architecture PBN

En termes d'entités fonctionnelles cette architecture se caractérise par sa simplicité puisqu'elle ne regroupe que les quatre composants suivants (Figure A.1) :

- Les nœuds d'un réseau sont appelés PEPs (*Policy Enforcement Point*). Ce sont ces entités qui appliquent les politiques pour gérer les flux des utilisateurs. Elles appliquent les décisions dictées par le PDP (*Policy Decision Point*) qui est un élément décisionnel ayant une vue globale du réseau. Un PEP est en général un routeur, commutateur, un pare-feu ou encore un terminal. L'intérêt primordial pour un PEP est d'être le plus facilement accessible et paramétrable de sorte à permettre au PDP de le configurer sans problèmes.
- Le PDP, que l'on peut aussi appeler serveur de politiques, est le point central qui doit décider des politiques à appliquer dans le réseau. Il est défini comme une entité logique prenant des décisions politiques pour elle-même ou pour d'autres éléments réseau (les PEPs) qui demandent ses décisions. Pour cela, le PDP recherche les informations dont il a besoin dans des serveurs qui peuvent être locaux (habituellement) ou encore distants. Ces serveurs gèrent chacun un domaine utile au PDP pour ses décisions. Parmi eux on trouve un serveur de gestion de sécurité, un serveur de mobilité, un serveur de facturation, un serveur de bande passante, et le plus important, un serveur d'annuaire des politiques appelée *Policy Repository*.
- Le *Policy Repository* stocke les règles de politique que le PDP va rechercher pour les appliquer aux PEPs (on préconise LDAP [127] pour l'accès à l'annuaire).

- Le dernier composant de l'architecture est une console administrative appelée *Policy Management Tool*, interface utile à l'administrateur pour éditer, modifier et consulter les règles s'appliquant au réseau en agissant sur le PDP ou le serveur d'annuaire.

A.3. COPS (COMMON OPEN POLICY SERVICE)

A.3.1. Généralités

Ce protocole, développé par l'IETF, permet le transport des demandes de configuration des PEPs et des réponses envoyées par le PDP concernant les politiques à appliquer. COPS [43] présente une approche révolutionnaire pour la gestion proactive d'équipement de réseau. Il a été développé en comparaison aux protocoles traditionnels de gestion de réseau tel que SNMP [124], lequel s'est avéré incapable de supporter efficacement une gestion de politique réseau. La pile du protocole COPS peut être conceptuellement divisée en 3 couches distinctes: le protocole de base, les directives d'usage du type client, et la représentation de données de politique. Ces trois couches ainsi que beaucoup d'autres avantages offerts par COPS par rapport à des protocoles traditionnels des gestions de réseau, permettent à COPS de s'adapter tout à fait à l'environnement de gestion de politique réseau.

COPS est un protocole très flexible de type requête/réponse fondé sur TCP qui consiste à l'établissement de connexion du PEP vers le PDP pour l'envoi de requêtes de politiques, la réception des décisions qui les concernent et optionnellement, l'envoi de rapports au PDP sur le déroulement des opérations. Le PEP et le PDP peuvent à tout moment mettre à jour respectivement une requête et une décision.

Ce protocole supporte deux modèles de gestion de politique qui sont:

- COPS-RSVP [128] ou le modèle de signalisation (l'*Outsourcing*). Dans ce modèle l'utilisateur et l'administrateur du réseau ne communiquent pas directement (il n'y a pas de lien préalablement tissé entre eux) puisque le PEP envoie une requête à son PDP qui lui autorise ou non l'établissement de la connexion. Le protocole utilisé pour gérer la QoS dans le modèle d'*outsourcing* est RSVP [69]. La communication se fait généralement comme suit: le client effectue une demande de connexion à un réseau auquel il n'est pas abonné. Cette demande appelle une décision concernant l'accès au réseau : l'opérateur accepte ou non la connexion du client. Le client doit donc faire une requête au réseau. Celle-ci arrive au nœud d'entrée du réseau qui la dirige vers le PDP grâce à une requête COPS. Une fois la décision prise par le PDP, une réponse transitant par le protocole COPS indique au nœud d'entrée si la demande RSVP est acceptée ou non. En cas d'acceptation, la requête RSVP peut continuer son chemin pour ouvrir une route satisfaisant la demande de l'émetteur.
- COPS-PR [44] ou le modèle de provision *(Provisioning*). Dans ce modèle un contrat, le SLA (*Service Level Agreement*), est établi entre l'administrateur et l'utilisateur du réseau pour définir la qualité de service attribuée à ce dernier. Les politiques définies dans ce contrat sont saisies par l'administrateur grâce à une console et sont alors enregistrées au niveau de la *Policy Repository*. Le *provisioning* est souvent associé à la technique DiffServ [135] dans laquelle les flots sont classifiés en différentes classes. La négociation SLS (Service Level Specification), partie technique du SLA, donne lieu à l'introduction des politiques dans la base de données de l'opérateur, politiques qui devront être appliquées dès que le client présentera un flux à l'entrée du réseau. Les politiques sont distribuées en temps réel par le PDP. Ce dernier décide si la politique doit être installée en permanence dans les PEPs traversés par le client ou non, suivant les caractéristiques techniques de la demande du client. Si les politiques sont implantées directement dans les nœuds, le PDP les envoie au travers d'une commande COPS. Les nœuds d'accès sont capables de traiter en temps réel ces demandes. C'est ce mode *provisioning* qui sera implémenté dans notre cas.

Il ne faut pas oublier de noter enfin que le protocole peut être étendu pour l'administration d'une grande variété de protocoles de signalisation ainsi que la configuration des équipements réseau. De même, il peut fournir un niveau de sécurité en assurant une authentification et une in-

tégrité des messages. Eventuellement, COPS peut utiliser certains protocoles existants pour la sécurité comme IPSEC [136] pour authentifier et sécuriser le canal de communication entre PDP et PEP.

A.3.2. Messages COPS

COPS définit un certain nombre de messages qui sont échangés entre un PEP et le PDP comme les messages d'ouverture et de fermeture des connexions, les messages transportant les politiques ou encore les messages d'erreurs.

L'encapsulation protocolaire est représentée dans la Figure A.2:

Figure A.2 - Encapsulation protocolaire

Un message est formé d'un entête suivi d'une séquence valide d'objets COPS. L'entête présente des informations sur l'identification du message, la taille de celui-ci, certains drapeaux ainsi que le type du client ou client-type (puisque l'une des caractéristiques de COPS est la possibilité de supporter une multitude de types de client pour l'ensemble des domaines de la gestion à base de politiques comme la sécurité, la Qualité de service (COPS-RSVP), le Contrôle d'admission, la mobilité (COPS-MU), et la comptabilité).

La Figure A.3 montre le format d'un message COPS et décrit plus particulièrement l'entête commun à tous les messages COPS:

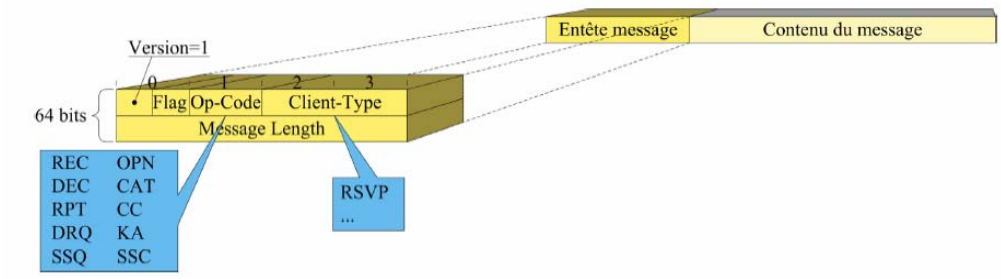

Figure A.3 – Format d'un message COPS

- **Version**: Indique la version du protocole COPS, la version courante est 1.
- **Reserved**: Ce champ est réservé, et doit être mis à 0.
- **S**: Solicited Message Flag, il est mis à 1 quand le message est sollicité par un autre message COPS.
- **Opcode**: utilisé pour identifier l'opération effectuée. La Table A.1 indique toutes les valeurs que peut prendre ce champ :

OpCode	Description
1	REQ, Request
2	DEC, Decision
3	RPT, Report State
4	DRQ, Delete Request State
5	SSQ, Synchronize State Request
6	OPN, Client-Open
7	CAT, Client-Accept
8	CC, Client-Close
9	KA, Keep-Alive
10	SSC, Synchronize Complete

Table A.1 - Codes opération d'un message COPS

- **Client type**: Identifie le type de client de politique. L'interprétation de tous les objets encapsulés se fera relativement à cet identificateur. Les types de clients sont considérés comme des extensions au protocole COPS de base puisqu'ils définissent des détails sur le format et la sémantique des données de configuration échangées entre les PEPs et le PDP. Ainsi on trouve l'avantage en COPS de pouvoir l'utiliser pour un échange de données d'un client de type

quelconque sans avoir à connaître la sémantique des données échangées. La Table A.2 indique quelques valeurs que peut prendre ce champ:

ClientType	Description
0x0001	RSVP
0x0002	DiffServ QoS
0x8001 - 0x8004	IP Highway
0x8005	Fujitsu Application Server Software Division
0x8006	HP OpenView PolicyXpert
0x8007	HP OpenView PolicyXpert COPS-PR PXPIB
0x8008	PacketCable Dynamic Quality of Service
0x8009	COPS usage for 3GPP GO interface

Table A.2 - Types des clients COPS

- **Message length**: Ce champ représente la taille du message en octets. Il prend en compte l'entête ainsi que tous les objets encapsulés. A noter que les messages COPS doivent être alignés sur des intervalles de 4 octets.
- **Data** : Contient tous les objets COPS encapsulés.

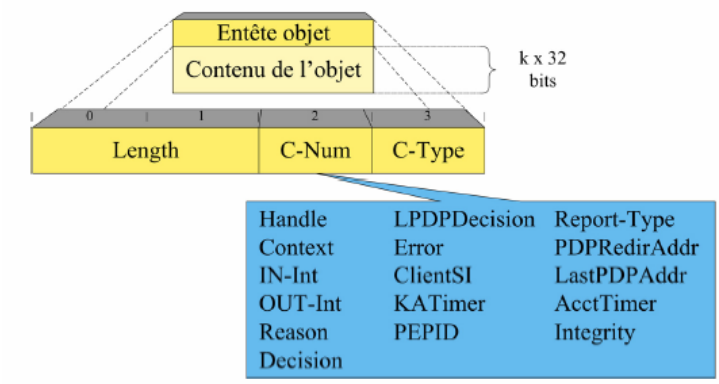

Figure A.4 - Entête d'un objet COPS

Ces objets sont constitués d'un entête au format identique (Figure A.4) suivi du contenu de l'objet formé par un ou plusieurs mots de 32 bits. L'entête d'un objet est formé des champs suivants:

- **Length**: Longueur de l'objet en octets.
- **C-Num (ou Class)**: Classe de l'objet. La Table A.3 indique les différentes classes utilisées par COPS:

Class	Description
1	Handle
2	Context
3	In Interface
4	Out Interface
5	Reason Code
6	Decision
7	LPDP Decision
8	Error
9	Client Specific Info
10	Keep-Alive Timer
11	PEP Identification
12	Report Type
13	PDP Redirect Address
14	Last PDP Address
15	Accounting Timer
16	Message Integrity

Table A.3 - Classes des objets COPS

– **C-Type (ou Subclass)** : Un champ sur 8 bits représentant la sous-classe.

La Figure A.5 détaille le format complet d'un message COPS :

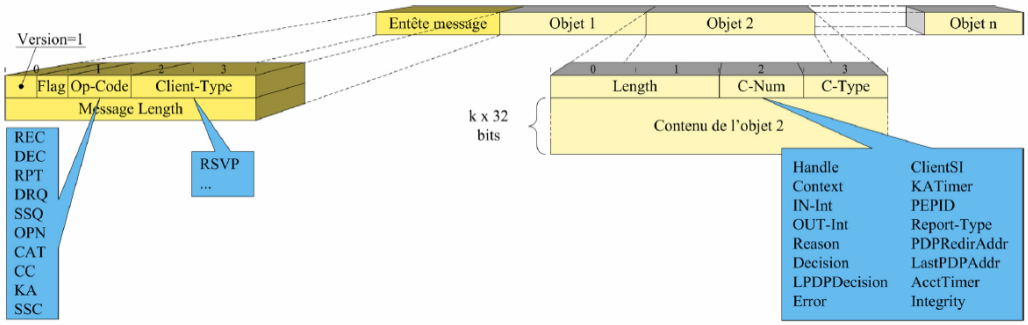

Figure A.5 - Format détaillé d'un message COPS

A.3.3. Exemple d'échange protocolaire

Etudions à présent un exemple d'échange protocolaire généralement exécuté. L'échange le plus courant de messages entre PEP et PDP est représenté dans la Figure A.6 et détaillé à la suite:

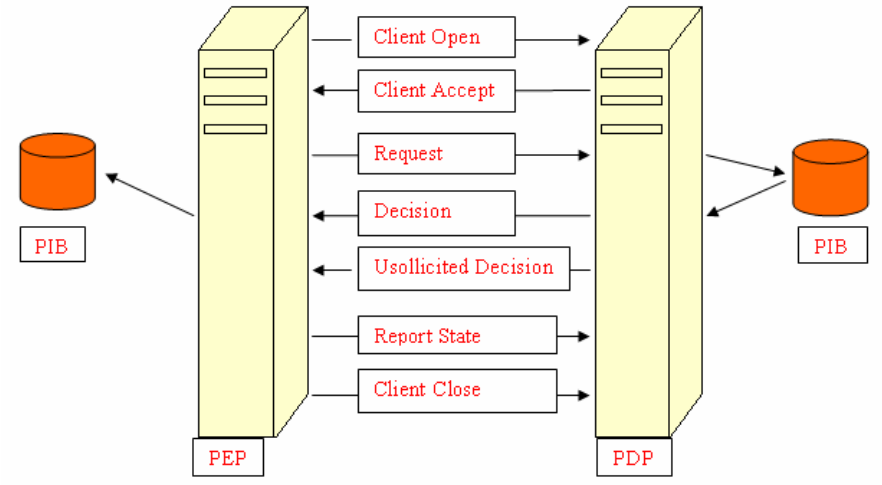

Figure A.6 - Echange de messages COPS

– **Client Open**: Ce message est transmis par le PEP pour demander l'ouverture d'une connexion au PDP en lui indiquant le type du client, exemple: COPS-PR, COPS-RSVP…. Ce message contient obligatoirement l'identité du PEP dans l'objet PEPID en plus de l'entête commune. Les champs entre crochets sont optionnels.

<Client-Open> : : = <CommonHeader> <PEPID> [<ClientSI>] [<LastPDPAddr>]

– **Client Accept**: Ce message est envoyé au PEP par le PDP pour confirmer l'ouverture de la connexion grâce à l'objet *Keep-Alive Timer*. Le PDP indique au PEP l'intervalle de temps maximum à respecter entre deux messages d'indication de présence.

<Client-Accept> : : =<CommonHeader> <KA Timer> [<ACCT Timer>] [Integrity]

Si le PDP refuse la connexion, il renvoie au PEP le message *Client-Close*, qui sera décrit par la suite.

– **Request**: Lorsque la connexion est ouverte, le PEP peut ainsi envoyer une requête au PDP contenant des informations internes concernant le PEP, exemple: taille maximale de ses queues, sa capacité, etc. Une requête est identifiée par un numéro ou *Client Handle*. Un client peut envoyer plusieurs requêtes de configuration chacune caractérisée par son *Client handle*, et donc pourra recevoir plusieurs contextes de configuration, une seule pouvant cependant être actif en même temps. Deux objectifs sont connus pour ce message :

o Ce message peut représenter une demande de requête issue du PEP et destinée au PDP.

Une configuration sera alors générée par le PDP pour être appliquée par le PEP. Dans ce cas c'est une décision sollicitée (*Solicited Decision*) puisque le PEP l'a expressément demandé.

- Ce message peut également ouvrir un canal qui permet au PDP d'envoyer d'une manière asynchrone au PEP, des décisions de politiques et cela lorsqu'il en voit la nécessité. Ces décisions peuvent être une mise à jour de la configuration déjà mise en place sur le PEP ou même l'envoie d'une nouvelle. On parle dans ce cas de Decision Non Sollicitée ou *Unsolicited Decision* avec COPS-PR.

```
<Request Message> : : = <Common Header><Client Handle> <Context>
                        [<IN-Int>] [<OUT-Int>] [LPDPDecision(s)] [Integrity]
< LPDPDecision (s) : : =  [<Context>]
                        [<LPDPDecision : Flags>]
                        [< LPDPDecision : Stateless Data>]
                        [< LPDPDecision : Replacement Data>]
                        [< LPDPDecision : ClientSI Data>]
                        [< LPDPDecision : Named Data>]
```

– **Decision** : En réponse au message de requêtes, le PDP envoie un message de décision qui contient les politiques et qui porte le même *client handle* que celui de la requête.
Le PDP peut cependant envoyer des messages de décisions qui ne sont pas sollicités par un message de requête comme nous l'avions précisé, pour faire une mise à jour de la configuration du PEP. A l'aide d'un message de décision, le PDP peut aussi obliger un PEP à lui envoyer un message de requête, c'est à dire ouvrir un nouveau contexte, ou encore l'obliger à effacer un contexte existant.
Un message peut contenir plusieurs décisions, c'est à dire qu'il peut contenir un ensemble de règles qui permettent de supprimer ou bien d'ajouter des politiques. Il doit être traité par le PEP comme étant une seule transaction. En d'autres termes il suffit que le PEP ne supporte pas une décision pour qu'il revienne à sa configuration initiale et ne prenne pas en compte cette nouvelle configuration.

```
<Decision Message> : : = <CommonHeader> <Client Handle> <Decision (s)> <Error> [Integrity]
```

– **Report State** : Le PEP accuse réception des messages de décision à l'aide du message Report State, et envoie un rapport au PDP incluant l'action prise. Le PDP est donc toujours au courant de la politique installée sur le PEP. Ce message peut être:
  - **Success** : Le PEP a pu installer toutes les décisions de politiques reçues du PDP dans sa propre PIB.
  - **Failure** : Si une décision a échoué, le PEP ne prend pas compte de cette nouvelle configuration et envoie ce message en indiquant quelle décision a causé l'erreur.

  De même, à n'importe quel moment le PEP peut envoyer au PDP l'état courant de n'importe quel contexte dans le message *Report-State.* On parle dans ce cas d'*Accounting*.

```
<Report-State> : : = <CommonHeader> <Client Handle> <Report-Type>
```

– **Client-Close**: Ce message est utilisé pour fermer la connexion. Il peut être envoyé par le PEP ou le PDP. Il peut être également utilisé en cas de refus d'ouverture de connexion par le PDP comme nous l'avions vu. La raison de refus sera alors mentionnée dans l'objet *Error*.

```
<Client- Close>: : = <CommonHeader> <Error> [<PDPRedirAddr>][Integrity]
```

A.4. COPS-PR

Le modèle de provision permet à un PEP de prendre des décisions politiques localement. Pour ce faire, durant une phase d'initialisation, le PDP pourvoit ses PEPs d'une base de données de décision applicable aux services déployés sur le réseau que l'on appelle la PIB [45] (Policy Information Base) ou base de données des informations de politiques. La PIB peut être décrite de manière abstraite comme étant un arbre d'espaces de noms où les branches représentent la structure des données (sous forme de classes de provisions ou PRCs) et les feuilles étant des instances de ces classes (appelées PRIs).

La demande de provision effectuée par le PEP est véhiculée par un message COPS-PR [44] de type *REQ*, où sont spécifiés l'ensemble des rôles supportés par le PEP. En retour, le PDP émet vers le PEP la base de données de politiques inhérente à la combinaison de rôles reçus.

Comme dans le travail effectué au niveau de cette thèse nous avons implémenté le protocole COPS-PR sur une plateforme active et nous avons ajouté de la sémantique au modèle d'information de la PIB, nous décrivons en détail dans cette section ce type de client COPS.

A.4.1. Objets et messages COS-PR

COPS-PR définit des messages et des objets bien spécifiques. Nous citons particulièrement les messages *Request*, *Decision*, *Report State* et six objets appelés *Complete Provisioning Instance Identifier*, *Prefix PRID*, *Encoded Provisioning Instance Data*, *Global Provisioning Error*, *PRC Class Provisioning Error* et *Error PRID* qui sont eux-mêmes encapsulés dans des objets de type *COPS Decision* ou *COPS Client Specific Information Objects*.

Le format d'un objet COPS-PR est décrit dans la Figure A.7:

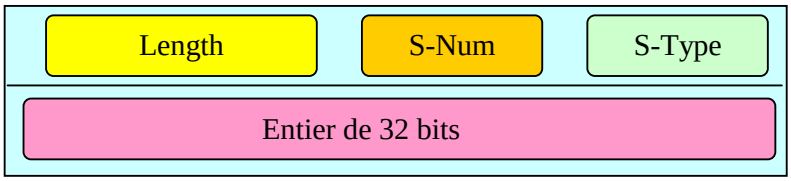

Figure A.7 - Format d'un objet COPS-PR

- **Length** (2 octets): indique la longueur de l'objet en octet.
- **S-Num** (1 octet): identifie l'objet, exemple: PRID, PPRID, …
- **S-Type** (1 octet): décrit le codage spécifique utilisé pour cet objet, exemple: BER [137], XML, …

Les différents types d'objets COPS-PR sont les suivants:

- **Provisioning Instance Identifier**: PRID

  S-Num = 1 (Complete PRID), S-Type = 1 (BER), Length = variable.

  Cet objet est utilisé pour transporter l'identificateur d'une Instance de provision (PRI). Cet identificateur est codé en suivant les mêmes règles de codage utilisées pour le OID (*Object Identifier*) en SNMP. Il est codé spécifiquement en utilisant le mode TLV (Type/Longueur/Valeur).

- **Prefix Provisioning Instance Identifier**: PPRID

  S-Num = 2 (Prefix PRID), S-Type = 1 (BER), Length = variable.

  C'est l'identificateur d'une classe de provision (PRC). On l'utilise dans certaines décisions comme les décisions d'effacement, pour éliminer toutes les PRIs d'une PRC au lieu de les éliminer chacune à part en utilisant son PRID. On les retire donc toutes ensembles en utilisant le PPRID de la PRC à laquelle elles appartiennent.

- **Encoding Provisioning Instance Data**: EPD

  S-Num = 3 (EPD), S-Type = 1 (BER), Length = variable.

  Cet objet est utilisé pour transporter la valeur code d'une PRI, c'est à dire la valeur de chacun de ses attribut. Toutes ces valeurs sont assemblées en commençant par l'attribut ayant le OID le plus petit en finissant par celui ayant le OID le plus grand. Cet ensemble formera le EPD.

- **Global Provisioning Error Object**: GPERR

  S-Num = 4 (GPERR), S-Type = 1 (for BER), Length = 8.

  Cet objet est utilisé pour communiquer des erreurs générales, exemple :

- *availMemLow* : La taille de la mémoire disponible est trop petite.
- *maxMsgSizeExceeded* : Taille maximale du message dépassé.
- *unknownError* : Erreur inconnue
- *unknownCOPSPRObject* : l'objet COPS-PR indiqué n'est pas connu.

- **PRC Class Provisioning Error Object**: CPERR

  S-Num = 5 (CPERR), S-Type = 1 (for BER), Length = 8.

  Il a pour rôle de communiquer des erreurs qui sont en relation avec des PRCs spécifiques. Quelques exemples de ces erreurs:
  - *priSpaceExhausted* : On ne peut plus installer d'instances dans cette classe.
  - *attrValueInvalid* : La valeur de cet attribut n'est pas valide.
  - *attrValueSupLimited* : La valeur spécifiée pour cet attribut est légale mais n'est pas supportée par la machine.
  - *attrMaxLengthExceeded* : La longueur de la valeur de cet attribut dépasse la limite.

- **Error PRID Object** : ErrorPRID

  S-Num = 6 (ErrorPRID), S-Type = 1 (BER), Length = variable.

  Cet objet est utilisé pour transporter le PRID de l'instance qui a causé une erreur.

A.4.2. La PIB dans COPS-PR

La PIB est en fait l'équivalent de la MIB [125] pour la gestion de QoS. C'est un modèle permettant de décrire en ASN.1 les politiques de gestion et leur format d'échange entre le PEP et le PDP.

La PIB (Policy Information Base) est donc une structure arborescente qui permet de stocker et de référencer de manière unique chaque instance de classe de provision (PRI). Une PRI est représentée par une feuille de l'arbre, et chaque noeud par le type OBJECT-IDENTIFIER.

Chaque feuille (PRI) et nœud (PRC) de l'arbre est muni d'un nombre entier qui constitue son identifiant. On peut ainsi référencer de manière unique chaque PRI en concaténant son identifiant à celui de chacun de ses nœuds parents.

La Figure A.8 illustre la représentation logique de cette PIB :

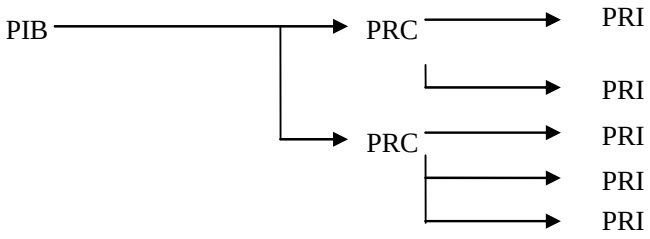

Figure A.8 - Représentation logique de la PIB

On peut imaginer que dans la PIB toute information est représentée sous forme d'une table (PRC) identifiée par un PPRID (*Prefix Provisioning Instance Identifier*) ayant un ensemble d'attributs et dont les entrées (lignes de la table) sont des PRIs (*Provisioning Instances*). Les instances d'un PRC sont identifiées par un PRID (Provisioning Instance Identifier). Les PIBs sont définies dans COPS-PR uniquement à base de structures abstraites. Les détails de chaque PIB (les PRCs et leurs sémantiques) sont spécifiés dans des standards différents (par exemple des Drafts ou des documents propriétaires à certaines entreprises). Plusieurs PIBs ont été définies pour pouvoir couvrir de la manière la plus complète possible la variété des domaines de la gestion des réseaux (Diffserv, Sécurité, Comptabilité…).

La représentation physique de la PIB par une table où les entrées sont les PRCs et chaque PRC contient plusieurs PRIs est illustrée dans la Figure A.9:

PRC 1	Attribut 1	Attribut 2	
PRI 1			
PRI 2			
PRC 2	Attribut 1	Attribut 2	Attribut 3
PRI 1			
PRI 2			
PRI 3			

Figure A.9 - Représentation physique de la PIB

Les deux opérations possibles sur la PIB sont les suivantes :

- Install: Cette opération crée ou met à jour une PRI. Elle a besoin pour cela de deux paramètres : PRID (pour nommer la PRI) et la EPD (Encoding Provisioning Instance Data), permettant de donner les valeurs des attributs.
- Remove: Cette opération permet de supprimer une PRI d'une PRC. Elle nécessite la précision d'un seul paramètre qui est le PRID (pour la suppression d'une PRI) ou bien le PPRID (pour la suppression d'une PRC).

Nous verrons dans le paragraphe suivant la spécification qui permet de définir la PIB dans COPS-PR.

A.4.3. La SPPI (Structure of Policy Provisioning Information)

La définition des PIBs se fait conformément à une spécification bien déterminée appelée SPPI [46] pour *Structure of Policy Provisioning Information*. Celle-ci présente de nombreuses constructions ainsi que leurs sémantiques, elle indique également la manière de réutiliser la définition des données des PIBs en évitant bien entendu la redondance des informations.

Dans le RFC 3159, la SPPI est définie en ASN.1 et basée sur la définition du SMI [138] et de la MIB du protocole SNMP, accompagnée d'une description textuelle de chaque élément. Elle est divisée en six parties :

- La macro MODULE-IDENTITY(1) utilisée pour spécifier la sémantique d'un module de la PIB (élément regroupant toutes les tables d'une même catégorie).
- La macro OBJECT-TYPE(2) qui spécifie la syntaxe et la sémantique des PRCs et de leurs attributs (déclaration d'une table, d'un attribut ou d'une entrée).
- La SPPI utilise les macros OBJECT-GROUP(3) et MODULE COMPLIANCE(4) pour présenter les contraintes d'implémentation qui sont acceptables pour les attributs des PRCs et donc indirectement pour indiquer celles des PRCs elles-mêmes.
- La macro TEXTUAL-CONVENTION(5) permet à la SPPI de définir des nouveaux types de données ayant une syntaxe et une sémantique particulière commune à plusieurs attributs d'une ou plusieurs PRCs (donc à partir d'un type de base prédéfini).
- Enfin, la macro OBJECT-IDENTITY(6) est utilisée dans les modules de la PIB pour fournir des informations sur la déclaration d'un OBJECT-IDENTIFIER.

Comme exemple de macros SPPI, nous donnons ci-dessous la définition de la macro OBJECT-TYPE:

```
OBJECT-TYPE MACRO ::=
TYPE NOTATION ::=
        "SYNTAX" Syntax
                UnitsPart
                "PIB-ACCESS"            Access  -- modified
                PibReferencesPart               -- new
                PibTagPart                      -- new
                "STATUS"                Status
```

```
        "DESCRIPTION"       Text
        ErrorsPart                  -- new
        ReferPart
        IndexPart                   -- modified
        MibIndexPart                -- modified
        UniquePart                  -- new
        DefValPart
    VALUE NOTATION ::=
        value(VALUE ObjectName)
```

A.4.4. Meta Politique du PIB

La Méta-Politique PIB [47] est une tentative de mouvoir une partie de la fonctionnalité et de l'intelligence du PDP vers le PEP en utilisant les procédures COPS-PR standards. L'idée fondamentale est que les équipements modernes de réseau (particulièrement les équipements actifs) ont suffisamment de ressources pour effectuer quelques traitements, et une partie de l'intelligence du PDP dans le modèle de PBN peut être distribuée ainsi entre eux. Ceci est réalisé par des "méta-politiques" qui sont des politiques définies dans une nouvelle PIB appelée PIB des méta-politiques. Les méta-politiques sont envoyées au PEP par le PDP et qui sont appliquées sur l'équipement comme étant des politiques de réseau "traditionnelles". L'application des méta-politiques sur un PEP contrôle le contenu des autres PIBs, ce qui permet d'appliquer les politiques de ces PIBs implémentées sur l'équipement. L'équipement, après réception des méta-politiques, peut surveiller plusieurs événements et prendre des décisions de politiques au nom du PDP d'une façon indépendante et décentralisée, rendant de cette manière le modèle COPS plus robuste, plus fiable et plus tolérant aux erreurs.

La Méta-Politique PIB est définie conformément à la spécification SPPI, elle comporte 14 classes de provision (PRC), groupées dans cinq groupes:

- Le groupe de capacités (Capabilities Group) contient les PRCs qui stockent les capacités et les limitations du PEP (tant que la PIB méta-politique est concernée). Les PRIs de ces classes sont rapportés au PDP quand le PEP se connecte.
- Le groupe de base de Méta-Politique (Base Meta-Policy Group) contient les classes qui définissent la forme des méta-politiques, qui définissent leur priorité en cas de conflits, et qui rapportent leur statut.
- Le groupe de condition (Condition Group) fournit des classes pour définir les conditions des méta-politiques.
- Le groupe d'action (Action Group) inclut les PRCs qui définissent les actions des méta-politiques.
- Le groupe de paramètre (Parameter Group) contient les PRCs qui définissent les paramètres et leurs méthodes d'évaluation.

A.5. LES REGLES DE POLITIQUE

Une politique de gestion est un ensemble de règles qui permet de fixer le comportement des éléments qu'elle administre. Par élément, on entend service, usager ou équipement.

Afin d'illustrer cette définition, considérons l'exemple de réseau local d'une entreprise. Le réseau est relié à l'Internet, comporte un routeur d'accès ainsi que deux sous-réseaux. L'un est utilisé par des employés administratifs, l'autre par des ingénieurs.

D'un point de vue fonctionnel, une gestion par politiques va permettre de répondre au cahier des charges suivant :

– Différenciation du traitement de trafic: on désire munir les usagers de classes de services différentes (au sens bande passante et délai de transit) en fonction de leurs besoins. Pa exemple, les ingénieurs devront bénéficier d'un service Premium (QoS garantie) et les employés administratifs d'une gestion standard de type Best Effort.

– Définition de contraintes horaires: on veut autoriser l'accès des usagers à l'Internet seulement durant les heures de travail, c'est à dire de 8h00 à 16h00.
– Sécurisation du trafic: les ingénieurs ont besoin de communiquer avec ceux d'un site distant. On veut donc assurer la confidentialité des données échangées en créant un canal privé virtuel.
– Définition de comportements critiques: en cas de congestion du routeur d'accès, on veut pouvoir reclasser le trafic généré par les deux sous-réseaux de la manière suivante: Le trafic ingénieur initialement muni d'une classe de service Gold sera traité d'une manière standard (Best Effort); le trafic généré par les employés administratifs sera supprimé.

La mise en oeuvre des besoins cités ci-dessus va s'effectuer sous la forme de règles qui vont régir le comportement, les droits et les interdictions des services, équipements et usagers inscrits dans le cadre de la gestion du réseau.

Une règle de politique est de la forme « **Si** *condition* **alors** *action* ». Dans ce cas la *condition* concerne l'utilisateur, le groupe, la date, le type d'application ou encore l'adresse réseau. L'*action* est justement le comportement à avoir et elle est caractérisée par les politiques à suivre.

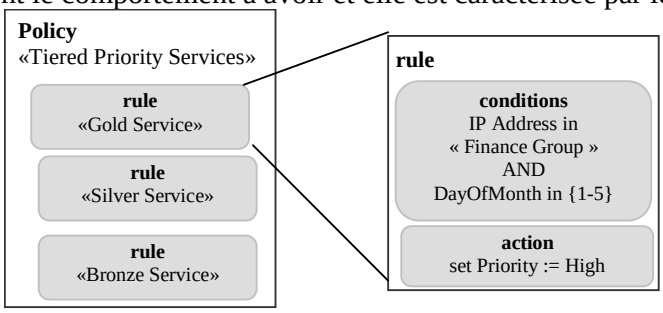

Figure A.10 - Forme d'une règle de politique

Nous donnons les exemples explicatifs suivants :
Si l'application est de type téléphonique **Alors** mettre les paquets en priorité *Premium*.
Si groupe=élèves **ET** heure=8 :00 – 18 :00 **Alors** bande passante Internet =2 Mbps.

Afin d'ordonner les différents types de politiques, l'IETF propose la classification fonctionnelle de politiques suivante:
– Politiques de motivation: concernent le fait et la manière d'atteindre un objectif politique. Les politiques de configuration et d'utilisation sont des types de politiques de motivation. Une politique de configuration définit la configuration par défaut d'un élément, alors que la politique d'utilisation définit la configuration d'un élément basée sur des données d'utilisation.
– Politiques d'installation: définissent ce qui peut être installé ou non sur un équipement.
– Politiques d'erreurs et évènementielles: définissent les actions à effectuer lorsqu'un élément du réseau disfonctionne.
– Politiques de sécurité: concernent les mécanismes à appliquer aux clients, concernant l'autorisation, l'authentification, et la facturation de l'accès aux ressources.
– Politiques de service: caractérisent le réseau et les services disponibles.

Une politique peut se définir à différents niveaux. Le niveau le plus haut correspond à celui de l'utilisateur. La détermination d'une politique dans ce cas se fait par une négociation entre l'utilisateur et l'opérateur et c'est pour cette négociation qu'on préconise l'utilisation de règles prédéfinies par cet opérateur. En effet, dans ce cas l'administrateur décrit à ce niveau les règles qui seront traduites en commande de niveau plus bas permettant l'installation des politiques spécifiées. L'utilisateur ne peut que choisir parmi ces règles la politique qu'il souhaite appliquer. Le niveau est donc ici *business*. Il faut néanmoins pouvoir traduire dans un langage niveau réseau cette politique qui détermine le protocole réseau de gestion de QoS par exemple et son paramétrage. Cette traduction est la tâche des langages de spécification des politiques. C'est ensuite qu'on passe à la traduction du niveau réseau au niveau le plus bas correspondant à la programmation des noeuds du réseau (configuration du nœud).

Ces différents niveaux de langage (niveau business, niveau réseau, niveau configuration) sont pris en charge par le groupe de travail *Policy* de l'IETF. Le modèle retenu provient d'un autre groupe de travail appelé DMTF [139] et ce modèle est appelé CIM [75] (Common Information Model). Ce modèle est le standard actuel sur lequel se base l'architecture PBN. Les extensions sont aujourd'hui développées conjointement par ces deux groupes de travail. L'objectif du travail des modèles d'information liés aux différents niveaux de langages est d'obtenir un modèle général qui puisse se décliner en modèles d'information par domaine, ainsi qu'une représentation indépendante des équipements et des implémentations. Les différents niveaux sont illustrés dans la Figure A.11:

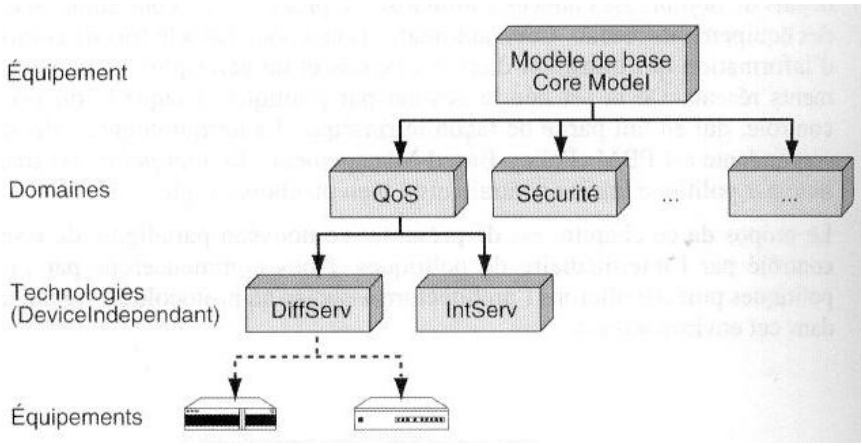

Figure A.11 - Différents niveaux de langages pour la gestion par politique

Si on considère la QoS par exemple, la structure des modèles successifs permet de passer de la définition générale d'une politique de qualité de service à la configuration d'un routeur.

La base de données des politiques doit être indépendante des constructeurs et des équipements afin d'être appliquée de manière homogène et cohérente dans plusieurs domaines de gestion.

L'administrateur doit également pouvoir saisir des règles de haut niveau grâce à une interface conviviale. Ces règles seront alors traduites automatiquement par le PDP et/ou le PEP en règles de bas niveau propres à un constructeur et/ou à un équipement, avant d'être exécutées. Cela permet à l'administrateur de se concentrer uniquement sur la logique des politiques.

Bien sûr, il n'est pas possible de spécifier ces règles avec le langage humain trop ambigu. En fait, il existe des langages spécialisés: Les langages de spécification.

Un langage de spécification est un modèle de description de politiques indépendant du domaine d'application permettant d'atteindre cet objectif.

Les langages de spécification les plus utilisés de nos jours sont Ponder [79], PCIM [129] (même si il est plus un modèle qu'un langage) et PFDL [130]. Notons que ces langages sont tous basés sur le modèle en niveaux de PCIM.

Les langages PFDL et PCIM sont basés sur le même modèle de classes. On trouve des classes structurelles qui forment les éléments de base des politiques et des classes d'associations qui déterminent les relations entre les éléments.

Un exemple de définition d'une règle en PFDL est le suivant:

```
IF   User in {"peter", "frank", "mary"} AND
     Time in Mon - Fri AND
     Time in 09:00 - 17:00 AND
     Http_Url_Path ends ".ra"
THEN
     Deny
```

Annexe B

WEB SEMANTIQUE

B.1. LE WEB COMME OUTIL DE PARTAGE D'INFORMATIONS

A son origine, le World Wide Web [144] a été conçu par Tim-Berners Lee comme un ensemble de pages HTML [134] (Hypertext Markup Language) distribuées et reliées par des liens hypertextes. La simplicité de ce modèle, l'implantation mondiale de l'Internet, la distribution massive de navigateurs HTML, la création du consortium W3C (World Wide Web Consortium) [145] et d'autres raisons économiques et techniques, ont eu ensuite comme résultat une croissance extraordinaire d'informations et d'applications accessibles à travers le Web. En même temps, le Web a été aussi la victime de son succès qui a montré les limites de l'approche hypertexte pour le partage d'informations à grande échelle et le besoin de modèles et outils nouveaux. Les initiatives du W3C autour du langage XML [23] et du Web sémantique [140] répondent à ce besoin par des modèles et langages standardisés pour la représentation et la gestion de données et de connaissances sur le Web, tout en respectant les caractéristiques fondamentales du Web d'origine qui sont l'autonomie des serveurs, la facilité d'accès, etc...

La section suivante détaille l'évolution des langages Web et l'avènement du langage d'ontologie OWL [74] que nous avons choisi d'utiliser dans le cadre de cette thèse comme langage de représentation de la PIB. Ceci pour la gestion des réseaux à base des politiques lors de l'implémentation du protocole COPS-PR. Enfin, nous traitons dans la dernière section, le langage SWRL qui sera utilisé pour la représentation des règles de politiques.

B.2. FORMALISMES ET EVOLUTIONS DES LANGAGES WEB

Depuis sa naissance, le Web subit une série d'évolution. Initialement utilisé pour présenter les données, le Web traditionnel insistait sur la forme plutôt que sur le contenu. L'information était alors seulement compréhensible par l'être humain et donc toute recherche sur l'information était inefficace. De là vient la nécessité d'améliorer le Web dans le sens où les informations stockées ne seront désormais pas uniquement compréhensibles par les humains mais également par les logiciels, ce qui améliore de loin la recherche.

Avant d'expliquer l'évolution du web en passant par les différentes formes qu'il a pris à travers les années, commençons par récapituler dans le tableau suivant la différence entre le web traditionnel et le web sémantique.

Critère	**Web traditionnel**	**Web sémantique**
Intitulé	web informel	web formalisé
	web quantitatif	web qualitatif
	web opaque	web transparent
	web anarchique	web informatique
	web brouillon	web indexé
	web unidimensionnel	web tridimensionnel

Contenus	fait de mots	fait de concepts
	compréhensible par les humains	compréhensible par les humains et les logiciels
	contenus amorphes	contenus structurés
	accumulation	intégration
	contenu quelconque	contenu de valeur

B.2.1. Le langage HTML

Le langage Web de base est bien entendu HTML [134] utilisé pour la création de pages Web à l'aide de balises prédéfinies permettant de créer la structure et la mise en forme de ces pages. Les balises contrôlent la forme du document mais ne permettent pas la gestion de son contenu.

L'exemple suivant montre le document HTML van-gogh.html contenant des informations sur deux peintures du peintre Van Gogh :

```
<html>
 <head>
    <Title>Peintre</Title>
    <H1>Nom: Van Gogh</H1>
    <H1>Liste des peintures:</H1>
</head>
<body>
  <table width="97%">
      <th width="20%">Titre</th>
      <th width="20%">Date</th>
      <th width="60%">Description</th>
     <tr>
      <td width="20%">Portrait du Docteur Gachet</td>
      <td width="20%">1890</td>
      <td width="60%">
        La pose choisie, avec la tête s'appuyant sur la main, est traditionnellement celle de la mélancolie,
        et l'expression de Gachet, la tristesse qui se lit dans ses yeux,
        l'impression d'affaissement du corps, tout concourt à créer une impression de malaise.
      </td>
     </tr>
     <tr>
      <td width="20%">La Chaise et la Pipe</td>
      <td width="20%">1888</td>
      <td width="60%">
        En posant sur la paille du siège sa pipe et son tabac, Van Gogh passe d'une chaise à sa chaise.
        C'est un substitut d'autoportrait.
      </td>
     </tr>
  </table>
</body>
</html>
```

B.2.2. Le langage XML

Un problème important rencontré quand on veut utiliser dans un programme informatique des données publiées sous forme d'un ensemble de pages HTML, est la difficulté d'extraire des informations précises d'une page HTML. Ceci est lié au fait que HTML est conçu pour l'affichage de l'information à l'écran et ne permet pas de préserver la structure des données d'origine. Par exemple, après avoir traduit une table relationnelle contenant des noms de peintres et leurs peintures en tableau HTML, l'extraction de toutes les peintures (par exemple par une requête) devient compliquée et doit prendre en compte des choix de présentation de l'information

dans la page. Afin de résoudre ce problème, XML [23], considéré comme le standard de transfert de données sur le Web, permet à l'utilisateur de créer ses propres balises. Il structure les données et facilite les recherches. Seulement, les balises créées sont uniquement compréhensibles par l'homme et non pas par la machine qui ne peut donc pas détecter les erreurs sémantiques. Une erreur peut être la création de balises de même nom à connotations différentes posant problème aux applications qui utilisent les deux définitions. Des XML Namespaces ont quand même été créés pour attribuer à chaque élément un identificateur unique. Ce principe est illustré par l'exemple suivant qui montre le document XML van-gogh.xml contenant des informations sur deux peintures du peintre Van Gogh :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Peintre>
    <Nom>Van Gogh</Nom>
    <Peintures>
        <Peinture titre= "Portrait du Docteur Gachet" année="1890">
            La pose choisie, avec la tête s'appuyant sur la main,
            est traditionnellement celle de la mélancolie, et l'expression de Gachet,
            la tristesse qui se lit dans ses yeux, l'impression d'affaissement du corps,
            tout concourt à créer une impression de malaise.
        </Peinture>
        <Peinture titre="La Chaise et la Pipe" année="1888">
            En posant sur la paille du siège sa pipe et son tabac,
            Van Gogh passe d'une chaise à  sa chaise. C'est un substitut d'autoportrait.
        </Peinture>
    </Peintures>
</Peintre>
```

B.2.3. Le Web sémantique : Partage des connaissances sur le Web

Grâce à sa faculté de représenter des documents et des données très diverses d'une manière uniforme et « sérialisable » (sous forme d'une séquence de caractères), XML est devenu le modèle de données standard pour la publication et l'échange d'informations sur le Web. Néanmoins XML ne répond que partiellement aux problèmes rencontrés quand on veut partager des informations sur le Web. Cette limite n'est pas d'ordre structurelle ou syntaxique (il est possible de représenter pratiquement toute information structurée en XML), mais concerne la description du sens de l'information représentée. Par exemple pour interroger le document van-gogh.xml précédent, on suppose que l'utilisateur connaît la signification des balises <Peintre>, <Peinture> etc. mais également le sens de la relation père/fils entre les différents nœuds de l'arbre DOM [147] (e.g. la relation de descendance entre un élément A de type Peintre et un élément B de type Peinture est interprétée comme « la peinture B a été créée par le peintre A »).

L'absence d'une description explicite et partagée de cette connaissance est une limitation pour l'exploitation automatique du Web comme source d'informations. Par exemple, un utilisateur qui cherche des adresses de peintres industriels sera étonné de retrouver Vincent Van Gogh parmi les personnes qui peuvent repeindre sa maison. A l'inverse, un deuxième utilisateur qui cherche des noms d'artistes en utilisant la balise <Artiste> sera déçu de ne pas trouver Van Gogh. Ce type d'erreurs sémantiques peut être résolu grâce à l'utilisation d'ontologies pour la description sémantique d'une ressource Web. Par exemple, afin de donner un sens plus formel aux balises utilisées dans le document précédent, il serait possible de définir ou d'utiliser une ontologie existante, qui représente formellement et explicitement des concepts (artiste, peintre, …) et de relations entre ces concepts (un peintre est un artiste, …) concernant le domaine de l'art.

Le Web sémantique est donc une extension du Web traditionnel, dans lequel les informations sont décrites d'une manière précise et définitive pour permettre une meilleure coopération entre les utilisateurs et leurs machines. Comme son nom l'indique, le Web sémantique permet de donner plus de sémantique ou de sens par rapport aux langages de programmation Web habituels.

Le Web sémantique a deux principaux objectifs :
- Arriver à un Web plus « intelligent » où les informations ne sont pas uniquement stockées dans le serveur Web mais sont aussi compréhensibles et utilisables pour un certain traitement par ce dernier.
- Pouvoir transformer la multitude de pages Web en un important index hiérarchisé facilitant ainsi l'accès aux ressources.

Ceci permet aux outils utilisés par les usagers de dépasser le stade de simple affichage des données, d'automatiser les requêtes, d'intégrer et de réutiliser les données dans plusieurs applications différentes.

Nous donnons comme exemple les moteurs de recherche du monde Internet actuel qui ne permettent de donner des réponses qu'à deux grandes catégories de questions de type « Quelles sont les pages contenant le terme X ?» ou encore « Quelles sont les pages les plus populaires au sujet de Y ?». Le Web sémantique permet une importante amélioration dans ce domaine dans le sens où des questions plus spécifiques pourront être posées et cela sur la base de mots-clé. Un exemple serait de pouvoir poser la question « Quel est l'hôpital le plus proche ayant une machine IRM ? ».

En résumé, le Web sémantique permet donc d'identifier les ressources de manière unique avec un sens bien spécifique pour chacune de ces ressources organisées autour d'une ontologie [81].

B.2.4. Les Ontologies

Le terme Ontologie [81] est un terme philosophique signifiant l'explication systématique de l'existence. Dans le domaine de l'intelligence artificielle, une ontologie est décrite comme une définition des termes et relations de base du vocabulaire d'un domaine, ainsi que des règles qui indiquent comment combiner ces termes et relations pour pouvoir étendre ce vocabulaire. En effet, une ontologie définit les termes utilisés pour décrire et représenter un champ d'expertise. Elle associe les concepts de base d'un certain domaine et les relations de ces concepts de manière compréhensible par les logiciels. Elle encode la connaissance d'un domaine particulier ainsi que les connaissances qui couvrent d'autres domaines, permettant ainsi la réutilisation de ces connaissances.

Une ontologie est caractérisée principalement par [80] :
- Le fait qu'elle soit explicite puisqu'elle définit les concepts, les propriétés, les relations et les fonctions qui la forment.
- Le fait qu'elle soit interprétée par un logiciel puisqu'elle est formelle.
- C'est une conceptualisation, vu que le modèle préconisé est un modèle abstrait et simplifié des ressources qu'elle représente.
- Enfin, elle est partagée puisqu'elle est créée d'un commun accord accepté par un groupe d'experts.

Il existe deux groupes d'ontologies: les ontologies légères (Lightweight) qui concernent la modélisation d'information d'un certain domaine de manière simple sans axiomes ou restrictions et les ontologies lourdes (Heavyweight) qui permettent la représentation de liens sémantiques plus spécifiques.

Nous voyons un intérêt particulier dans l'utilisation d'un langage de description d'ontologies standard pour créer celles-ci afin qu'elles soient utilisables par différentes applications. Elles peuvent fournir les interfaces idéales entre les informations et les diverses applications qui souhaitent les traiter.

Le Web sémantique mêlé à l'utilisation des ontologies permettent ensemble :
- d'améliorer les recherches puisque celles-ci tiennent compte des liens unissant les éléments d'informations et les contraintes définies sur ces éléments,

- d'automatiser les tâches étant donné que chaque élément des documents est bien représenté,
- d'assurer l'interopérabilité entre les différents systèmes car la modélisation des informations est indépendante de l'application dans laquelle elles se trouvent.

B.2.5. RDF

Le besoin d'un modèle conceptuel devient nécessaire pour la description de chaque ressource. D'où l'introduction de RDF [70] pour *Resource Description Framework* qui est un modèle conceptuel, abstrait et formel, fondé sur un modèle de graphe de ressources, permettant de décrire les éléments simplement et sans ambiguïté selon un mécanisme basé sur des déclarations RDF. Une déclaration RDF est une phrase composée d'un triplet <Sujet, Prédicat, Objet> qui peut être traitée par la machine pour permettre à celle-ci de le faire tout en comprenant la signification de ce triplet. Chaque ressource, et plus précisément, chaque sujet du triplet est identifié par un URI [141] (*Uniform Resource identifier*). Cette identification se fait de manière unique à l'aide d'un nom sans avoir à localiser la ressource (un bon exemple d'URI est l'URL). Le prédicat exprime la propriété, comme par exemple « est créateur » dans le triplet « Monsieur X - est créateur – du Web Sémantique » où Monsieur X est le sujet et enfin « Web Sémantique » est l'objet.

Par exemple, le fait que le document van-gogh.xml « parle de » peintures à l'huile sur toile de la période néo-impressionniste peut être représenté par le graphe suivant fondé sur une ontologie provenant du domaine de l'art :

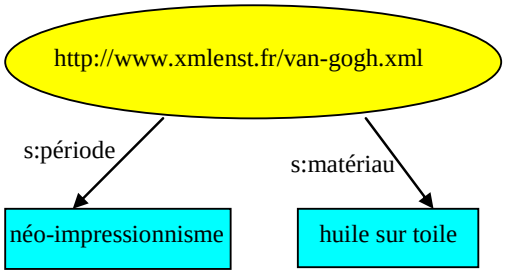

Figure B.1 - Exemple d'un graphe de ressource RDF

La recommandation RDF propose l'utilisation de XML pour la représentation des graphes de ressources. Voici une représentation XML du graphe de la Figure B.1 :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:s="http://schemas.xmlenst.fr/peintres_rdf/">
<rdf:Description about="http://www.xmlenst.fr/van-gogh.xml">
        <s:période>néo-impressionnisme</s:période>
        <s:matériau>huile sur toile</s:matériau>
</rdf:Description>
</rdf:RDF>
```

Pour utiliser les termes d'une ontologie, il faut pouvoir indiquer les différents vocabulaires utilisés. Ainsi, au début de chaque ontologie, on intègre un ensemble de déclarations d'espaces de noms dans une balise *<rdf:RDF>*. Ces déclarations permettent l'interprétation sans ambiguïté des URIs et la lecture facile du reste de l'ontologie. On y trouve l'identification de l'espace de noms implicite de l'ontologie, l'identification de l'espace de noms relative aux préfixes utilisés. Enfin, on y trouve les différentes descriptions RDF ou, comme on le verra dans les sections suivantes, des éléments RDFS ou OWL.

Ainsi, RDF permet de définir aisément une ontologie, son inconvénient est qu'il ne supporte pas la vérification de la cohérence des données (vérification que le champ « date de naissance » est vraiment une date par exemple).

B.2.6. RDFS

Une évolution de RDF est introduite dans RDFS (pour RDF Schema) [71]. Ce langage est simple et permet l'implémentation du modèle RDF pour la définition des ontologies avec une approche cette fois-ci orientée objet. Les trois notions principales permettant cela sont: la ressource (*rdfs:Resource*), la classe (*rdfs:Class*) et la propriété. On a avec RDFS l'avantage de pouvoir créer une hiérarchie de classes et de propriétés, comme dans les langages orientés objet, grâce aux notions de *subClassOf* et *subPropertyOf*. On peut bien entendu instancier une classe à l'aide de *rdf:type*. L'autre avantage vient du fait que RDFS restreint le domaine d'application des propriétés (avec un domaine qui précise la classe du sujet du triplet utilisant la propriété et un intervalle précisant la classe de l'objet du triplet utilisant la propriété).

La représentation RDFS de l'exemple de la section B.2.5 est la suivante :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <rdfs:Class rdf:ID="Objet_Iconographique"/>
    <rdfs:Class rdf:ID="Peinture">
        <rdfs:subclassOf rdf:resource="#Objet_Iconographique"/>
    </rdfs:Class>
    <rdfs:Class rdf:ID="Période"/>
    <rdf:Property rdf:ID="date">
        <rdfs:domain rdf:resource="#Objet_Iconographique"/>
        <rdfs:range rdf:resource="Période"/>
    </rdf:Property>
    <rdf:Property rdf:ID="matériau">
        <rdfs:domain rdf:resource="Peinture"/>
        <rdfs:range rdf:resource="rdfs:Literal"/>
    </rdf:Property>
    <Peinture about="van-gogh.xml">
        <date>
            <Période id="#néo-impressionnisme"/>
        </date>
        <matériau>huile sur toile</matériau>
    </Peinture>
</rdf:RDF>
```

Dans ce document, l'ontologie est composée de trois concepts ou classes (Literal, Peinture, Période) et de deux propriétés (matériau et date). Peinture est une description RDF, elle est appelée la ressource sujet des deux propriétés qui ont comme ressources objets, respectivement, Matériau et Période.

L'intérêt de définir un schéma RDFS n'est pas seulement de pouvoir contrôler la terminologie et la structure des descriptions RDF, mais également d'introduire la possibilité de raisonner sur les liens *isA* (est-un) qui existent entre les concepts et les propriétés. Ceci est surtout utile quand on veut interroger des métadonnées pour découvrir des ressources. Voici un exemple de requête RQL [146], qui cherche toutes les ressources sur des objets iconographiques de la période néo-impressionniste :

```
select X  from Objet_Iconographique{X}.Période.{Y}  where Y = "#néo-impressionnisme"
```

Cette requête prend en compte le fait que toutes les instances de la classe Peinture sont également des instances de la classe *Objet_Iconographique* (à travers la propriété rdfs:subclassOf) et renvoie la ressource *van-gogh.xml*.

Un inconvénient majeur est cependant à noter: il est difficile de définir dynamiquement une nouvelle classe à partir d'une classe donnée avec des restrictions supplémentaires (Exemple d'une telle classe: Un enfant est une personne dont l'age est inférieur à 18 ans).

B.2.7. OWL (Ontology Web Langage)

Les langages RDF et RDFS vus précédemment pour le développement d'outils et d'ontologies ne sont pas compatibles avec le Web actuel et de manière plus flagrante avec le Web sémantique. D'où la création d'un langage de définition d'ontologies orientées Web, appelé OWL [74], qui est une extension de RDFS.

OWL est un vocabulaire XML basé sur RDF et permettant de définir des ontologies Web structurées. L'existence d'ontologies produites à partir du même standard autorise une intégration plus riche et garantit l'interopérabilité des données malgré les limites applicatives. OWL facilite l'interprétation par la machine des informations Web et de ce fait est utilisé dans les applications destinées à traiter des informations et non pas à les présenter uniquement à l'utilisateur final.

OWL étend RDFS en utilisant les notions qui y sont définies (ressource, classe, relation, type, subClassOf, domaine, intervalle…). Il permet en plus la création de relations complexes entre différentes classes ainsi que la définition de contraintes sur les classes et les propriétés, plus précises que dans RDFS (*inverseOf*, *transitiveProperty*, *cardinality*, *maxCardinality*, *equivalentClass*, *equivalentProperty*, *SameAs*, *DifferentFrom*, *AllDifferent* …).

Les principaux éléments de base d'OWL sont :

- Les classes qui sont toutes implicitement des sous-classes de la classe *owl:thing.*
- Les individus qui sont des instances d'une classe (créés par *rdf:type*).
- Les propriétés qui permettent la description des membres d'une classe.

Les propriétés sont de type *ObjectProperty*, qui prennent leurs valeurs d'une classe de OWL, ou *DataTypeProperty* qui correspond à un type de donnée bien défini (*integer*, *float* …). Comme pour les classes, on peut créer une hiérarchie de propriétés à l'aide de *subPropertyOf*).

L'exemple suivant spécifie que les instances du concept Peinture ne peuvent pas être des instances du concept Sculpture (Peinture est une sous-classe du complément de la classe Sculpture) :

```
<owl:Class rdf:about="Peinture">
   <rdfs:comment>une peinture n'est pas une sculpture</rdfs:comment>
   <rdfs:subClassOf>
      <owl:Class>
         <owl:complementOf rdf:resource="#Sculpture"/>
      </owl:Class>
   </rdfs:subClassOf>
</owl:Class>
```

Dans OWL, il est également possible de décrire un concept à travers ses propriétés. Ainsi, l'expression suivante décrit les instances de la classe Peintre comme des personnes qui ont créé au moins une peinture :

```
<owl:Class rdf:ID="Peintre">
   <rdfs:subClassOf rdfs:resource="#Personne">
      <owl:Restriction>
         <owl:onProperty rdf:resource="#a_créé"/>
         <owl:hasClass rdf:resource="#Peinture"/>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

B.2.8. Editeurs d'ontologie

Différents éditeurs existent pour définir des ontologies. Nous avons fait le choix de Protégé [142] qui semble être le plus intéressant puisqu'il est simple (il fournit un didacticiel détaillé),

présente une interface graphique conviviale, et permet de tester et valider une ontologie. C'est de plus un logiciel libre (open-source) et extensible par des plugins.

La Figure B.2 montre l'interface graphique de *protégé*.

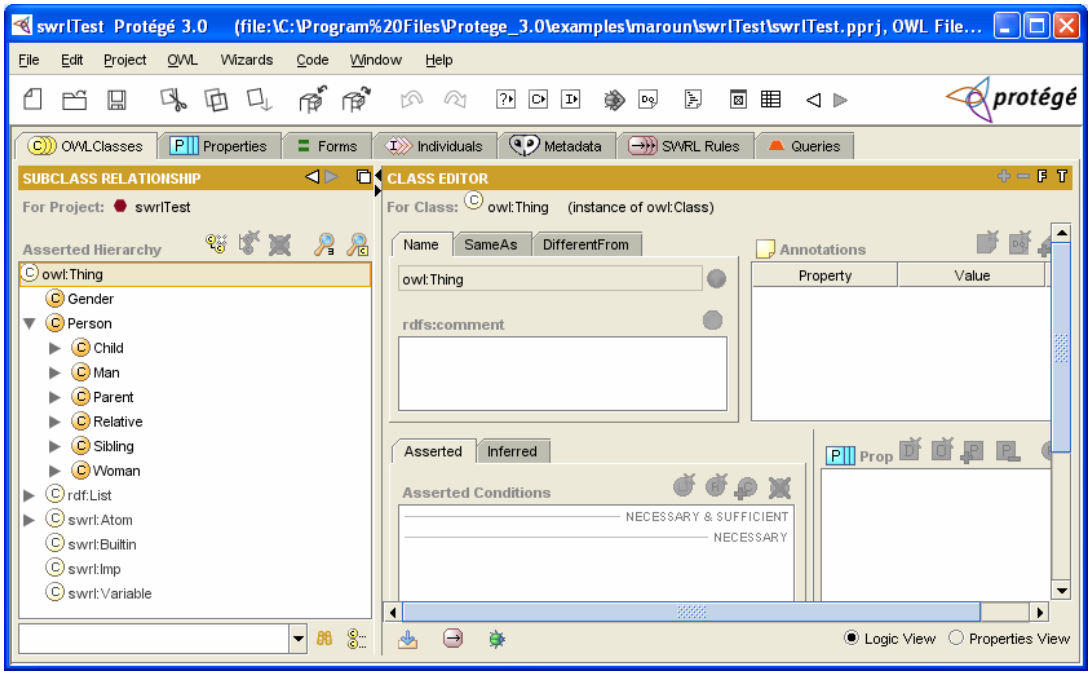

Figure B.2 - Interface graphique de l'éditeur d'ontologies "protégé"

B.2.9. Conclusion

Les paragraphes précédents ont montré l'utilité de OWL et de RDF qui constituent les piliers du Web sémantique en se basant principalement sur les concepts suivants :

- Un schéma d'identification global avec les URIs.
- Une syntaxe standard pour décrire une ressource avec RDF.
- Une manière de décrire les propriétés de cette ressource avec RDFS.
- Une manière de décrire les relations entre les ressources avec OWL particulièrement.

B.3. SWRL (SEMANTIC WEB RULE LANGUAGE)

OWL permet de spécifier des propriétés algébriques pour les relations. Mais il lui manque surtout des possibilités pour encoder des connaissances plus générales, relatives en particulier à la composition des relations. Ainsi, SWRL [143] (Semantic Web Rule Language ou langage de règles du Web sémantique) constitue un pas vers le rapprochement entre les langages OWL et les règles logiques. SWRL est basé sur une combinaison du langage ontologique Web OWL, avec les sous langages Datalog RuleML unaire/binaire du langage Rule Markup Language [169]. SWRL intègre une syntaxe abstraite de haut niveau pour les règles de Horn [170] dans le langage OWL.

SWRL [143] préserve la sémantique intéressante de OWL. La syntaxe SWRL définit une règle qui est une relation d'implication entre un antécédent (body) et un conséquent (head). Si les conditions spécifiées dans l'antécédent sont vérifiées alors les conditions spécifiées dans le conséquent le sont aussi (antecedent ⇒ consequent).

L'antécédent et le conséquent d'une règle sont formés d'aucun ou d'un ensemble d'atomes. Les atomes peuvent avoir les formes C(x), P(x,y), sameAs(x,y), DifferentFrom(x,y), où C est une description OWL, P est une propriété OWL, et x et y sont des variables, des individus OWL ou encore des valeurs de données (datavalues) de OWL.

Les atomes multiples dans un antécédent sont traités de manière conjointe et un antécédent vide est considéré comme étant vérifié (satisfait par n'importe quelle interprétation), d'où la né-

cessité pour un conséquent d'être satisfait par n'importe quelle interprétation. Les atomes multiples dans un conséquent sont eux traités de manière séparée, c'est à dire qu'il faut que tous les atomes soient satisfaits pour que l'antécédent soit satisfait. Un conséquent vide est traité comme étant non vérifié (il n'est pas satisfait quelque soit l'interprétation de l'antécédent).

Un exemple peut être donné en appliquant cette syntaxe pour une règle combinant les propriétés de parent et frère qui implique la propriété d'oncle, cette règle pouvant être écrite de la manière suivante :

parent(?a, ?b) ^ frere(?b, ?c) → oncle(?a,?c)

Ainsi si John a Mary comme parent et Mary a Bill comme frère, alors cette règle implique forcément que Bill est l'oncle de John.

Voici cet exemple représenté en SWRL :

```
<swrl:Imp rdf:ID="Def-hasUncle">
  <swrl:head>
   <swrl:AtomList>
    <rdf:first>
     <swrl:IndividualPropertyAtom>
      <swrl:argument1 rdf:resource="#x"/>
      <swrl:argument2 rdf:resource="#z"/>
      <swrl:propertyPredicate rdf:resource="#hasUncle"/>
     </swrl:IndividualPropertyAtom>
    </rdf:first>
    <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
   </swrl:AtomList>
  </swrl:head>
  <swrl:body>
   <swrl:AtomList>
    <rdf:first>
     <swrl:IndividualPropertyAtom>
      <swrl:argument2 rdf:resource="#y"/>
      <swrl:propertyPredicate rdf:resource="#hasParent"/>
      <swrl:argument1 rdf:resource="#x"/>
     </swrl:IndividualPropertyAtom>
    </rdf:first>
    <rdf:rest>
     <swrl:AtomList>
      <rdf:first>
       <swrl:IndividualPropertyAtom>
        <swrl:propertyPredicate rdf:resource="#hasBrother"/>
        <swrl:argument1 rdf:resource="#y"/>
        <swrl:argument2 rdf:resource="#z"/>
       </swrl:IndividualPropertyAtom>
      </rdf:first>
      <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
     </swrl:AtomList>
    </rdf:rest>
   </swrl:AtomList>
  </swrl:body>
 </swrl:Imp>
```

Annexe C

TECHNOLOGIE .NET

C.1. INTRODUCTION

Afin de comprendre l'implémentation de ASWA sur une plateforme Windows on va décrire les éléments de base qui sont utilisés dans cette implémentation. Il s'agit là de quatre composants principaux:

- plateforme .Net
- ASP.NET
- Les Services Web et les messages SOAP
- Microsoft WSE (Web Service Enhancements).

C.2. LA PLATEFORME .NET

La plateforme .Net [110] est une infrastructure qui permet de créer des applications fortement distribuées sur l'Internet pour différents environnements et périphériques ainsi que des applications Windows standards, des composants serveurs, des applications basées sur tout type d'équipement.

La plateforme .Net a réussi un des plus grands défis de l'industrie des logiciels: L'échange de données entre des applications écrites en différents langages de programmation. En effet, il permet l'échange de données entre plusieurs applications en utilisant les services Web. Il fournit une infrastructure distribuée qui permet aux applications tournant en deux processus différents, sur une ou plusieurs machines d'échanger des données en utilisant le protocole HTTP (Hypertext Transfer Protocol), XML et SOAP (Simple Object Access Protocol).

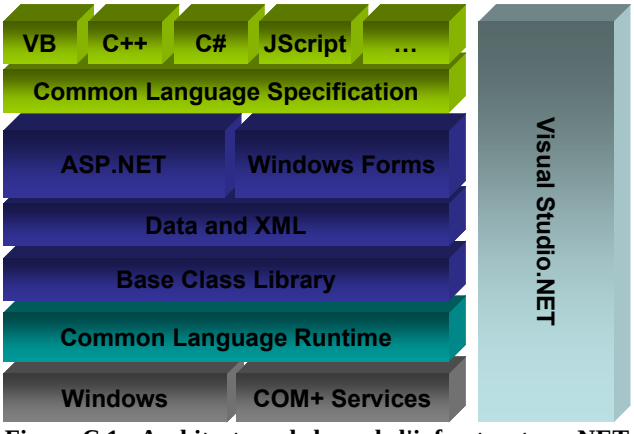

Figure C.1 - Architecture de base de l'infrastructure .NET

La plateforme .NET (Figure C.1) est constituée principalement de deux composants: la librairie de classes et le CLR (Common Language Runtime).

La plateforme .Net est un environnement riche en classes. La librairie de classes de la

plateforme .NET fournit les types communs à tous les langages du .NET. Les Programmeurs peuvent utiliser ces types pour différentes catégories d'applications comme les applications console, les applications Windows, les formulaires Web, et les services Web XML.

Le CLR (Common Language Runtime) est le moteur d'exécution des applications de l'infrastructure .NET. Il fournit des services tels que le chargement et l'exécution du code en renforçant la sécurité et la sûreté des types, l'isolation de la mémoire d'application, la gestion de la mémoire, le traitement des exceptions, l'accès aux méta-données et la conversion du langage intermédiaire (MicroSoft Intermediate Language ou MSIL) en code de la plateforme native.

C.3. ASP.NET

C.3.1. LE SERVEUR WEB IIS

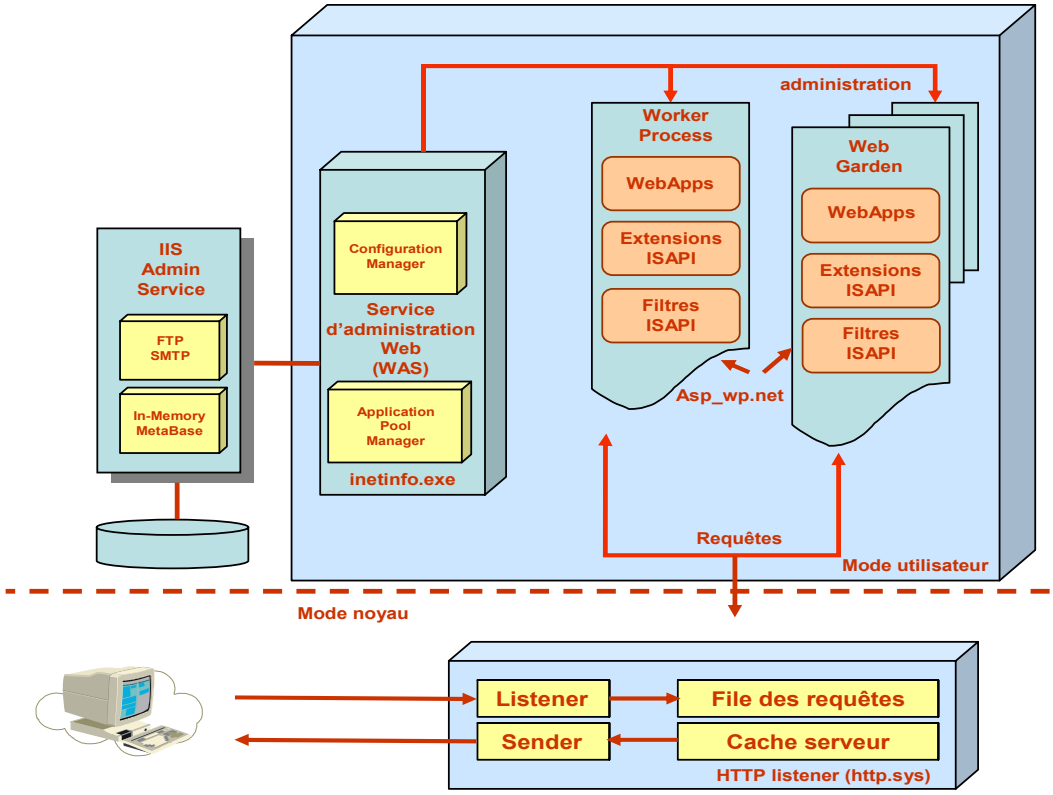

Figure C.2 - Architecture du serveur Web IIS

IIS [115] est un service Windows spécialisé dans le traitement des requêtes HTTP. Lorsqu'une requête est interceptée, IIS examine l'extension du fichier de requête à la recherche d'une valeur connue. Certains fichiers peuvent être traités nativement, html, image, d'autres le sont par une application spécialisée appelée extension ISAPI (Internet Service Application Programming Interface). C'est par exemple le cas des fichiers aspx. Considérons par exemple l'URL suivante:

http://www.amethyste.com/login.aspx

L'URL se termine par l'extension .aspx. Cette extension est caractéristique des applications ASP.NET. IIS communique à .NET cette URL via l'extension ISAPI non managée aspnet_isapi.dll et le filtre aspnet_filter.dll. L'extension agit comme routeur d'URL tandis que le filtre prend en charge les sessions d'ASP.NET.

C.3.2. ASP.NET

ASP.NET [114] fournit un modèle de développement Web unifié qui offre les services nécessaires pour générer des applications Web d'entreprise. Il fournit un modèle de programmation et une infrastructure qui permettent de créer une catégorie d'applications intégrées à la

plateforme .NET, ASP.NET permet de tirer parti des fonctionnalités du Common Language Runtime, telles que la sécurité de type, l'héritage, l'interopérabilité entre les langages et la gestion des versions.

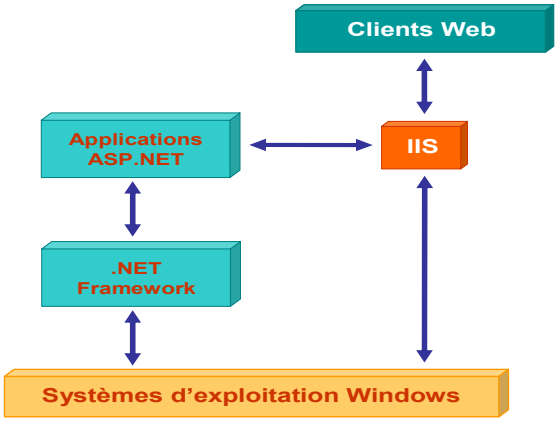

Figure C.3 - Architecture de ASP.NET

Les pages ASP.NET sont compilées. Cette opération est transparente, elle est réalisée lors du chargement de la page par le moteur ASP.NET. La compilation est effectuée à la volée. Mais auparavant, la page est d'abord analysée et transformée en classe. La nouvelle classe générée est compilée et utilisée par le moteur ASP.NET.

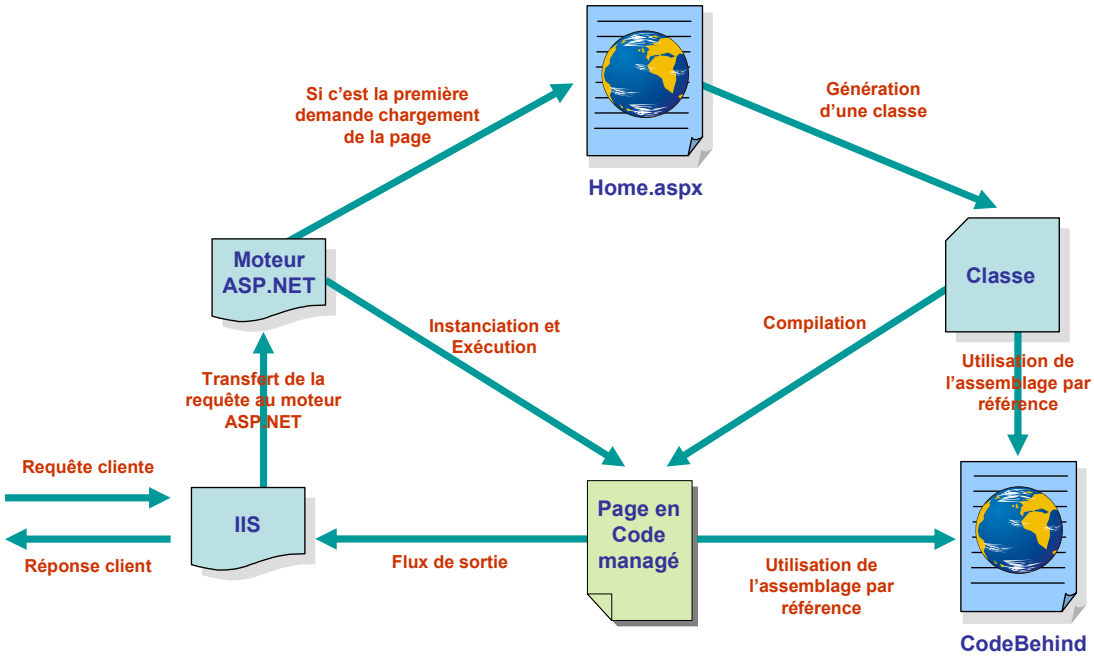

Figure C.4 - Processus de compilation à la volée des pages ASP.NET

Ce mécanisme a un coût lors du premier chargement lié à la compilation à la volée. Ensuite, la page compilée est mise en mémoire cache. Les demandes successives de la même page n'entraîneront plus de compilation tant que le serveur restera actif ou que l'application Web ne sera pas arrêtée. Une fois compilée, la page ASP.NET réagit comme un script ISAPI, avec les mêmes performances qu'un développement en C/C++.

En revanche, si le fichier est modifié, que ce soit au niveau du graphisme ou du code de présentation, le moteur ASP.NET déclenchera un nouveau cycle de compilation pour cette page lors de sa prochaine demande. Cette opération est transparente, elle ne nécessite pas l'arrêt de l'application Web et encore moins celui du serveur.

Une autre caractéristique des pages ASP.NET est qu'un modèle de programmation est

proposé pour faciliter la séparation du code de présentation lié aux balises HTML et du code applicatif, généralement écrit dans un langage avancé comme C#, VB.NET, etc. Ce modèle se nomme CodeBehind, il déporte le code applicatif dans un fichier externe qui porte le même nom, si ce n'est qu'il porte l'extension du langage utilisé.

C.3.3. Le pipeline HTTP

Une architecture en pipeline est une architecture axée sur un flux dans lequel des modules contenant des traitements peuvent interagir pour lire et modifier son contenu. L'intérêt principal de ce modèle est de proposer un mécanisme simple d'ajout et de suppression de traitement sur des données.

Le pipeline HTTP [116] est le mécanisme mis en place dans ASP.NET qui permet d'ajouter des modules de traitement des requêtes avant ou après leur traitement. Grâce à ce concept, ASP.NET est très évolutif. C'est d'ailleurs par ce biais que l'authentification au niveau de SOAP a facilement été ajoutée en ASP.NET. Grâce à ce système, de nouveaux protocoles peuvent être mis en place, ainsi que de nouveaux services, aussi bien techniques que fonctionnels, sans qu'il soit nécessaire de retoucher le code des applications.

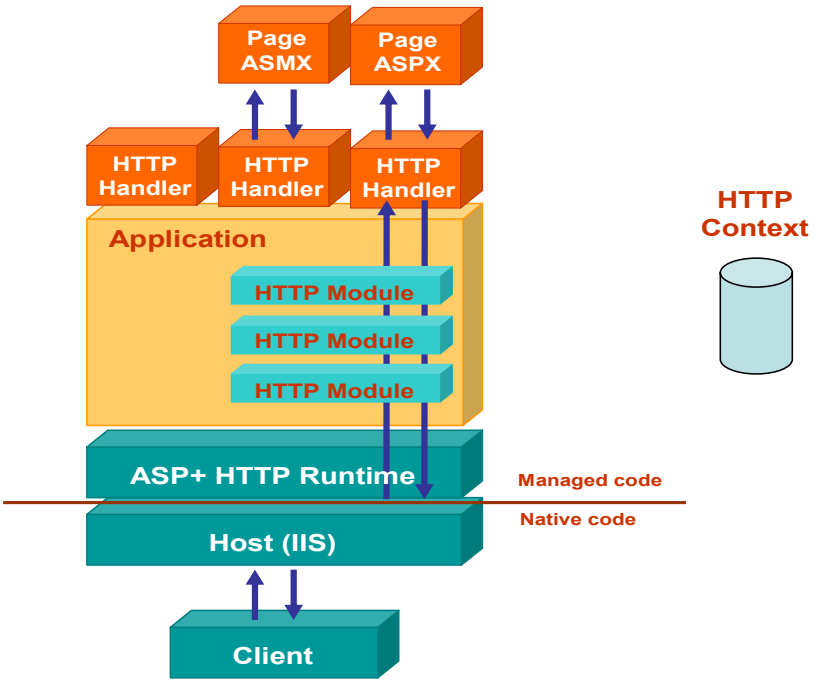

Figure C.5 - Le pipeline HTTP de ASP.NET

Le pipeline HTTP de ASP.NET repose sur deux types d'éléments :

- Les modules HTTP
- Les handlers HTTP.

Les modules HTTP permettent de réaliser des traitements pour toutes les requêtes clientes avant ou après que celles-ci soient traitées par un autre module ou un handler HTTP.

Quant aux handlers HTTP, ils permettent de produire une réponse à une requête cliente. Un handler définit une ressource particulière et généralement un URI correspond à un handler spécifique. Ainsi, il peut y avoir chaînage entre différents modules, mais les handlers ne se chaînent pas. Il s'agit du dernier composant de la chaîne HTTP. Par exemple, la page aspx est un type de handler HTTP.

C.3.4. La notion de handler HTTP

Généralement, un URI exprime la demande d'une ressource par un client. Cette demande porte généralement sur une page aspx, un service Web, ou tout objet disposant d'une

extension référencée dans ASP.NET. Si dans la majorité des cas la ressource existe physiquement sur le serveur, dans certains cas, elle peut être virtuelle.

Ainsi, pour des besoins particuliers, une nouvelle extension peut être déclarée dans le serveur IIS. Cette extension sera associée à un handler HTTP qui générera dynamiquement une page HTML en fonction du nom de la ressource. Typiquement, dans un modèle MVC, un contrôleur gère seul toutes les demandes. Dans ASP.NET, le contrôleur pourrait être défini avec un handler HTTP.

Physiquement, un handler HTTP est un composant défini dans un assemblage. Pour qu'il soit actif, un lien sera établi entre le composant et le nom de la ressource ou son extension. Par exemple, toutes les ressources d'extensions .aswa pourraient être traitées par un handler HTTP unique qui joue le rôle d'un environnement d'exécution (EE) d'un réseau actif.

C.4. LES WEB SERVICES

C.4.1. Introduction

Un Web Service est un composant applicatif accessible sur le Web, par l'entremise d'une interface standard, qui peut interagir dynamiquement avec d'autres applications en utilisant des protocoles de communication basés sur le XML, et cela indépendamment du système d'exploitation et des langages de programmation utilisés.

Les Web services s'appuient sur un ensemble de protocoles standardisant les modes d'invocation mutuels de composants applicatifs. Ils combinent les meilleurs aspects du développement à base de composants et du Web et présentent plusieurs caractéristiques:

- Réutilisabilité.
- Indépendance :
  - de la plateforme (UNIX, Windows, …),
  - de l'implémentation (VB, C#, Java, …) et
  - de l'architecture sous-jacente (.NET, J2EE, Axis…)

La technologie des Web Services est aujourd'hui de plus en plus incontournable et se présente comme le nouveau paradigme des architectures logicielles. Cette technologie englobe de nombreux concepts et tend à s'imposer comme le nouveau standard en terme d'intégration et d'échanges B2B (Business to Business).

C.4.2. Les objets répartis

Un objet réparti est un composant logiciel indépendant contenant sa propre intelligence de fonctionnement et capable d'interopérer avec d'autres objets distribués, indépendamment des types d'ordinateurs, des langages de programmation ayant servi à les développer, ou des systèmes d'exploitation.

A l'heure actuelle, trois systèmes sont fonctionnels:

- CORBA/CCM [15]: le middleware objet standard (Common Object Request Broker Architecture), dû à l'OMG (Object Management Group), un groupe de travail rassemblant de nombreux constructeurs et éditeurs de logiciel dans le but de fournir une base de travail indépendante de la plate-forme.
- (D)COM/MTS [17]: le middleware objet de Microsoft (Distributed Component Object Model) concurrent direct de CORBA que cherche à imposer Microsoft depuis les plate-formes de type Windows.
- Java/RMI [18] et Enterprise JavaBeans [21] (proposé par SunSoft, filiale logiciel de Sun) système d'objets distribués axé sur l'utilisation du langage Java. Son rapprochement avec le système CORBA est indéniable.

Dans un système d'objets distribués, les objets sont répartis dans une ou plusieurs applications sur une ou plusieurs machines. Ces objets ont été développés séparément, reliés entre eux par un réseau, et tournent sous des systèmes d'exploitation divers. Les messages doivent

traverser les frontières d'une application, circuler sur un réseau reliant des machines hétérogènes, et trouver sans erreur leur destinataire, c'est le problème de l'intégration de systèmes. Donc, il faut mettre en place une infrastructure informatique de distribution permettant la circulation des messages entre objets au travers d'un réseau.

C.4.3. Les intergiciels (Middleware)

En vue de gérer la complexité et l'extension des applications basées sur le réseau, les systèmes de middleware distribué ont émergé pour s'en occuper. **C**'est la couche logicielle qui donne l'impression à l'utilisateur de travailler sur un système monolithique alors qu'il accède à plusieurs serveurs. Le Middleware est une nouvelle catégorie de logiciel dont la fonction est d'assurer un lien entre des composants applicatifs distincts, sur des plateformes hétérogènes. Dans une architecture 3-tiers, le middleware occupe la position centrale.

Par rapport à une application, le middleware se présente comme une API (Application Programming Interface), c'est-à-dire un ensemble de fonctions. L'avantage de cette API est de fournir des fonctions plus adaptées et plus faciles à utiliser que s'il fallait utiliser directement les services du système d'exploitation et du réseau.

L'objectif principal du middleware est d'unifier, pour les applications, l'accès et la manipulation de l'ensemble des services disponibles sur le réseau, afin de rendre l'utilisation de ces derniers presque transparente.

Un middleware est susceptible de rendre les services suivants:

- Conversion : Service utilisé pour la communication entre machines mettant en œuvre des formats de données différents,
- Adressage : Permet d'identifier la machine serveur sur laquelle est localisé le service demandé afin d'en déduire le chemin d'accès. Dans la mesure du possible, cette fonction doit faire appel aux services d'un annuaire.
- Sécurité : Permet de garantir la confidentialité et la sécurité des données à l'aide de mécanismes d'authentification et de cryptage des informations.
- Communication : Permet la transmission des messages entre les deux systèmes sans altération. Ce service doit gérer la connexion au serveur, la préparation de l'exécution des requêtes, la récupération des résultats et la déconnexion de l'utilisateur.

Le middleware masque la complexité des échanges inter-applications et permet ainsi d'élever le niveau des API utilisées par les programmes. Sans ce mécanisme, la programmation d'une application client-serveur serait extrêmement complexe et rigide.

On distingue aujourd'hui quatre grandes classes de Middleware :

- l'exécution de transaction (Transactions Processing) qui est une classe de logiciel plutôt orienté "base de données".
- les RPC (Remote Procedure Calls) qui distribue l'exécution de routines sur un réseau.
- les MOM (Message oriented Middleware) qui permettent l'échange de données entre applications
- les ORB (Object Request Broker) qui permettent la distribution d'objets sur un réseau de machines.

C.4.4. Evolution du Web

A ses débuts, excepté pour le monde de la recherche, le modèle du Web se résumait uniquement à distribuer de l'information aux utilisateurs. Petit à petit, les échanges entre les applications intranet ont créé une démarche qui permet d'effectuer des transactions sur un socle Web. Il s'agissait d'abord uniquement de prélever une information d'un site pour l'exposer sur un autre. L'exemple le plus significatif est la récupération de la météo d'un site spécialisé pour l'afficher sur son portail. Cette opération était très barbare, on récupérait l'information en analysant la page HTML du site météo… Il n'y avait pas de standard d'échange des données.

Avant l'ère Internet, les échanges entre applications se réalisaient principalement avec des protocoles de type Remote Procedure Call (RPC) [120] tels que DCom[17], RMI [18] et Corba [15]. Mais face aux besoins entraînés par l'extension d'Internet, ces protocoles se sont heurtés à leur difficulté de mise en œuvre et au manque d'infrastructure nécessaire pour la sécurité des applications Web. Par conséquent, aucun protocole de type RPC ne s'est réellement imposé sur Internet, le seul standard dans cet environnement est le protocole HTTP.

Devant ce constat, certains acteurs de l'informatique tels qu'IBM et Microsoft ont soumis l'idée d'un protocole d'échange fondé sur le formalisme XML afin d'être universel. L'organisme de standardisation W3C a alors établi une équipe pour définir ce protocole. Après quelques mois, le message SOAP est apparu, ce qui a officiellement standardisé la notion de Web service, c'est-à-dire l'envoi de messages SOAP sur HTTP.

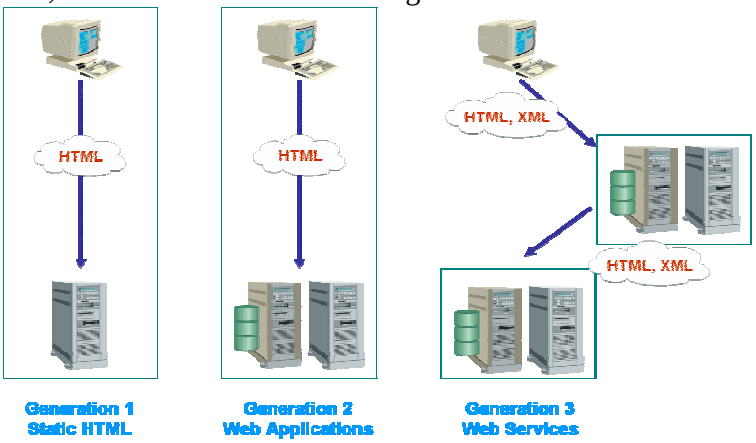

Figure C.6 - Evolution du Web

N'ayant pas de réel alternatif, les Web services s'imposent en tant que socle des interopérabilités entre différentes applications et différentes plateformes centrées sur les technologies Web.

En conclusion, les Web services permettent de remplacer les protocoles actuels (RPC, DCOM, RMI) par une approche entièrement ouverte et interopérable et permettent de faire interagir des composants hétérogènes, distants, et indépendants avec un protocole standard (SOAP).

Le tableau suivant permet de comparer les Web services avec les autres technologies des objets distribués :

	Web Services	**Autres technologies des objets distribués**
Invocation distante	SOA	IIOP ([121] page 15-51)
Description	WSDL	IDL ([121] page 3-1)
Enregistrement et découverte	UDDI	NameService [122]

C.4.5. Le modèle SOA (Service Oriented Architecture)

Les Web Services [25] disposent de leur propre modèle d'architecture orientée service (SOA). Ce modèle permet de mieux exploiter les applications et services existants et de les faire évoluer vers des Web Services.

Développer et intégrer des Web Services au sein d'une application n'implique pas de modifier ou changer l'architecture du système. Le modèle SOA [117] est tout simplement supporté par l'architecture en place.

Les Web services ont recours à un ensemble de standards : le langage XML [23] pour décrire les informations, la norme UDDI [28] pour trouver les services dont on a besoin, le langage WSDL [27] pour décrire leur interface et le protocole SOAP [26] pour les exécuter à distance. Ils sont référencés dans l'annuaire UDDI et permettent de créer des applications dis-

tribuées et accessibles depuis n'importe quel navigateur XML. Le concept de «Web service» repose sur le principe de la mise à disposition sur un site Web d'une fonction précise, sans repasser par le développement complet de l'application. Il permet de faciliter les échanges de données, mais aussi l'accès aux applications au sein des entreprises et surtout entre les entreprises. Il a pour but de permettre à une application de trouver automatiquement sur Internet le service dont elle a besoin et d'échanger des données avec lui. Pour faire appel à un service Web, il faut d'abord le découvrir (dans l'annuaire UDDI par exemple), récupérer la description XML de son interface et connaître le ou les protocoles de communication qu'il sait exploiter.

La Figure C.7 présente les grandes phases de la mise en œuvre d'un Web service. En préambule, le fournisseur réalise un service Web ainsi que son interface au format WSDL.

1. Publication (Etape 1): le fournisseur publie (enregistre) son Web service auprès d'un distributeur (sur un annuaire UDDI). Cette opération se fait en envoyant directement à l'annuaire un message UDDI (encapsulé dans une enveloppe SOAP) via un protocole de transport. Les informations fournies regroupent la localisation du service, la méthode d'invocation (et les paramètres associés) ainsi que le format de réponse. Toutes ces informations seront formalisées ensuite à l'aide de WSDL.
2. Recherche (Etape 2): le client effectue une recherche de Web service auprès du distributeur. Cette recherche se fait en envoyant un message UDDI (requête) encapsulé dans une enveloppe SOAP via un protocole de transport.
3. Résultat (Etape 3): le client reçoit en retour (via un protocole de transport) une réponse de l'annuaire. Cette réponse est un message WSDL encapsulé dans une enveloppe SOAP.
4. Interrogation (Etapes 4, 5 et 6): à l'aide de la réponse reçue, le client va pouvoir accéder directement au Web service chez le fournisseur. La demande de service s'effectue à l'aide d'un message SOAP via un protocole de transport.
5. Réponse (Etape 7): le client reçoit une réponse (via un protocole de transport) du Web service sous la forme d'un message SOAP. Il ne reste alors qu'à exploiter cette réponse.

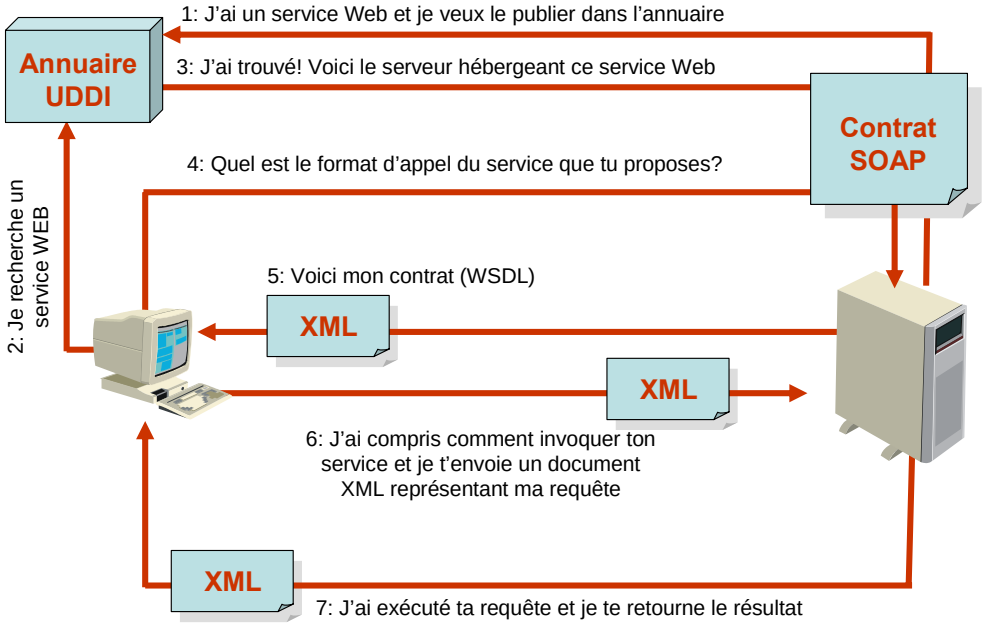

Figure C.7 - Cycle de vie d'utilisation d'un service Web

C.4.6. WSDL (Web Services Description Language)

WSDL (Web Services Description Language) [27] est un langage de description de Web Services, au format XML. Il permet de définir, de manière abstraite et indépendante du langage, l'ensemble des opérations et des messages qui peuvent être transmis vers et depuis un service Web donné. En d'autres termes, WSDL est à SOAP ce qu'IDL est à CORBA ou à COM.

Le WSDL décrit quatre ensembles de données importants:

- information d'interface décrivant toutes les fonctions disponibles publiquement,
- information de type de donnée pour toutes les requêtes de message et requêtes de réponse,
- information de liaison sur le protocole de transport utilisé,
- information d'adresse pour localiser le service spécifié.

Une fois qu'un Web Service est développé, il faut publier sa description et faire un lien vers elle dans un catalogue UDDI (Universal Description, Discovery and Integration), de sorte que les utilisateurs potentiels peuvent le trouver. Quand un utilisateur souhaite utiliser le service, il fait une demande de son fichier WSDL afin de connaître son emplacement, les appels de fonctions et comment y accéder. Ce qui lui permet de créer une classe Proxy appelée Proxy Web. A partir de cela, il peut composer, par exemple, une requête SOAP (Simple Object Access Protocol) vers l'ordinateur du service.

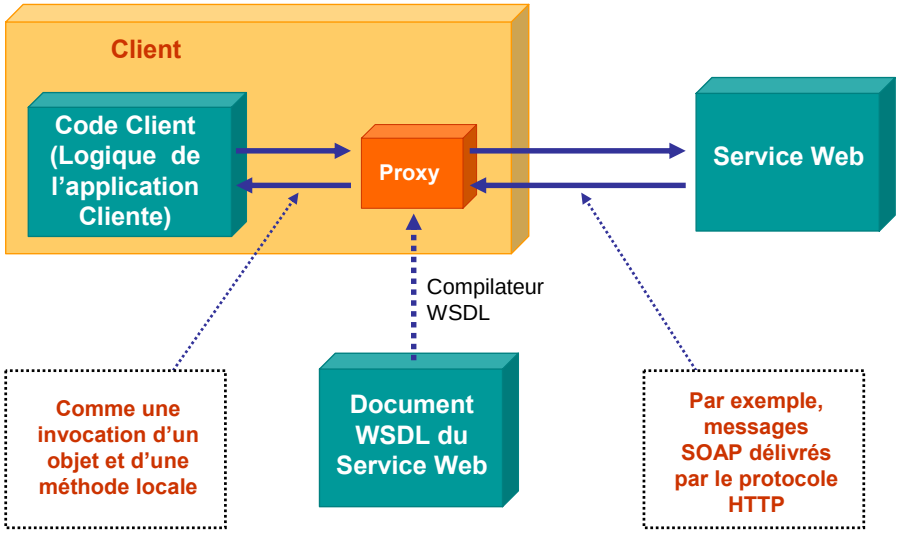

Figure C.8 - WSDL et Proxy Web

C.4.7. UDDI (Universal Discovery Description and Integration)

Universal Discovery Description and Integration [28] est l'équivalent des pages jaunes dans les Web services. Tout comme avec les pages jaunes classiques, vous pouvez rechercher une entreprise offrant les services dont vous avez besoin, consulter le service offert et contacter qui de droit pour en savoir plus. Bien entendu, vous pouvez proposer un service Web sans l'inscrire dans UDDI, exactement comme vous pouvez ouvrir une entreprise dans votre sous-sol et compter sur la publicité de bouche à oreille, mais si vous souhaitez toucher un marché plus important, UDDI permet à vos clients de vous trouver.

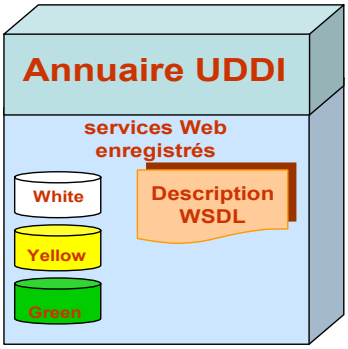

Figure C.9 - Annuaire UDDI

Une entrée du répertoire UDDI est constituée d'un fichier XML qui décrit une entreprise et les services qu'elle offre. Chaque entrée du répertoire UDDI est constituée de trois parties:

- Les "pages blanches" décrivent l'entreprise qui offre le service : nom, adresse, contacts, etc.

- Les "pages jaunes" comportent les catégories industrielles basées sur des classements standards tels que le système de classification de l'industrie d'Amérique du Nord et la classification industrielle standard.
- Les "pages vertes" décrivent l'interface vers le service avec suffisamment de détail pour qu'il soit possible d'écrire une application permettant d'utiliser le service Web.

Les services sont définis dans un document UDDI appelé *Type Model* ou *tModel*. Dans de nombreux cas, le *tModel* contient un fichier WSDL qui écrit une interface SOAP vers un Web service, mais le *tModel* est suffisamment souple pour décrire presque tous les types de services.

Le répertoire UDDI propose également de nombreuses façons pour rechercher les services qui vous permettront de créer vos applications. Par exemple, vous pouvez rechercher des fournisseurs de services dans une région géographique particulière ou une entreprise d'un type donné. Le répertoire UDDI fournit ensuite les informations, les contacts, les liens et les données techniques qui vous permettront d'estimer si les services correspondent à vos besoins.

C.4.8. Le protocole SOAP (Simple Object Access Protocol)

C.4.8.1. Généralités

SOAP (**S**imple **O**bject **A**ccess **P**rotocol) [26] est un protocole de transmission de messages. Il définit un ensemble de règles pour structurer des messages qui peuvent être utilisés dans de simples transmissions unidirectionnelles. Il n'est pas lié à un protocole particulier mais à HTTP qui est populaire. Il n'est pas non plus lié à un système d'exploitation ni à un langage de programmation, donc, théoriquement, les clients et serveurs de ces dialogues peuvent tourner sur n'importe quelle plate-forme et être écrits dans n'importe quel langage du moment qu'ils puissent formuler et comprendre des messages SOAP. SOAP serait donc un important composant de base pour développer des applications distribuées qui exploitent des fonctionnalités publiées comme services par des intranets ou internet.

SOAP définit un protocole permettant des appels de procédures à distances s'appuyant principalement sur le protocole HTTP et sur XML, mais aussi SMTP et POP. Il permet ainsi de définir des services WEB. Les paquets de données circulent sous forme de texte structuré au format XML (Extensible Markup Language).

C.4.8.2. Avantages

Le principal avantage de SOAP est qu'il repose sur 2 standards : XML (pour la structure des messages) et HTTP (pour le transport) bien qu'il soit utilisable avec d'autres protocoles de transport (Figure C.10). Par rapport à tous les autres protocoles, celui-ci présente l'avantage de l'interopérabilité, on est indépendant des plateformes et des langages de programmation.

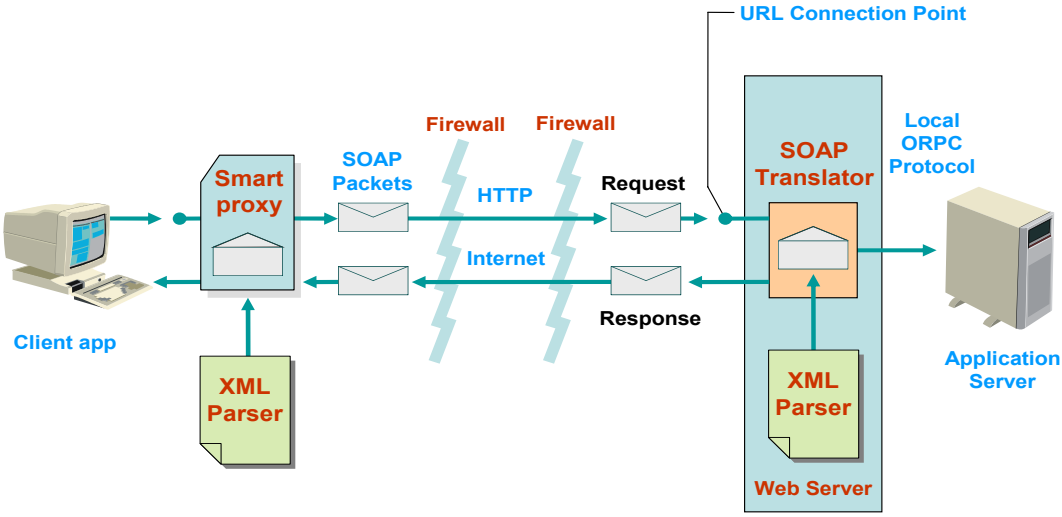

Figure C.10 - Application dans un contexte multi site

Le second avantage réside dans le déploiement des applications, principalement dans un contexte multi-sites (Figure C.10), pour communiquer entre 2 sociétés via Internet, c'est souvent mission-impossible d'utiliser autre chose que du HTTP ou du POP/SMTP à cause des Firewalls, car pour les autres protocoles il faut les reconfigurer, avec tous les trous de sécurité que cela peut engendrer, et cela implique souvent de longues négociations avec les administrateurs réseaux. SOAP permet de s'affranchir de tout ça.

C.4.8.3. Description

a. SOAP Framework

SOAP est constitué de plusieurs parties:

- La construction de l'enveloppe SOAP définit un cadre qui exprime le contenu d'un message, qui doit le traiter et si c'est optionnel ou obligatoire.
- Les règles de codage SOAP définissent un mécanisme de sérialisation qui peut être utilisé pour échanger des instances de data types d'application définies.
- La représentation RPC de SOAP définit une convention qui est utilisé pour représenter un appel de fonctions à distances et des réponses.

Comme illustré dans la Figure C.11, un message SOAP est un document XML contenant les éléments XML suivants:

- « SOAP Envelope », qui définit le contenu des messages.
- « SOAP Header », qui contient l'entête d'informations.
- « SOAP Body », qui contient les appels, les réponses et les erreurs.

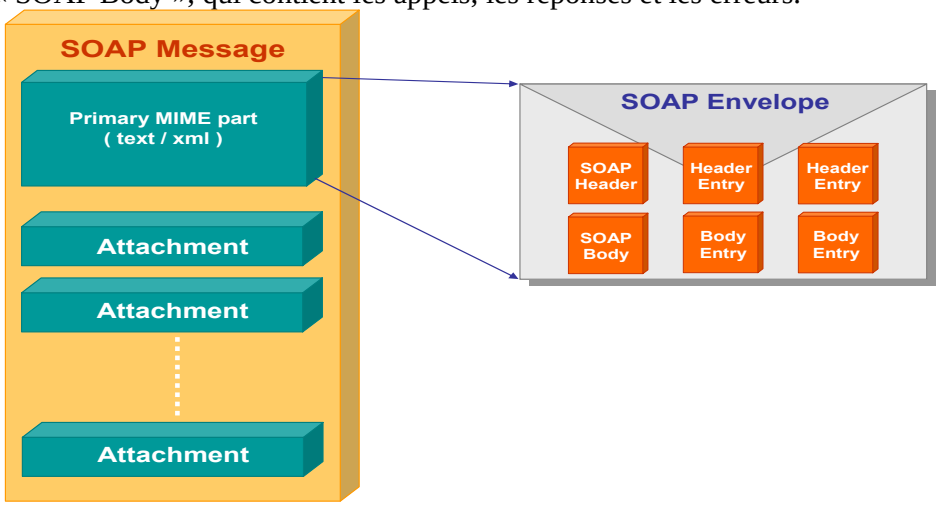

Figure C.11 - Message SOAP

b. Enveloppe SOAP

L'enveloppe est l'élément racine du document XML qui représente le message. L'élément «Enveloppe» doit être présent dans un message SOAP. Il peut contenir une ou plusieurs déclarations de librairies et des attributs supplémentaires. Dans l'enveloppe, on spécifie la manière d'interprétation du corps SOAP.

c. SOAP Header

L'élément «Header» est le premier fils après l'élément «Enveloppe SOAP».

C'est un élément optionnel, mais s'il existe il doit être le premier après le «root». Il est identifié par un URI et son nom local. Il permet d'étendre les messages d'une façon décentralisée et modulaire.

d. SOAP Body

L'élément «Body» marque la séparation entre les méta-données SOAP (*metadata) et les* données. Il doit exister dans un message SOAP et doit être un fils immédiat de SOAP Enveloppe. Il est identifie par un URI et son nom local et permet un mécanisme d'échange d'information vers un destinataire.

e. SOAP Fault

L'élément SOAP Fault est utilisé pour transporter les erreurs et les informations. Quand il existe, il doit apparaître, en tant qu'entrée dans le Body, une et une seule fois. Le SOAP Fault définit les « subelements » suivants:

Fault Element	Description
<faultcode>	Code identifiant l'erreur
<faultstring>	L'erreur n chaîne de caractère
<faultactor>	La cause de l'erreur
<detail>	Information supplémentaire

f. Exemple d'une requête SOAP

Voici une requête SOAP typique (avec les entêtes HTTP) pour un appel à une méthode RPC nommée EchoString, qui prend une chaîne comme paramètre :

```
POST /test/simple.asmx HTTP/1.1
Host: 131.107.72.13
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://soapinterop.org/echoString"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soapenc=http://schemas.xmlsoap.org/soap/encoding/
        xmlns:tns=http://soapinterop.org/
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<tns:echoString>
<inputString>ASWA</inputString>
</tns:echoString>
</soap:Body>
</soap:Envelope>
```

Le nom de la méthode est codé comme le XML : <tns:echoString> et le paramètre de type chaîne est codé comme <inputString>. La méthode C# que cela représente ressemble à ceci :

```
public String echoString(String inputString);
```

Et voici la réponse du serveur :

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
        xmlns:xsd=http://www.w3.org/2001/XMLSchema
        xmlns:soapenc=http://schemas.xmlsoap.org/soap/encoding/
        xmlns:tns="http://soapinterop.org/"
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<tns:echoStringResponse>
<Return>ASWA</Return>
</tns:echoStringResponse>
</soap:Body>
</soap:Envelope>
```

C.5. WSE (WEB SERVICE ENHANCEMENTS)

C.5.1. Introduction au Web Service Enhancements

Le *.Net framework* fournit nativement de nombreuses classes permettant de développer des applications qui consomment et produisent des Web services. Microsoft propose avec les Web Service Enhancements (WSE) [118] une amélioration au *.Net framework*. Il permet au développeur de construire des Web services sécurisés basés sur les derniers standards et spécifications en vigueur. Les WSE étendent les fonctionnalités de sécurité, de routage et d'intégration de pièces jointes dans des messages SOAP. Ils permettent aux développeurs de construire des applications basées sur les dernières spécifications publiées par Microsoft et différents acteurs de l'industrie (IBM, Verisign, SAP …) telles que WS-Security, WS-policy, WS-SecurityPolicy, WS-Trust, WS-SecureConversation et WS-Addressing. En utilisant le WSE pour les Web services XML on peut :

– Sécuriser les applications tout au long d'un domaine.
– Modifier d'une façon transparente, au niveau des nœuds, le chemin que peut prendre un message SOAP pour arriver au Web service.
– Attacher un fichier avec un message SOAP, durant une communication entre les services Web XML, sans le sérialiser en XML.

C.5.2. Architecture de WSE

Un message SOAP est constitué d'une enveloppe contenant le corps d'un message et d'un entête pour le décrire. L'entête est utilisé pour transmettre des informations sur la manière dont le message doit être traité: par exemple des informations de chiffrement et d'authentification.

WSE est constitué :

- d'un jeu de classes qui implémentent de nouveaux protocoles (WS-Security …)
- d'un jeu de filtres hébergés par ASP.Net qui interceptent les messages SOAP entrants et sortants.
- D'un contexte qui est le canal de communication entre une application (exemple un Web Service) et l'infrastructure (exemple un filtre). La classe SoapContext met dans l'entête l'état du programme.

Les filtres interprètent ou génèrent les entêtes permettant de prendre en charge les fonctionnalités requises. La Figure C.12 illustre le pipeline WSE et présente les différents types de filtres.

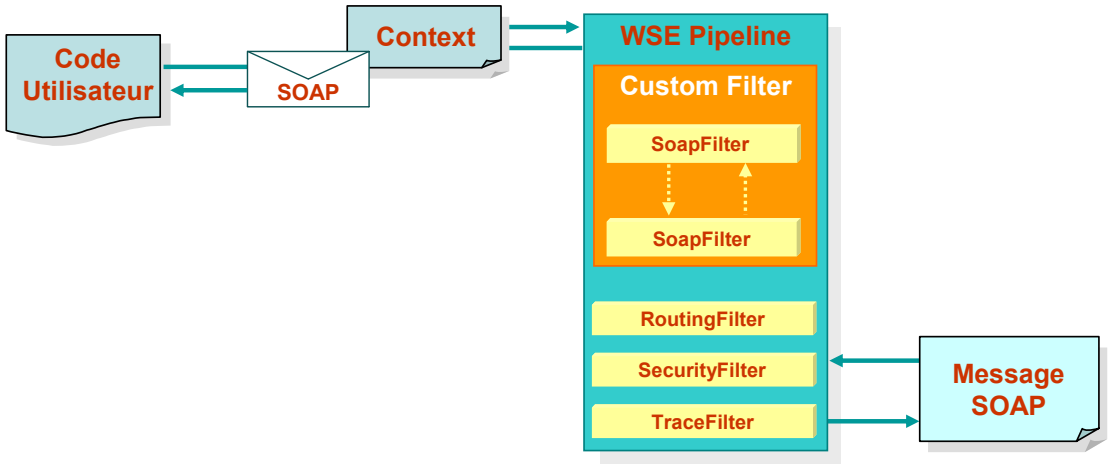

Figure C.12 - Pipeline WSE

Les filtres *WSE* entrants et sortants présentent les fonctionnalités de diagnostique, sécurité et routage (referral). De même, *WSE* permet à l'utilisateur de définir des filtres personnalisés (*custom filters*) à la sortie et à l'entrée des messages SOAP. Ces filtres font accès à n'importe quelle information contenue dans le message SOAP pour déterminer comment trai-

ter ce dernier. Plusieurs filtres personnalisés peuvent être définis dans un même pipeline.

Le paquet SOAP sortant du client (respectivement du Web service) sera intercepté par le pipeline WSE et traité par les différents filtres selon l'ordre précisé dans la Figure C.13 : «Custom Output Filters», «Routing Output Filter», «Security Output Filter» puis enfin le «Trace Output Filter», ce dernier une fois activé, il permet de sauvegarder dans un fichier de trace tous les messages SOAP traversant le pipeline WSE. Le paquet traité par le pipeline WSE sera transmis vers le réseau sous forme d'une requête (respectivement d'une réponse) SOAP. A l'arrivée du paquet à un service (respectivement à un client), il sera de nouveau intercepté par le pipeline WSE et traité par les mêmes filtres mais dans le sens inverse: le «Trace Input Filter», le «Security Input Filter», le «Routing Input Filter» et les «Custom Input Filters».

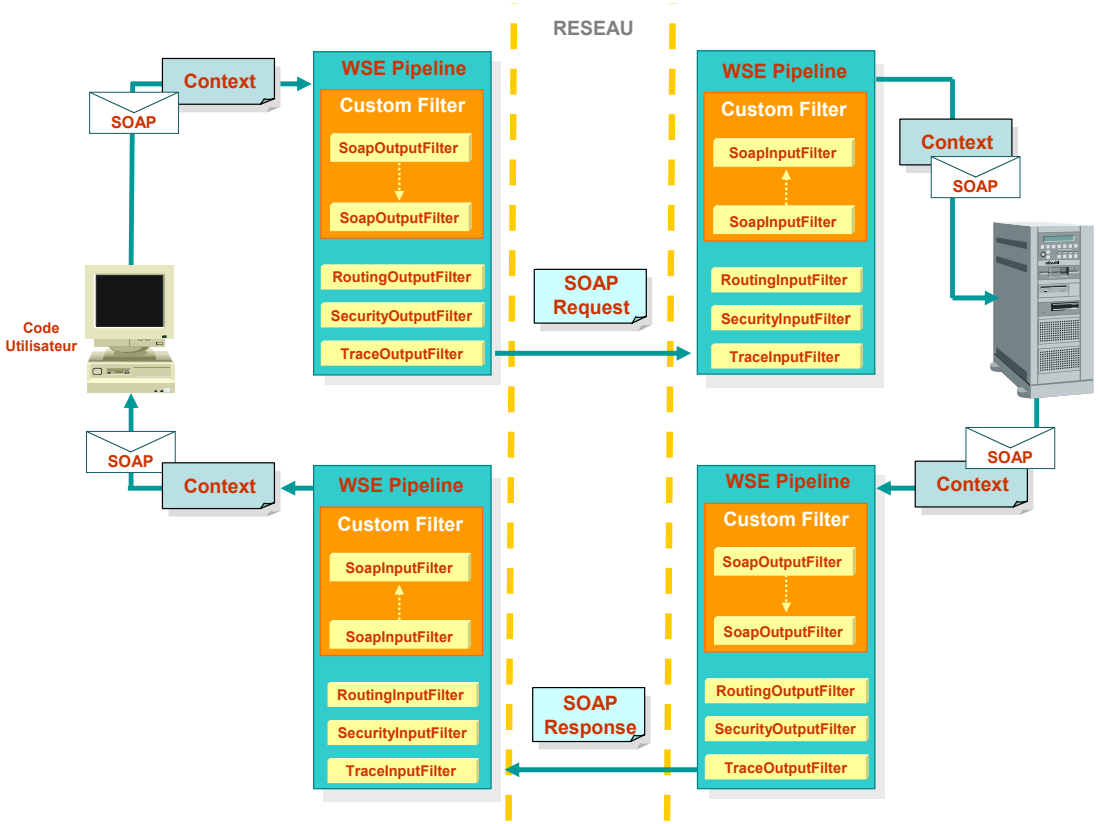

Figure C.13 - Traitement d'un message SOAP

C.5.3. WS-Routing

WS-Routing [30] est un simple protocole de routage des messages SOAP suivant une variété de protocole de transport comme TCP, UDP et HTTP. *WS-Routing* ajoute des fonctionnalités à SOAP en définissant un nouveau *chemin à* l'entête du message SOAP. Le tableau suivant montre les éléments ajoutés *à* l'entête du message SOAP et leurs rôles.

Element	***Description***
From	La source du message (optionnelle)
To	La destination (obligatoire)
Via	Un routeur intermédiaire (optionnel)
Fwd	La route dans le sens entrant (optionnelle)
Rev	La route dans le sens inverse (optionnelle)
Id	Elément d'identification unique d'un message (obligatoire)
relatedTo	Elément qui indique la relation avec d'autres messages (optionnel)
action	Elément qui indique le but du message (obligatoire)

Le tableau suivant montre les éléments retournés dans l'entête en cas d'erreur :

Element	*Description*
fault	Contient l'information sur l'erreur (obligatoire)
code	Rend le numéro de l'erreur spécifique (obligatoire)
reason	Retourne une phrase qui décrit la cause de l'erreur (obligatoire)

L'exemple ci-dessous nous montre une erreur :

```
<fault>
   <code>710</code>
   <reason>Endpoint not Found</reason>
   <endpoint>http://D.example.com/some/endpoint</endpoint>
</fault>
```

L'exemple suivant montre un message envoyé par un noeud A vers un destinataire D à travers B et C.

```
<S:Envelope
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header>
      <wsrp:path S:actor="http://schemas.xmlsoap.org/soap/actor/next"
            S:mustUnderstand="1" xmlns:wsrp="http://schemas.xmlsoap.org/rp">
         <wsrp:action>http://www.contoso.com/chat</wsrp:action>
         <wsrp:to>http://D.example.com/some/endpoint</wsrp:to>
         <wsrp:fwd>
            <wsrp:via>http://B.example.com</wsrp:via>
            <wsrp:via>http://C.example.com</wsrp:via>
         </wsrp:fwd>
         <wsrp:from>mailto:someone@example.com</wsrp:from>
         <wsrp:id>
            uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6
         </wsrp:id>
      </wsrp:path>
   </S:Header>
   <S:Body>
      ...
   </S:Body>
</S:Envelope>
```

C.5.4. WS-Security

WSE utilise les mécanismes définis dans les spécifications WS-Security [29] pour permettre à une application de sécuriser les Web services et :

- D'identifier un utilisateur qui demande l'exécution d'un Web service (authentification).
- De vérifier le rôle de l'utilisateur et les droits qui lui sont attribués (autorisation).
- De s'assurer qu'un message n'a pas été corrompu durant le transport (signature).
- De s'assurer que seul le destinataire d'un message peut le lire (chiffrement).

Un client doit obtenir un jeton de sécurité d'une source (qui doit avoir la confiance de l'expéditeur et du récepteur). Quand un client envoie une requête SOAP, les jetons de sécurité (*Security tokens*) sont placés dans le message SOAP. Quand le serveur Web reçoit la requête, le pipeline WSE vérifie que les jetons sont authentifiés avant d'envoyer le message vers le serveur Web et cela sans envoyer une requête vers le client pour vérifier l'intégrité du jeton de sécurité.

Le WSE fournit 3 méthodes pour sécuriser un message SOAP :

– Des jetons de sécurité pour identifier un utilisateur. Les informations permettant la mise en œuvre de la sécurité sont placées directement dans les entêtes SOAP.
  La signature numérique qui sécurise le Web service en permettant à un destinataire de vérifier que le message n'a pas été modifié depuis qu'il a été signé.

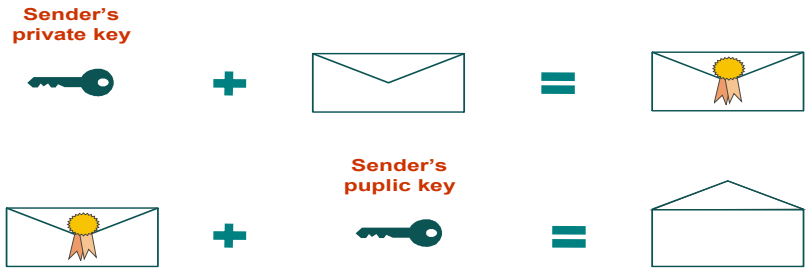

Figure C.14 - Signature

La Figure C.14 montre la clé nécessaire à l'expéditeur pour signer le message et la clé nécessaire au récepteur pour vérifier l'intégrité du message. Pour signer le message SOAP, l'expéditeur a besoin de la clé privée et pour vérifier la signature, le récepteur doit avoir la clé publique de l'expéditeur.

– Le chiffrement d'un message SOAP qui permet de sécuriser le message d'une façon extrême car le message ne pourra être lu qu'à l'arrivée à sa destination (Web service). La Figure C.15 montre les clés nécessaires à l'expéditeur et au récepteur pour chiffrer et déchiffrer le message. Pour chiffrer un message SOAP, l'expéditeur doit avoir la clé publique du récepteur qui à son tour doit posséder sa clé privée pour déchiffrer le message.

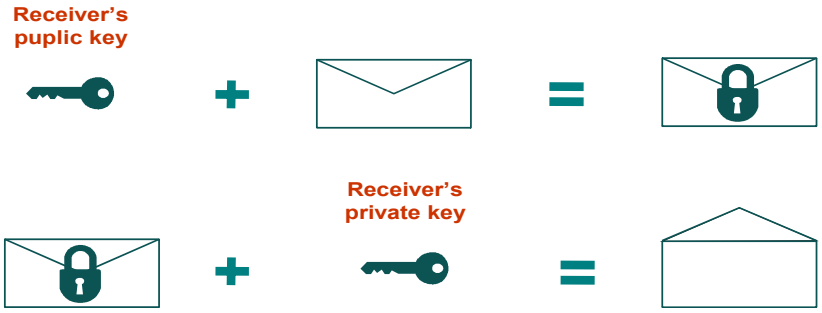

Figure C.15 - Chiffrement

La Figure C.16 montre que seulement le récepteur d'un message SOAP chiffré peut avoir accès à la partie chiffrée car il est le seul qui possède la clé privée.

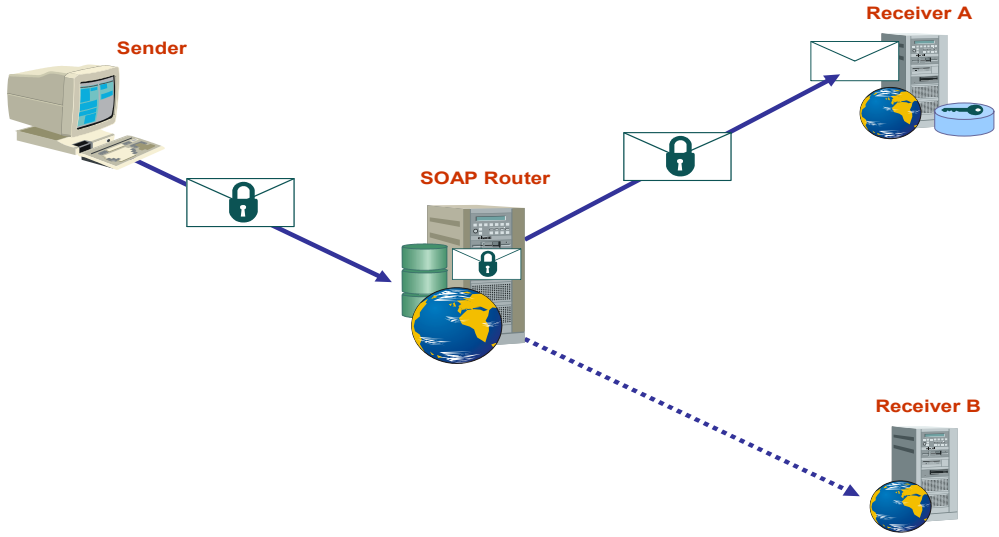

Figure C.16 - Message chiffré

C.5.5. Signature numérique

Le WSE fournit un mécanisme pour signer un message SOAP en utilisant un *Username Token* ou un certificat X.509 [119] comme on peut utiliser un *custom binary security tokens*. Quand le message est signé par un *Username Token*, le client fournit un *username* et un mot de passe. La signature numérique peut vérifier si le message n'a pas été modifié, mais elle ne chiffre pas le message; le message est transmis en texte clair.

C.5.6. Chiffrement

Le WSE permet au Web Services, créé par ASP.Net, et à leurs clients de chiffrer et déchiffrer les messages SOAP échangés pour communiquer avec un Web service. Les messages SOAP sont transmis en texte clair par défaut ce qui permet à n'importe quel récepteur de lire le message. Mais avec un message SOAP chiffré on s'assure que seul le récepteur désigné va déchiffrer le message et va pouvoir le lire.

Le WSE supporte le chiffrement symétrique et asymétrique. Le chiffrement asymétrique permet au client d'un Web service de chiffrer le message en utilisant la clé publique d'un certificat X.509, d'une manière que seul le propriétaire de la clé privée du certificat X.509 peut déchiffrer le message SOAP. Le chiffrement symétrique exige la possession d'un secret partagé par le Web service et son client. Par suite le client chiffre le message SOAP par le secret partagé et le Web service déchiffre par le même secret.

Par défaut, le WSE chiffre la partie **<Body >** d'un message, mais on peut spécifier la partie d'un message SOAP à chiffrer.

C.5.7. Exemple d'un message chiffré et signé

Le fichier suivant est un exemple d'un paquet SOAP signé et chiffré au sein de la plateforme ASWA:

```
<soap:Envelope xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
        xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
       xmlns:wsse=http://docs.oasis-open.org/wss/2004/01/oasis-200401-
            wss-wssecurity-secext-1.0.xsd
       xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
            wss-wssecurity-utility-1.0.xsd">
  <soap:Header>
       <wsa:Action wsu:Id="Id-164af08f-efd0-4f96-a9c6-9b17ba09a55e">
            http://tempuri.org/addNode
       </wsa:Action>
       <wsa:MessageID wsu:Id="Id-1574c04c-0254-4434-94cf-1c1e518eb724">
            uuid:31569e97-c4d1-4e8a-8e4d-e310b1d3f761</wsa:MessageID>
       <wsa:ReplyTo wsu:Id="Id-e336cb87-4771-47ea-9149-5986401351a0">
            <wsa:Address>
                 http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
            </wsa:Address>
       </wsa:ReplyTo>
       <wsa:To wsu:Id="Id-6b9d4103-5715-452c-8ad9-449154dce5dc">
            http://10.10.10.15/ASWAControl/ASWANodes.asmx
       </wsa:To>
       <wsse:Security soap:mustUnderstand="1">
            <wsu:Timestamp wsu:Id="Timestamp-ac1735ec-f239-449c-9307-78455ed41d1c">
                 <wsu:Created>2005-07-15T18:47:23Z</wsu:Created>
                 <wsu:Expires>2005-07-15T18:48:23Z</wsu:Expires>
            </wsu:Timestamp>
            <wsse:BinarySecurityToken
                 ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-
                      200401-wss-x509-token-profile-1.0#X509v3"
                 EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-
                      200401-wss-soap-message-security-1.0#Base64Binary"
                 xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
                      200401-wss-wssecurity-utility-1.0.xsd"
                 wsu:Id="SecurityToken-62106891-e910-4426-b24e-9cf6741a3e78">
                 MIICYTCCAcqgAwIBAgIDDyJnMA0GCSqGSIb3DQEBBAUAMGIxCzAJBg
```

```
            NVBAYTAlpBMSUwIwYDVQQKExxUaGF3dGUgQ29uc3Vsd
            …
            AbJHwFAzGDZZDAGoIRASxGMEVHJspEKklvp28nHavs527lDI/tTZ0b1w==
        </wsse:BinarySecurityToken>
        <wsse:BinarySecurityToken
            ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-
                    200401-wss-x509-token-profile-1.0#X509v3"
            EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-
                    200401-wss-soap-message-security-1.0#Base64Binary"
            xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
                    200401-wss-wssecurity-utility-1.0.xsd"
            wsu:Id="SecurityToken-1d678e18-1fa5-446b-aeb7-4f2561122384">
            MIICYTCCAcqgAwIBAgIDDyJnMA0GCSqGSIb3DQEBBAUAMGIxCzAJBg
            NVBAYTAlpBMSUwIwYDVQQKExxUaGF3dGUgQ29uc3Vsd
            …
            AbJHwFAzGDZZDAGoIRASxGMEVHJspEKklvp28nHavs527lDI/tTZ0b1w==
        </wsse:BinarySecurityToken>
        <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
          <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
            <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
              <wsse:SecurityTokenReference>
                <wsse:Reference
                    URI="#SecurityToken-1d678e18-1fa5-446b-aeb7-4f2561122384"
                    ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-
                            200401-wss-x509-token-profile-1.0#X509v3" />
              </wsse:SecurityTokenReference>
            </KeyInfo>
            <xenc:CipherData> <xenc:CipherValue>
                    DzHBfKEZ2XAT/kjAl6vsWnczS3ezt6wEWHuPR60o9u8m
                    FHkbRLDr4MlA3qMYvrrWshgCrY+6/p1jbZEcJr49IhXj+N
                    n+hgxX5GDZCnTuNbZzYW8QYw+diMLqICy6hpl38T5/C
                    0YY86D5FU6/g6NdqpsQsb0M4cgMeau2InsCDhg=
                </xenc:CipherValue> </xenc:CipherData>
      <xenc:ReferenceList>
        <xenc:DataReference URI="#EncryptedContent-ed7286ea-667c-4af0-9a82-613f3eebec33" />
      </xenc:ReferenceList>
    </xenc:EncryptedKey>
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#"> <SignedInfo>
            <ds:CanonicalizationMethod Algorithm=http://www.w3.org/2001/10/xml-exc-c14n#
                    xmlns:ds="http://www.w3.org/2000/09/xmldsig#" />
            <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
            <Reference URI="#Id-164af08f-efd0-4f96-a9c6-9b17ba09a55e">
                <Transforms>
                    <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
                </Transforms>
                <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
                <DigestValue>fdEcDVLjpEfY1KSEbTtpG2+Zt+0=</DigestValue>
            </Reference>
            <Reference URI="#Id-1574c04c-0254-4434-94cf-1c1e518eb724">
                <Transforms>
                    <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
                </Transforms>
                <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
                <DigestValue>Hl+Si78ZPFF1earcZegt48YWbZQ=</DigestValue>
            </Reference>
            <Reference URI="#Id-e336cb87-4771-47ea-9149-5986401351a0">
                <Transforms>
```

```
                        <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
                    </Transforms>
                    <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
                    <DigestValue>DHwUGzxhvFLTyN+XDsucpG1MZBY=</DigestValue>
                </Reference>
                <Reference URI="#Id-6b9d4103-5715-452c-8ad9-449154dce5dc">
                    <Transforms>
                        <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
                    </Transforms>
                    <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
                    <DigestValue>eqzwEXn/Zj4ONFIJtzU2/2fUjKo=</DigestValue>
                </Reference>
                <Reference URI="#Timestamp-ac1735ec-f239-449c-9307-78455ed41d1c">
                    <Transforms>
                        <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
                    </Transforms>
                    <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
                    <DigestValue>vRKssEJb+wpPvhQ46d+mU3lfi0A=</DigestValue>
                </Reference>
                <Reference URI="#Id-114c8848-78e1-47eb-b5ce-19353c51004c">
                    <Transforms>
                        <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
                    </Transforms>
                    <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
                    <DigestValue>6XE3SZgtKS4k11SBqn/m9uXC7t0=</DigestValue>
                </Reference>
            </SignedInfo> <SignatureValue>
                CNc+hj5GYMJBV766yjIqxxluxlnI18p4No/ggPdEshIB0zOk8Mpo5o6OR40
                vxaCCbciMmdhburhaBiyOeZHWublulDLEg6ykrfhY2zMRikOnfKuZx+jLU
                ODorx8xrW4HPVDyu1D0CntZqppp59qxPpgXwdXgldSfPSHpxLGgypI=
            </SignatureValue>
            <KeyInfo> <wsse:SecurityTokenReference>
                <wsse:Reference
                        URI="#SecurityToken-62106891-e910-4426-b24e-9cf6741a3e78"
                        ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-
                                200401-wss-x509-token-profile-1.0#X509v3" />
                </wsse:SecurityTokenReference> </KeyInfo>
        </Signature>
        </wsse:Security>
    </soap:Header>
    <soap:Body wsu:Id="Id-114c8848-78e1-47eb-b5ce-19353c51004c">
        <xenc:EncryptedData Id="EncryptedContent-ed7286ea-667c-4af0-9a82-613f3eebec33"
            Type=http://www.w3.org/2001/04/xmlenc#Content
            xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
        <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc" />
        <xenc:CipherData>  <xenc:CipherValue>
                    YOpl9ot+rvRDsd0cIp4y/vrxQo9dGKbf6EweZeEUV3Vq
                    +rOhTY8QbuxmTHDm4rWlmxQ/k/e1Oh+WbXG6GhEb
                    7DlmMP8v56Y916DUzsfGdJurkMJvtv3UIFgXxv16MrjU
                </xenc:CipherValue> </xenc:CipherData>
    </xenc:EncryptedData>
    </soap:Body>
</soap:Envelope>
```

Annexe D

EXEMPLES DE REPRESENTATION DES PIBS

Cette annexe contient la représentation détaillée des deux PIBs qu'on a implémentées pour valider notre travail. Ces deux PIBs étant la PIB DiffServ et la PIB de Filtrage. Dans les sections suivantes, on va présenter chaque PIB en ASN.1 et son équivalent en OWL.

D.1. LA PIB DIFFSERV

D.1.1. La représentation en ASN.1

```
qosPolicyPib  MODULE-IDENTITY
 LAST-UPDATED "200107201100Z"
   ORGANIZATION "IETF DIFFSERV WG"
   CONTACT-INFO "
            Michael Fine
            Cisco Systems, Inc.
            170 West Tasman Drive
            San Jose, CA  95134-1706 USA
            Phone: +1 408 527 8218
            Email: mfine@cisco.com

            Keith McCloghrie
            Cisco Systems, Inc.
            170 West Tasman Drive,
            San Jose, CA 95134-1706 USA
            Phone: +1 408 526 5260
            Email: kzm@cisco.com

            John Seligson
            Nortel Networks, Inc.
            4401 Great America Parkway
            Santa Clara, CA 95054 USA
            Phone: +1 408 495 2992
            Email: jseligso@nortelnetworks.com"
   DESCRIPTION
        "The PIB module containing a set of provisioning classes
        that describe quality of service (QoS) policies for
        DiffServ. It includes general classes that may be extended
        by other PIB specifications as well as a set of PIB
        classes related to IP processing."
::= { pib xxx }

qosCapabilityClasses  OBJECT IDENTIFIER ::={qosPolicyPib 1}
qosPolicyClasses   OBJECT IDENTIFIER ::= {qosPolicyPib 2}
qosPolicyParameters OBJECT IDENTIFIER ::= {qosPolicyPib 3 }
```

```
                                                                    …
--
-- The Base Capability Table
--
qosBaseIfCapsTable OBJECT-TYPE
   SYNTAX        SEQUENCE OF QosBaseIfCapsEntry
   PIB-ACCESS    notify, 3
   STATUS        current
   DESCRIPTION
      "The Base Interface Capability class.  This class represents
       a generic capability supported by a device in the ingress,
       egress or both directions."
   ::= { qosCapabilityClasses 1 }

qosBaseIfCapsEntry OBJECT-TYPE
   SYNTAX        QosBaseIfCapsEntry
   STATUS        current
   DESCRIPTION
      "An instance of this class describes the qosBaseIfCaps class."
   PIB-INDEX { qosBaseIfCapsPrid }
::= { qosBaseIfCapsTable 1 }

QosBaseIfCapsEntry ::= SEQUENCE {
      qosBaseIfCapsPrid InstanceId,
      qosBaseIfCapsDirection Integer32
}

qosBaseIfCapsPrid OBJECT-TYPE
   SYNTAX        InstanceId
   STATUS        current
   DESCRIPTION
      "An arbitrary integer index that uniquely identifies an
      instance of the class."
   ::= { qosBaseIfCapsEntry 1 }

qosBaseIfCapsDirection OBJECT-TYPE
   SYNTAX        Integer32 {
                    inbound(1),
                    outbound(2),
                    inAndOut(3)
                 }
   STATUS        current
   DESCRIPTION
      "This object specifies the direction(s) for which the capability
       applies.  A value of 'inbound(1)' means the capability applies
       only to the ingress direction.  A value of 'outbound(2)' means
       the capability applies only to the egress direction.  A value of
       'inAndOut(3)' means the capability applies to both directions."
   ::= { qosBaseIfCapsEntry 2 }
                                                                    …
```

Figure D.1 - Représentation en ASN.1 d'une partie de la PIB DiffServ

D.1.2. Représentation en OWL

```
<owl:ObjectProperty rdf:ID="qosBaseIfCapsDirection_Attribute">
  <rdfs:domain>
     <OT_Entry_PibIndex rdf:ID="qosBaseIfCapsEntry">
```

```
        <rdfs:subClassOf>
          <OT_Table rdf:ID="qosBaseIfCapsTable">
            <rdfs:subClassOf>
            <PibMetaClass rdf:ID="qosCapabilityClasses">
            <rdfs:subClassOf>
              <ModuleIdentity rdf:ID="qosPolicyPib">
                <Description rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                  The PIB module containing a set of provisioning classes that describe
                  quality of  service (QoS) policies for DiffServ. It includes general
                  classes that may be extended by other PIB specifications as well as
                  a set of PIB classes related to IP processing.
                </Description>
                <LastUpdated rdf:datatype=http://www.w3.org/2001/XMLSchema#dateTime>
                  2005-09-09
                </LastUpdated>
                <Organization rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
                  IETF DIFFSERV WG
                </Organization>
                <rdfs:subClassOf rdf:resource="#PIB"/>
                <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">2</OID>
              </ModuleIdentity>
            </rdfs:subClassOf>
            <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">2.1</OID>
          </PibMetaClass>
         </rdfs:subClassOf>
         <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">2.1.1</OID>
         <OT_Syntax rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
              sequence of QosBaseIfCapsEntry
        </OT_Syntax>
        <OT_Status> <SPPI_Status rdf:ID="current"/> </OT_Status>
        <OT_Description rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
              The Base Interface Capability class. This class represents a generic capability
              supported by a device in the ingress, egress or both directions.
          </OT_Description>
          </OT_Table>
        </rdfs:subClassOf>
        <OT_Syntax rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
          QosBaseIfCapsEntry
        </OT_Syntax>
        <OT_Description rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
          An instance of this class describes the qosBaseIfCaps class.
        </OT_Description>
        <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">2.1.1.1</OID>
        <OT_Status rdf:resource="#current"/>
    </OT_Entry_PibIndex>
  </rdfs:domain>
 <rdfs:range>
     <OT_Attribute rdf:ID="qosBaseIfCapsDirection">
        <rdfs:subClassOf>
          <OT_Attribute rdf:ID="qosBaseIfCapsEntryAttributes">
            <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">2.1.1.2</OID>
            <rdfs:subClassOf rdf:resource="#qosBaseIfCapsTable"/>
          </OT_Attribute>
        </rdfs:subClassOf>
        <OT_Status rdf:resource="#current"/>
        <OT_Description rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
          This object specifies the direction(s) for which the capability applies.
```

```
            A value of 'inbound(1)' means the capability applies only to the ingress direction.
            A value of 'outbound(2)' means the capability applies only to the egress direction.
            A value of 'inAndOut(3)' means the capability applies to both directions.
        </OT_Description>
        <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">2.1.1.2.1</OID>
        <OT_Syntax rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
              Integer32
        </OT_Syntax>
      </OT_Attribute>
  </rdfs:range>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:ObjectProperty>
```

Figure D.2 - Représentation en OWL d'une partie de la PIB DiffServ

D.2. LA PIB DE FILTRAGE

D.2.1. Représentation en ASN.1

```
ipv4FilterIpFilter OBJECT IDENTIFIER ::= { someExampleOID 1 }
  -- The IP Filter Table
  ipv4FilterTable OBJECT-TYPE
     SYNTAX        SEQUENCE OF Ipv4FilterEntry
     MAX-ACCESS    not-accessible
     STATUS        current
     DESCRIPTION
        "Filter definitions.  A packet has to match all fields in a filter.
         Wildcards may be specified for those fields that are not relevant."
     ::= { ipv4FilterIpFilter 1 }

  ipv4FilterEntry OBJECT-TYPE
     SYNTAX        Ipv4FilterEntry
     MAX-ACCESS    not-accessible
     STATUS        current
     DESCRIPTION
        "An instance of the filter class."
     INDEX { ipv4FilterIndex }
     ::= { ipv4FilterTable 1 }

  Ipv4FilterEntry ::= SEQUENCE {
        ipv4FilterIndex       Unsigned32,
        ipv4FilterDstAddr     IpAddress,
        ipv4FilterDstAddrMask  IpAddress,
        ipv4FilterSrcAddr     IpAddress,
        ipv4FilterSrcAddrMask  IpAddress,
        ipv4FilterDscp        Integer32,
        ipv4FilterProtocol    Integer32,
        ipv4FilterDstL4PortMin Integer32,
        ipv4FilterDstL4PortMax Integer32,
        ipv4FilterSrcL4PortMin Integer32,
        ipv4FilterSrcL4PortMax Integer32,
        ipv4FilterPermit      TruthValue
  }

  ipv4FilterIndex OBJECT-TYPE
     SYNTAX        Unsigned32
     MAX-ACCESS    read-write
     STATUS        current
     DESCRIPTION
```

```
      "An integer index to uniquely identify this filter among all the filters."
    ::= { ipv4FilterEntry 1 }

  ipv4FilterDstAddr OBJECT-TYPE
    SYNTAX        IpAddress
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
     "The IP address to match against the packet's destination IP address."
    ::= { ipv4FilterEntry 2 }

  ipv4FilterDstAddrMask OBJECT-TYPE
    SYNTAX        IpAddress
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
     "A mask for the matching of the destination IP address.
     A zero bit in the mask means that the corresponding bit in the address always matches."
    ::= { ipv4FilterEntry 3 }

  ipv4FilterSrcAddr OBJECT-TYPE
    SYNTAX        IpAddress
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
       "The IP address to match against the packet's source IP address."
    ::= { ipv4FilterEntry 4 }

  ipv4FilterSrcAddrMask OBJECT-TYPE
    SYNTAX        IpAddress
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
       "A mask for the matching of the source IP address."
    ::= { ipv4FilterEntry 5 }

  ipv4FilterDscp OBJECT-TYPE
    SYNTAX        Integer32 (-1 | 0..63)
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
       "The value that the DSCP in the packet can have and match.
         A value of -1 indicates that a specific
        DSCP value has not been defined and thus all DSCP values are considered a match."
    ::= { ipv4FilterEntry 6 }

  ipv4FilterProtocol OBJECT-TYPE
    SYNTAX        Integer32 (0..255)
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
       "The IP protocol to match against the packet's protocol. A value of zero means match all."
    ::= { ipv4FilterEntry 7 }

  ipv4FilterDstL4PortMin OBJECT-TYPE
    SYNTAX        Integer32 (0..65535)
    MAX-ACCESS    read-write
```

```
      STATUS        current
      DESCRIPTION
         "The minimum value that the packet's layer 4 destination port number
          can have and match this filter."
      ::= { ipv4FilterEntry 8 }

   ipv4FilterDstL4PortMax OBJECT-TYPE
      SYNTAX        Integer32 (0..65535)
      MAX-ACCESS    read-write
      STATUS        current
      DESCRIPTION
      "The maximum value that the packet's layer 4 destination port number
       can have and match this filter."
      ::= { ipv4FilterEntry 9 }

   ipv4FilterSrcL4PortMin OBJECT-TYPE
      SYNTAX        Integer32 (0..65535)
      MAX-ACCESS    read-write
      STATUS        current
      DESCRIPTION
         "The minimum value that the packet's layer 4 source port number can have
          and match this filter."
      ::= { ipv4FilterEntry 10 }

   ipv4FilterSrcL4PortMax OBJECT-TYPE
      SYNTAX        Integer32 (0..65535)
      MAX-ACCESS    read-write
      STATUS        current
      DESCRIPTION
         "The maximum value that the packet's layer 4 source port number can have
          and match this filter."
      ::= { ipv4FilterEntry 11 }

   ipv4FilterPermit OBJECT-TYPE
      SYNTAX        TruthValue
      MAX-ACCESS    read-write
      STATUS        current
      DESCRIPTION
         "If false, the evaluation is negated.  That is, a valid match will be evaluated as not a match
          and vice versa."
      ::= { ipv4FilterEntry 12 }
```

Figure D.3 - PIB de Filtrage en ASN.1

D.2.2. Représentation en OWL

```
<owl:ObjectProperty rdf:ID="ipv4FilterDstAddr">
  <rdfs:domain>
    <OT_Entry_PibIndex rdf:ID="ipv4FilterEntry">
    <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.1.1.1</OID>
    <rdfs:subClassOf>
      <OT_Table rdf:ID="ipv4FilterTable">
      <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.1.1</OID>
      <OT_Description rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
        Filter definitions.  A packet has to match all fields in a filter.
        Wildcards may be specified for those fields that are not relevant.
      </OT_Description>
      <rdfs:subClassOf>
```

```
        <PibMetaClass rdf:ID="ipv4FilterIpFilter">
        <rdfs:subClassOf>
          <ModuleIdentity rdf:ID="ipv4FilterPib">
            <rdfs:subClassOf rdf:resource="#PIB"/>
            <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1</OID>
          </ModuleIdentity>
        </rdfs:subClassOf>
        <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.1</OID>
      </PibMetaClass>
    </rdfs:subClassOf>
    <OT_Syntax rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
      SEQUENCE OF Ipv4FilterEntry
    </OT_Syntax>
    </OT_Table>
    </rdfs:subClassOf>
    <OT_Description rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
        An instance of the filter class.
    </OT_Description>
    <OT_Syntax rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
      Ipv4FilterEntry
    </OT_Syntax>
  </OT_Entry_PibIndex>
  </rdfs:domain>
  <rdfs:range>
    <OT_Attribute rdf:ID="_ipv4FilterDstAddr">
    <OT_Status> <SPPI_Status rdf:ID="current"/> </OT_Status>
    <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.1.1.2.2</OID>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty> <owl:FunctionalProperty rdf:ID="VALUE"/> </owl:onProperty>
        <owl:allValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <OT_Description rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
      The IP address to match against the packet's destination IP address.
    </OT_Description>
    <OT_Syntax
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">IpAddress</OT_Syntax>
    <rdfs:subClassOf>
      <OT_Attribute rdf:ID="ipv4FilterEntryAttributes">
        <rdfs:subClassOf rdf:resource="#ipv4FilterTable"/>
        <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.1.1.2</OID>
      </OT_Attribute>
    </rdfs:subClassOf>
    </OT_Attribute>
  </rdfs:range>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="ipv4FilterIndex">
  <rdfs:range>
    <OT_Attribute rdf:ID="_ipv4FilterIndex">
    <rdfs:subClassOf rdf:resource="#ipv4FilterEntryAttributes"/>
    <OT_Syntax rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Unsigned32
    </OT_Syntax>
    <OT_Description rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
      An integer index to uniquely identify this filter among all the filters.
```

```
      </OT_Description>
      <OT_Status rdf:resource="#current"/>
      <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.1.1.2.6</OID>
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:allValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
          <owl:onProperty> <owl:FunctionalProperty rdf:about="#VALUE"/> </owl:onProperty>
        </owl:Restriction>
       </rdfs:subClassOf>
      </OT_Attribute>
    </rdfs:range>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="#ipv4FilterEntry"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="ipv4FilterSrcL4PortMin">
    <rdfs:range>
      <OT_Attribute rdf:ID="_ipv4FilterSrcL4PortMin">
      <OT_Description rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
        The minimum value that the packet's layer 4 source port number can have and match this filter.
      </OT_Description>
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty> <owl:FunctionalProperty rdf:about="#VALUE"/> </owl:onProperty>
          <owl:allValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
        </owl:Restriction>
      </rdfs:subClassOf>
      <OT_Syntax rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        Integer32 (0..65535)
      </OT_Syntax>
      <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.1.1.2.12</OID>
      <OT_Status rdf:resource="#current"/>
      <rdfs:subClassOf rdf:resource="#ipv4FilterEntryAttributes"/>
      </OT_Attribute>
    </rdfs:range>
    <rdfs:domain rdf:resource="#ipv4FilterEntry"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="ipv4FilterSrcL4PortMax">
    <rdfs:range>
      <OT_Attribute rdf:ID="_ipv4FilterSrcL4PortMax">
        <rdfs:subClassOf rdf:resource="#ipv4FilterEntryAttributes"/>
          <OT_Description rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
            The maximum value that the packet's layer 4 source port number
            can have and match this filter.
          </OT_Description>
          <OT_Syntax rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
            Integer32 (0..65535)
          </OT_Syntax>
          <OT_Status rdf:resource="#current"/>
          <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.1.1.2.11</OID>
          <rdfs:subClassOf>
            <owl:Restriction>
              <owl:onProperty> <owl:FunctionalProperty rdf:about="#VALUE"/> </owl:onProperty>
              <owl:allValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
            </owl:Restriction>
          </rdfs:subClassOf>
      </OT_Attribute>
    </rdfs:range>
```

```
  <rdfs:domain rdf:resource="#ipv4FilterEntry"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="ipv4FilterPermit">
  <rdfs:range>
    <OT_Attribute rdf:ID="_ipv4FilterPermit">
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:allValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
          <owl:onProperty> <owl:FunctionalProperty rdf:about="#VALUE"/> </owl:onProperty>
        </owl:Restriction>
      </rdfs:subClassOf>
      <OT_Syntax rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        TruthValue
      </OT_Syntax>
      <rdfs:subClassOf rdf:resource="#ipv4FilterEntryAttributes"/>
      <OT_Description rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
        If false, the evaluation is negated.
        That is, a valid match will be evaluated as not a match and vice versa.
      </OT_Description>
      <OT_Status rdf:resource="#current"/>
      <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.1.1.2.7</OID>
    </OT_Attribute>
  </rdfs:range>
  <rdfs:domain rdf:resource="#ipv4FilterEntry"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="ipv4FilterDstAddrMask">
  <rdfs:domain rdf:resource="#ipv4FilterEntry"/>
  <rdfs:range>
  <OT_Attribute rdf:ID="_ipv4FilterDstAddrMask">
    <OT_Status rdf:resource="#current"/>
    <OT_Description rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
      A mask for the matching of the destination IP address.A zero bit
      in the mask means that the corresponding bit in the address always matches.
    </OT_Description>
    <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.1.1.2.3</OID>
    <rdfs:subClassOf rdf:resource="#ipv4FilterEntryAttributes"/>
    <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty> <owl:FunctionalProperty rdf:about="#VALUE"/> </owl:onProperty>
          <owl:allValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
        </owl:Restriction>
    </rdfs:subClassOf>
      <OT_Syntax rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        IpAddress
      </OT_Syntax>
    </OT_Attribute>
  </rdfs:range>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="ipv4FilterDstL4PortMax">
  <rdfs:domain rdf:resource="#ipv4FilterEntry"/>
  <rdfs:range>
  <OT_Attribute rdf:ID="_ipv4FilterDstL4PortMax">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
```

```
          <owl:onProperty> <owl:FunctionalProperty rdf:about="#VALUE"/> </owl:onProperty>
        </owl:Restriction>
      </rdfs:subClassOf>
      <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.1.1.2.4</OID>
      <OT_Status rdf:resource="#current"/>
      <OT_Syntax rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
        Integer32 (0..65535)
      </OT_Syntax>
      <OT_Description rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
        The maximum value that the packet's layer 4 destination port number can have
        and match this filter.
      </OT_Description>
      <rdfs:subClassOf rdf:resource="#ipv4FilterEntryAttributes"/>
    </OT_Attribute>
    </rdfs:range>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:ObjectProperty>
<owl:FunctionalProperty rdf:ID="ipv4FilterSrcAddrMask">
    <rdfs:domain rdf:resource="#ipv4FilterEntry"/>
    <rdfs:range>
      <OT_Attribute rdf:ID="_ipv4FilterSrcAddrMask">
        <OT_Description rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
          A mask for the matching of the source IP address.
        </OT_Description>
        <OT_Syntax rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
          IpAddress
        </OT_Syntax>
        <rdfs:subClassOf rdf:resource="#ipv4FilterEntryAttributes"/>
        <rdfs:subClassOf>
          <owl:Restriction>
            <owl:allValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
            <owl:onProperty> <owl:FunctionalProperty rdf:about="#VALUE"/> </owl:onProperty>
          </owl:Restriction>
        </rdfs:subClassOf>
        <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.1.1.2.10</OID>
        <OT_Status rdf:resource="#current"/>
      </OT_Attribute>
    </rdfs:range>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="ipv4FilterProtocol">
    <rdfs:domain rdf:resource="#ipv4FilterEntry"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:range>
      <OT_Attribute rdf:ID="_ipv4FilterProtocol">
        <OT_Syntax rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
          Integer32 (0..255)
        </OT_Syntax>
        <OT_Description rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
          The IP protocol to match against the packet's protocol. A value of zero means match all.
        </OT_Description>
        <rdfs:subClassOf rdf:resource="#ipv4FilterEntryAttributes"/>
        <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.1.1.2.8</OID>
        <OT_Status rdf:resource="#current"/>
        <rdfs:subClassOf>
          <owl:Restriction>
            <owl:allValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
```

```
            <owl:onProperty> <owl:FunctionalProperty rdf:about="#VALUE"/> </owl:onProperty>
          </owl:Restriction>
        </rdfs:subClassOf>
      </OT_Attribute>
    </rdfs:range>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="ipv4FilterSrcAddr">
    <rdfs:domain rdf:resource="#ipv4FilterEntry"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:range>
      <OT_Attribute rdf:ID="_ipv4FilterSrcAddr">
        <OT_Status rdf:resource="#current"/>
        <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.1.1.2.9</OID>
        <OT_Description rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
          The IP address to match against the packet's source IP address.
        </OT_Description>
        <OT_Syntax rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
          IpAddress
        </OT_Syntax>
        <rdfs:subClassOf>
          <owl:Restriction>
            <owl:onProperty> <owl:FunctionalProperty rdf:about="#VALUE"/> </owl:onProperty>
            <owl:allValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
          </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf rdf:resource="#ipv4FilterEntryAttributes"/>
      </OT_Attribute>
    </rdfs:range>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="ipv4FilterDstL4PortMin">
    <rdfs:domain rdf:resource="#ipv4FilterEntry"/>
    <rdfs:range>
      <OT_Attribute rdf:ID="_ipv4FilterDstL4PortMin">
        <rdfs:subClassOf rdf:resource="#ipv4FilterEntryAttributes"/>
        <OT_Status rdf:resource="#current"/>
        <rdfs:subClassOf>
          <owl:Restriction>
            <owl:allValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
            <owl:onProperty> <owl:FunctionalProperty rdf:about="#VALUE"/> </owl:onProperty>
          </owl:Restriction>
        </rdfs:subClassOf>
        <OT_Syntax rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
          Integer32 (0..65535)
        </OT_Syntax>
        <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.1.1.2.5</OID>
        <OT_Description rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
          The minimum value that the packet's layer 4 destination port number can have
          and match this filter.
        </OT_Description>
      </OT_Attribute>
    </rdfs:range>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="ipv4FilterDscp">
    <rdfs:domain rdf:resource="#ipv4FilterEntry"/>
    <rdfs:range>
      <OT_Attribute rdf:ID="_ipv4FilterDscp">
```

```
        <OT_Status rdf:resource="#current"/>
        <rdfs:subClassOf rdf:resource="#ipv4FilterEntryAttributes"/>
        <OT_Description rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
          The value that the DSCP in the packet can have and match.
          A value of -1 indicates that a specific
          DSCP value has not been defined and thus all DSCP values are considered a match.
        </OT_Description>
        <OT_Syntax rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
          Integer32 (-1 | 0..63)
        </OT_Syntax>
        <OID rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.1.1.2.1</OID>
        <rdfs:subClassOf>
          <owl:Restriction>
            <owl:onProperty rdf:resource="#VALUE"/>
            <owl:allValuesFrom rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
          </owl:Restriction>
        </rdfs:subClassOf>
      </OT_Attribute>
    </rdfs:range>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>
```

Figure D.4 - Format OWL de la PIB de filtrage

REFERENCES

[1] Campbell, A.T, et al., “A Survey of Programmable Networks”, ACM SIGCOMM Computer Communication Review, Vol. 29 No. 2 pg. 7-24, April 1999.

[2] Tennenhouse, D. L., Smith, J. M., Sincoskie, W. D., Wetherall, D. J., Minden, G. J., “A Survey of Active Network Research”, IEEE Communications Magazine, Vol. 35, No. 1, Jan 1997, pp80-86.

[3] Calvert, K., (Editor), “Architectural Framework for Active Networks,” DARPA AN Working Group Draft, 1998.

[4] Kulkarni, A.B. Minden G.J., Hill, R., Wijata, Y., Gopinath, A., Sheth, S., Wahhab, F., Pindi, H., and Nagarajan, A., “Implementation of a Prototype Active Network”, First International Conference on Open Architectures and Network Programming (OPENARCH), San Francisco, 1998.

[5] Dan Decasper and Bernhard Plattner, DAN: Distributed Code Cashing for Active Networks, Proc. IEEE INFOCOM '98, San Francisco, CA, 29 March-2 April 1998.

[6] Wetherall, D., et al., “ANTS: A toolkit for building and dynamically deploying network protocols,” IEEE OPENARCH’98, San Francisco, CA, April 1998.

[7] Tim Stack, Eric Eide, Jay Lepreau, “Bees: A Secure, Resource-Controlled, Java-Based Execution Environment”, IEEE Conference on Open Architectures and Network Programming Proceedings (OPENARCH 2003), pages 97-106, San Francisco, CA, April 4-5, 2003.

[8] Alexander, D. S. et al., “The SwitchWare Active Network Architecture”, IEEE Network Special Issue on Active and Controllable Networks, May/June 1998, vol. 12 no. 3, pp. 29 – 36.

[9] Hicks, M., Kakkar , P., Moore , J. T., Gunter, C. A., Nettles, S., “PLAN: A Packet Language for Active Networks”, Proceedings of the International Conference on Functional Programming (ICFP) '98.

[10] Da Silva, S., Florissi, D. and Yemini, Y., “NetScript: A Language-Based Approach to Active Networks”, Technical Report, Computer Science Dept., Columbia University January 27, 1998.

[11] Delgrossi, L. and Ferrari D., “A Virtual Network Service for Integrated-Services Internetworks”, 7th International Workshop on Network and Operating System Support for Digital Audio and Video, St. Louis, May 1997.

[12] Schwartz, B., Jackson, W.A., Strayer W.T., Zhou, W., Rockwell, R.D., and Partridge, C., "Smart Packets for Active Networks”, Second International Conference on Open Architectures and Network Programming (OPENARCH), New York, 1999.

[13] Alexander, D. S., Braden, B., C. Gunter, A., Jackson, A. W., Keromytis, A. D., Minden, G.J., Wetherall, D., “Active Network Encapsulation Protocol (ANEP)”, Draft RFC, Active Networks Group, July 1997 Available at http://www.cis.upenn.edu/~switchware/ANEP/.

[14] K. Psounis, “Active Networks: Applications, Security, Safety, and Architectures” IEEE Communications survey, Vol. 2, No 1, 1st qtr. 1999.

[15] http://www.corba.org/

[16] David Acremann, Gilles Moujeard and Laurent Rousset, Certified Authors Sun & Inprise/VisiBroker, “Developping with Corba in Java and C++”, CampuPress 2nd Edition

[17] http://www.microsoft.com/com/tech/DCOM.asp

[18] http://java.sun.com/products/jdk/rmi/

[19] http://java.sun.com/products/rmi-iiop/index.html

[20] Gamma, E., Helm, R., Johnson, R., Vlissides, J., “Design Patterns : Elements of Reusable Object-Oriented Software” Addison-Wesley Professional Computing Series. Addison-Wesley, 1995

[21] Sun Microsystems Enterprise JavaBeans Specification, available at java.sun.com/products/ejb/docs.html

[22] www.microsoft.com/com/tech/MTS.asp

[23] Markup Language XML (W3C): http://www.w3.org/XML/

[24] XML Schema: http://www.w3.org/XML/Schema

[25] Web Services Activity: http://www.w3.org/2002/ws

[26] Simple Object Access Protocol: http://www.w3.org/TR/SOAP.

[27] WSDL: Web Services Description Language: www.w3.org/TR/wsdl

[28] UDDI: Universal Description, Discovery and Integration: http://www.uddi.org/
[29] M. Powell, WS-Security Authentication and Digital Signatures with Web Services Enhancements, December 2002.
[30] H. Nielsen, S. Thatte, Web Services Routing Protocol (WS-Routing), 23 October 2001.
[31] Galis, A., Plattner, B., Moeller, E., Laarhuis, J., Denazis, S., Guo, H., Klein, C., Serrat, J., Karetsos, G. T., Todd, C. "A Flexible IP Active Networks Architecture" In The Second International Conference on Active Networks (IWAN), edited by H. Yasuda, Volume 1942. IEEE, Springer Press, Tokjo Japan, October 2000.
[32] www.promethos.org/
[33] Matthias Bossardt, Roman Hoog Antink, Andreas Moser, and Bernhard Plattner: Chameleon: Realizing Automatic Service Composition for Extensible Active Routers. In Proceedings of Fifth Annual International Working Conference on Active Networks (IWAN 2003), Kyoto, Japan, Lecture Notes in Computer Science, Springer Verlag, Berlin Heidelberg New York, December, 2003.
[34] Cook, C., Pawlikowski, K., Sirisena, H., "ComAN: A Multiple-Language Active Network Architecture Enabled via Middleware", OpenArch'02, New York, NY, Jun 2002.
[35] Rima Kilany et Ahmed Serhrouchni, "Using Distributed Component Model for Active Service Deployment", ISCC 2002.
[36] Rima KILANY, Dany ZEBIANE, Michel RIGUIDEL, Ahmed SERHROUCHNI, "A Control Architecture for Active Networks", SoftCom2001, http://www.fesb.hr/SoftCOM/2001.
[37] http://www.aspencrypt.com/task_pkcs7.html
[38] S. J. Shepard, "Policy-based Networks: Hype and Hope," IT Prof., vol. 2, no. 1, Jan.-Feb. 2000, pp.12–16.
[39] "Introduction to Policy-Based Networking and Quality of Service," IPHighway, White Paper, Jan. 2000.
[40] M. Sloman, "Policy Driven Management For Distributed Systems," Int'l. J. Net. Sys. Mgmt., vol. 2, no. 4, Dec. 1994, pp. 333-360.
[41] A. Westerinen et al., "Terminology," IETF, Internet draft, draft-ietf-policy-terminology-02.txt, Nov. 2000; http://www.ietf.org/internet-drafts/draft-ietfpolicy-terminology-02.txt
[42] "Resource Allocation Protocol (rap)," http://www.ietf.org/html.charters/rap-charter.html
[43] Boyle, J., R. Cohen, D. Durham, S. Herzog, R. Rajan, and A. Sastry: The COPS (Common Open Policy Service) Protocol, RFC 2748, January 2000.
[44] Chan, K., J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, A. Smith: COPS Usage for Policy Provisionning (COPS-PR), RFC 3084, January 2000.
[45] Fine, M., K. McCloghrie, J. Seligson, K. Chan, S. Hahn, R. Sahita, A. Smith, F. Reichmeyer,: Framework Policy Information Base, draft-ietf-framework-pib-07.txt, January 2002.
[46] McCloghrie, K., M. Fine, J. Seligson, K. Chan, S. Hahn, R. Sahita, A. Smith, F. Reichmeyer: Structure of Policy Provisioning Information (SPPI), RFC3159, August 2001.
[47] Andreas Polyrakis, Raouf Boutaba, The Meta-Policy Information Base, IEEE Network Magazine, Special issue on Policy Based Networking, Vol. 16, No. 2, March/April 2002, pp 40-48.
[48] "Policy Based Networking Products, Design and Architecture," IPHighway, White Paper, Jan. 2000.
[49] "COPS Download Page," http://www.vovida.org/protocols/downloads/cops
[50] M. Stevens, et al., 'Policy Framework', Internet Draft, March 2000
[51] R. Yavatkar, et al, 'A Framework for Policy-based Admission Control', RFC 2753, January 2000
[52] E. Lupu, M. Sloman, 'Conflict Analysis for Management Policies', Proceedings of the 5th International Symposium on Integrated Network Management IM'97, San Diego, Chapman & Hall, May 1997.
[53] K. Kato, S. Shiba, "Designing Policy Networking System Using Active Networks", Second International Working Conference on Active Networks (IWAN'2000), Tokyo, Japan, October 2000, Springer-Verlag LNCS, Yasuda (ed.).
[54] Seraphim Project homepage, "Seraphim: Building Dynamic Interoperable Security Architecture For Active Networks", http://choices.cs.uiuc.edu/Security/seraphim/.
[55] Active Network Distributed Open Infrastructure Development (ANDROID) - http://www.cs.ucl.ac.uk/research/android/.
[56] Y. Kanada, "Dynamically extensible policy server and agent", Policies for Distributed Systems

and Networks, 2002. Proceedings, pages 236-239.
[57] M. Fonseca, N. Agoulmine, O. Cherkaoui, “Active Networks as a Flexible Approach to deploy QoS Policy Based Management”, HP Openview University Association 8th Annual Workshop, HP-OVUA, Berlin, Germany, June 24 – 27, 2001, http://citeseer.nj.nec.com/483138.html.
[58] Polynet, “PolyNet: Policy Based Management of Adaptive Networks” - http://www.doc.ic.ac.uk/~mss/polynet.html.
[59] M. Sloman, and E. Lupu, “Policy Specification for Programmable Networks”, International Working Conference on Active Networks (IWAN'99), Berlin, Germany, June-July 1999, Springer-Verlag LNCS, Stefan Covaci (ed.).
[60] “Future Active IP Networks (FAIN)” FAIN homepage http://www.ist-fain.org FAIN Deliverable D1, “Requirements Analysis and Overall AN Architecture”, FAIN, May 2001 – http://www.ist-fain.org.
[61] FAIN Deliverable D40, “FAIN Demonstrators”, FAIN, May 2003 – http://www.ist-fain.org.
[62] Tsarouchis C., Kitahara C., Denazis S., Julio Vivero J., et.al , "Policy-Based Management Architecture for Active and Programmable Networks", IEEE Network, May 2003, Vol.17, No.3.
[63] FAIN Deliverable D8, “Final Specification of Case Study Systems”, FAIN, May 2003 – http://www.ist-fain.org.
[64] FAIN Deliverable D7, “Final Active Node Architecture and Design”, FAIN, May 2003 – http://www.ist-fain.org.
[65] Alex Galis, Alvin Tan, Joan Serrat, Yiannis Nikolakis, Julio Vivero, Spyros Denazis, Juan Luis Manas, Jan H Laarhuis, ‘Policy-based Network Management for Active Networks’, IEEE ICT 2001 Conference proceedings, Bucharest, Romania, 4-7 June 2001
[66] Raouf Boutaba, Andreas Polyrakis, Alvaro Fernandez Casani, “Active Networks as a Developing and Testing Environment for Network Protocols”, icoin’2003
[67] http://www.itu.int/TMN/
[68] “An Overview of the Oplet Runtime Environment (ORE)” www.openetlab.com/ore.latest/doc/ore/overview.html
[69] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. “Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification”, RFC 2205, Network Working Group, IETF,September 1997. (www.ietf.org)
[70] http://www.w3.org/RDF/
[71] http://www.w3.org/TR/2004/REC-rdf-schema-20040210/
[72] DAML Program, The DARPA Agent Markup Language Homepage, May 2002. Available at http://www.daml.org/.
[73] Dan Connolly, Frank van Harmelen, Ian Horrocks, Deborah L. McGuiness, Peter F. Patel-Schneider, Lynn Andrea Stein, “DAML+OIL (March 2001) reference Description”, W3C Notes, 18 December 2001.
[74] Michael K. Smith, Chris Welty, Deborah L. McGuinness, “OWL Web Ontology Language Guide,” W3C Candidate Recommendation, 10 February 2004.
[75] Distributed Management Task Force, Inc., Common Information Model Specification version 2.2, DMTF Document. June 1999.
[76] Asunción Gómez Pérez, V. Richard Benjamins, “Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem-Solving Methods”, Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods (KRR5), Stockholm, Sweden. August 1999.
[77] Dominique Benech, Interaction Frameworks for Distributed and Cooperative Paradigms of Intelligent Systems and Networks Management. Ph. D. Thesis. Université Paul Sabatier de Toulouse III, France. November 1999.
[78] Jorge E. López de Vergara, Víctor A. Villagrá, Julio Berrocal, Applying the Web Ontology Language to management information definitions, accepted for its publication in IEEE Communications Magazine, special issue on XML Management, Vol. 42, Issue 7, July 2004.
[79] N. Damianou, N. Dulay, E. Lupu, M Sloman, The Ponder Specification Language, Workshop on Policies for Distributed Systems and Networks (Policy2001), HP Labs Bristol, 29-31 Jan 2001.
[80] R. Studer, V.R. Benjamins, and D. Fensel, “Knowledge Engineering: Principles and Methods”, Data & Knowledge Engineering. 25: 161-197. 1998.
[81] Asunción Gómez-Pérez, Oscar Corcho, “Ontology Languages for the Semantic Web”, IEEE Intelligent Systems, Vol. 17, Issue 1, January/February 2002.

[82] Natalya Fridman Noy, Mark A. Musen, "An Algorithm for Merging and Aligning Ontologies: Automation and Tool Support", Proceedings of the workshop on Ontology Management, Sixteenth National Conference On Artificial Intelligence (AAAI-99), Orlando, Florida, U.S.A., July 1999.
[83] Deborah L. McGuinness, Richard Fikes, James Rice, Steve Wilder, "An Environment for Merging and Testing Large Ontologies", Proceedings of the Seventh National Conference On Principles of Knowledge Representation and Reasoning (KR2000), Breckenridge, Colorado, U.S.A., April 2000.
[84] Gerd Stumme, Alexander Maedche, "FCA-MERGE: Bottom-Up Merging of Ontologies", Proceedings of the Seventeenth International Joint Conference On Artificial Intelligence (IJCAI 2001), Seattle, Washington, U.S.A., August 2001.
[85] E-Commerce Integration Meta-Framework (ECIMF) Project, "ECIMF Semantic Translation Tool", November 2001, available at http://www.ecimf.org/software.html.
[86] Alexander Maedche, Boris Motik, Nuno Silva, Raphael Volz, "MAFRA – A Mapping FRAmework For Distributed Ontologies", Proceedings of the Thirteenth Europian Conference On Knowledge Engineering and Knowledge Management (EKAW'02), Madrid, Spain, October 2002.
[87] Natalya F. Noy, Michael Sintek, Stefan Decker, Monica Crubézy, Ray W. Fergerson, Mark A. Musen, "Creating Semantic Web Contents with Protégé-2000", IEEE Intelligent Systems, Vol. 16, Issue 2, March/April 2001.
[88] Jun Shen, Yun Yang: RDF-Based Knowledge Models for Network Management. In: Proceedings of the Eighth IFIP/IEEE International Symposium on Integrated Network Management (IM'2003), Colorado Springs, Colorado, U.S.A. (March 2003).
[89] Emmanuel Lavinal, Thierry Desprats, Yves Raynaud: A Conceptual Framework for Building CIM-Based Ontologies. In: Proceedings of the Eighth IFIP/IEEE International Symposium on Integrated Network Management (IM'2003), Colorado Springs, Colorado, U.S.A. (March 2003).
[90] Bradshaw, J. M., Dutfield, S., Benoit, P., & Woolley, J. D. (1997). KAoS: Toward an industrial-strength generic agent architecture. In J. M. Bradshaw (Ed.), Software Agents. (pp. 375-418). Cambridge, MA: AAAI Press/The MIT Press.
[91] Suri, N., Bradshaw, J. M., Breedy, M. R., Groth, P. T., Hill, G. A., Jeffers, R., Mitrovich, T. R., Pouliot, B. R., & Smith, D. S. (2000). NOMADS: Toward an environment for strong and safe agent mobility. Proceedings of Autonomous Agents 2000. Barcelona, Spain, New York: ACM Press.
[92] Kahn, M., & Cicalese, C. (2001). CoABS Grid Scalability Experiments. O. F. Rana (Ed.), Second International Workshop on Infrastructure for Scalable Multi-Agent Systems at the Fifth International Conference on Autonomous Agents. Montreal, CA, New York: ACM Press.
[93] Greaves, M., Holmback, H., & Bradshaw, J. M. (2001). Agent conversation policies. In J. M. Bradshaw (Ed.), Handbook of Agent Technology. (pp. in preparation). Cambridge, MA: AAAI Press/The MIT Press.
[94] http://www.cougaar.net
[95] http://www.recursionsw.com/osi.asp
[96] http://www.java-agent.org
[97] http://java.sun.com/security/
[98] http://www.ksl.stanford.edu/software/JTP/
[99] Knoll, G., Suri, N., & Bradshaw, J. M. (2001). Path-based security for mobile agents. Proceedings of the First International Workshop on the Security of Mobile Multi-Agent Systems (SEMAS-2001) at the Fifth International Conference on Autonomous Agents (Agents 2001), (pp. 54-60). Montreal, CA, New York: ACM Press.
[100] Lupu, E. C., & Sloman, M. S. (1999). Conflicts in policy-based distributed systems management. IEEE Transactions on Software Engineering—Special Issue on Inconsistency Management.
[101] J. Kornblum, D. Raz and Y. Shavitt. The active process interaction with its environment. In IWAN 2000, October 2000.
[102] V. Galtier, K. Mills, Y. Carlinet, S. Bush, A. Kulkarni. Prediction and controlling resource usage in a heterogeneous Active Network. In MILCOM 2001, October 2001.
[103] Ian Horrocks, Peter F.P. atel-Schneider, A proposal for an OWL Rules Language, WWW2004,

May 17-22 2004.
[104] HP Labs Semantic Web, API Jena, http://www-uk.hpl.hp.com/people/bwm/rdf/jena.
[105] BISWAS, J., et al., “The IEEE P1520 Standards Initiative for Programmable Network Interfaces”, IEEE Communications Magazine, Special Issue on Programmable Networks, October 1998.
[106] A.D. Keromytis, “draft-keromytis-disfi-compare-00”.
[107] Huang H., "Active Networks: An overview".
[108] Albert Banchs et al., Multicasting Multimedia Streams with Active Networks, ICSI technical report 97-050.
[109] David J. Wetherall and David L. Tennenhouse, The ACTIVE_IP option, In the 7th ACM SIGOPS European Workshop.
[110] Microsoft .NET: http://www.microsoft.com/net/basics.mspx.
[111] Rima Kilany, Michel Riguidel, Ahmed Serhrouchni, Dany Zebiane, “Control Architecture for ANTS”, IFIP Workshop on IP and ATM Traffic Management WATM’2001 And EUNICE’2001.
[112] Web Services Referral Protocol (WS-Referral), H. Nielsen, S. Thatte, 23 October 2001.
[113] http://www.mono-project.com/Main_Page
[114] Microsoft ASP.NET: http://www.asp.net/Default.aspx
[115] Microsoft IIS: http://www.microsoft.com/windowsserver2003/iis/default.mspx
[116] Tim Ewald, Keith Brown, “HTTP Pipelines: Securely Implement Request Processing, Filtering, and Content Redirection with HTTP Pipelines in ASP.NET”, http://msdn.microsoft.com/msdnmag/issues/02/09/HTTPPipelines/default.aspx.
[117] David Booth, Hugo Haas, “Web Services Architecture”, section 3.1, W3C Working Group Note 11 February 2004, http://www.w3.org/TR/ws-arch/.
[118] Matt Powell, “Programming with Web Services Enhancements 2.0”, MSDN Web Services, May 2004.
[119] Draft ISO/IEC 9594-8, “ITU-T Recommendation X.509”, May 3, 2001.
[120] R. Srinivasan, “RPC: Remote Procedure Call Protocol Specification Version 2”, RFC 1831, Sun Microsystems, August 1995.
[121] Object Management Group (OMG), “Common Object Request Broker Architecture: Core Specification”, Version 3.0.3, March 2004.
[122] Object Management Group (OMG), “Naming Service Specification”, Version 1.3, October 2004.
[123] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog,A. Huynh, M. Carlson, J., “Terminology for Policy-Based Management”, RFC 3198, November 2001.
[124] D. Harrington, R. Presuhn, B. Wijnen, “An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks”, STD 62, RFC 3411, December 2002.
[125] R. Presuhn, J. Case, K. McCloghrie, M. Rose, S. Waldbusser, “Management Information Base for the Simple Network Management Protocol (SNMP)”, STD 62, RFC 3418, December 2002.
[126] Internet Engineering Task Force (IETF), http://www.ietf.org/
[127] Wahl M., Howes T., Kille S., “Lightweight Directory Access Protocol (v3)”, RFC 2251, December 1997.
[128] Herzog S ., Boyle J., Cohen R., Durham D., Rajan R., Sastry A.: COPS usage for RSVP (COPS-RSVP), RFC 2749, January 2000.
[129] B. Moore, E. Ellesson, J. Strassner, A. Westerinen, “Policy Core Information Model - Version 1 Specification”, RFC 3060, February 2001.
[130] J. Nicklisch, “A rule language for network policies”, C and C Network Product Development Laboratories.
[131] J. Postel, J. Reynolds, “FILE TRANSFER PROTOCOL (FTP)”, RFC 959, October 1985.
[132] J. Klensin, “Simple Mail Transfer Protocol (SMTP)”, RFC 2821, April 2001.
[133] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1”, RFC 2616, June 1999.
[134] T. Berners-Lee, D. Connolly, “Hypertext Markup Language (HTML)- 2.0”, RFC 1866, November 1995.
[135] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, “An Architecture for Differentiated Services”, RFC 2474, December 1998.
[136] S. Kent, R. Atkinson, “Security Architecture for the Internet Protocol”, RFC 2401, November

1998.
[137] ISO 8825 Information Processing Systems - Open Systems Interconnection – “Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)”, 1987.
[138] M. Rose, K. McCloghrie, “Structure and identification of management information for TCP/IP-based internets”, RFC 1155, May 1990.
[139] DMTF: http://www.dmtf.org/home.
[140] Berners-Lee T., Hendler J., Lassila O., “The Semantic Web”, Scientific American, May 2001.
[141] Berners-Lee T., "Universal Resource Identifiers in WWW," RFC 1630, June 1994.
[142] Natalya F. Noy, Michael Sintek, Stefan Decker, Monica Crubézy, Ray W. Fergerson, Mark A. Musen, “Creating Semantic Web Contents with Protégé-2000”, IEEE Intelligent Systems, Vol. 16, Issue 2, March/April 2001.
[143] Horrocks I., Patel-Scheider P.F., Boley H., Tabet S., Grosof B., Dean M., “SWRL: A semantic Web Rule Language Combining OWL and RuleML”, W3C Member Submission 21 May 2004.
[144] T. Berners-Lee, R. Cailliau, J.F. Groff, and B. Pollermann. World-Wide Web: The information universe. Electronic Networking: Research, Applications and Policy, 2(1):52–58, 1992. Meckler Publishing, CT, USA.
[145] Site Web du W3C : http://www.w3.org/
[146] RQL: The RDF query language. Site Web: http://139.91.183.30:9090/RDF/RQL/
[147] A. Le Hors, P. Le Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, S. Byrne, “Document Object Model (DOM) Level 2 Core Specification – version 1.0“, W3C Recommendation, 13 November, 2000.
[148] K. Chan, R. Sahita, S. Hahn, K. McCloghrie, “Differentiated Services Quality of Service Policy Information Base”, RFC 3317, March 2003.
[149] PktFilter, http://www.hsc.fr/outils/pktfilter/
[150] Windows driver development kit; http://www.microsoft.com/whdc/ddk/winddk.mspx
[151] Ivo Ivanov, “API hoking revealed”, http://codeguru.earthweb.com/system/apihook.html
[152] VicSoft.RDF : http://www.schemaweb.info/parser/Parser.aspx.
[153] Internet Protocol Version 4 (IPv4) Specification. Internet Standard RFC 791.
[154] Internet Protocol Version 6 (IPv6) Specification. Internet Standard RFC 2460.
[155] OPENSIG: Open Signalling and Service Creation. Disponible sur : http://comet.columbia.edu/opensig/
[156] PETERSON Larry. Node OS and Interface Specification. January 2000. Disponible sur : http://www.cs.princeton.edu/nsg/papers/nodeos.ps
[157] YEMINI Y. Active Network Management. Disponible sur : http://www.cs.columbia.edu/dcc/anm
[158] Hoa-Binh Nguyen, « Services Actifs et Passerelles Programmables », thèse, 16 janvier 2004.
[159] Netfilter, http://www.netfilter.org
[160] AMIR E., McCANNE S., KATZ R. An Active Service Framework and its Application to Real-time Multimedia Transcoding. ACM Communication Review, vol. 28, no. 4, pp. 178-189, Sep. 1998.
[161] Tcl (Tool Command Language). Disponible sur :http ://www.tcl.tk/software/tcltk/
[162] Distributed Management Task Force, inc., “Web Based Enterprise Management”, DMTF WBEM Description presentation. June 2000.
[163] S. M. Klerer, “The OSI Management Architecture: An overview”, IEEE Network Magazine, pp. 20-29, March 1988.
[164] Jorge E. López de Vergara, Víctor A. Villagrá, Julio Berrocal, “An ontology-based method to merge and map management information models”, in Proc. HP Openview University Association 10th Plenary Workshop, Geneva, Switzerland, July 2003.
[165] Bradshaw, J. M., Uszok, A., Jeffers, R., Suri, N., Hayes, P., Burstein, M. H., Acquisti, A., Benyo, B., Breedy, M. R., Carvalho, M., Diller, D., Johnson, M., Kulkarni, S., Lott, J., Sierhuis, M., & Van Hoof, R. (2003). “Representation and reasoning for DAML-based policy and domain services in KAoS and Nomads”. Proceedings of the Autonomous Agents and Multi-Agent Systems Conference (AAMAS 2003). Melbourne, Australia, New York, NY: ACM Press.
[166] R. France, « AMARRAGE Project », http://www.telecom.gouv.fr/rnrt/projets/res_d41_ap99.htm, septembre 2002.
[167] R. France, « AMARILLO Project », http://www.telecom.gouv.fr/rnrt/projets/res_02_57.htm, mai 2002

[168] Rima Kilany, «Conception, implémentation et simulation d'un réseau actif ouvert et contrôlable», thèse, 13 juin 2003.
[169] Michael Schroeder and Gerd Wagner (Eds.), “Proceedings of the International Workshop on Rule Markup Languages for Business Rules on the Semantic Web”. Sardinia, Italy, June 14, 2002. CEUR-WS Publication Vol-60.
[170] Alon Y. Levy and Marie-Christine Rousset, “The limits on combining recursive horn rules and description logics”, Proceedings of the AAAI Thirteenth National Conference on Artificial Intelligence, 1996.

PUBLICATIONS

[1] Maroun Chamoun, Rima Kilany, Ahmed Serhrouchni, "Web Sémantique et Gestion par Politiques", Notere 2005, Gatineau, Canada, 29 Août au 1er Septembre 2005.

[2] Maroun Chamoun, Rima Kilany, Ahmed Serhrouchni, "A Semantic Active Policy-based Management Architecture", 2004 IEEE International Workshop on IP operations and management (IPOM 2004), October 11-13, 2004, Beijing, China.

[3] Maroun Chamoun, Rima Kilany, Ahmed Serhrouchni, "Proposition of Ontology for Network Policy Management", The 4th International Symposium on SIGNAL PROCESSING AND INFORMATION TECHNOLOGY, IEEE ISSPIT 2004, December 18-21, 2004 Rome, Italy.

[4] Maroun Chamoun, Rima Kilany, Ahmed Serhrouchni, "Policy-based Management using Web Services based Active network Architecture", SoftCom 2004, Split (Croatia), Dubrovnik (Croatia), Venice (Italy) October 10-13, 2004.

[5] Maroun Chamoun, Rima Kilany, Ahmed Serhrouchni, "Filtrage distribué à l'aide d'une Architecture d'un Réseau Actif à base de Services Web", Notere 2004, Saidia, Maroc, 27-30 Juin 2004.

[6] Maroun Chamoun, Rima Kilany, Ahmed Serhrouchni, "Web Services based Active network Architecture", Annales des télécommunications, Vol. 59, n° 5-6, mai-juin 2004, Réseaux actifs: architecture et applications reconfigurables.

GLOSSAIRE

AA	Applications Actives
ACLs	Access Control Lists
AN	Active Network
ANEP	Active Network Encapsulation Protocol
ANTS	Active Network Transfert System
API	Application Programming Interface
AS1	Active Service version 1
ASP	Active Service Provisioning
ASWA	Architecture à base de Services Web Actifs
BER	Basic Encoding Rules
CIM	Common Information Model
CMIP	Common Management Information Protocol
COM	Component Object Model
COMAN	Component Object Model Active Network
COPS	Common Object Policy Service
COPS-PR	Common Object Policy Service - PRovisioning
CORBA	Common Object Request Broker Architecture
CPERR	PRC Class Provisioning Error Object
DAML	DARPA Agent Markup Language
DAML+OIL	DARPA Agent Markup Language plus Ontology Inference Layer
DAN	Distributed Code Cashing for Active Networks
DARPA	Defence Advanced Research Projects Agency
DCOM	Distributed Component Object Model
DiffServ	Differentiated Services
DLL	Dynamic Link Library
DMIB	Dynamic Management Information Base
DMTF	Distributed Management Task Force
DOM	Document Object Model
EE	Environnement d'Exécution
EJB	Entreprise Java Beans
EMS	Element Management System
EPD	Encoding Provisioning Instance Data
FAIN	Future Active IP Networks
FTP	File Transfer Protocol
GDMO	Guidelines for Definition of Managed Objects
GPERR	Global Provisioning Error Object
HABA	Hyper Active Beans component Architecture
HTML	Hypertext Markup Language

HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol over SSL
IETF	Internet Engineering Task Force
IPSec	Internet Protocol Security
KAoS	Knowledgeable Agent-oriented System
LDAP	Lightweight Directory Access Protocol
MIB	Management Information Base
MOF	Managed Object Format
NE	Network Element
NMS	Network Management System
OID	Object Identifier
OPENSIG	Open Signalling and Service Creation
ORB	Object Request Broker
OWL	Ontology Web Language
PBANEM	Policy-Based Active Network Element Management
PBM	Policy-Based Management
PBNM	Policy-Based Network Management
PCIM	Policy Core Information Model
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PFDL	Policy Framework Definition Language
PIB	Policy Information Base
PLAN	Packet Language for Active Networks
PPRID	Prefix Provisioning Instance Identifier
PRC	Provisioning Class
PRI	Provisioning Instance
PRID	Provisioning Instance Identifier
QoS	Quality of Service
RA	Réseau Actif
RAP	Ressource Allocation Protocol
RDF	Resource Description Framework
RDF-S	Resource Description Framework - Schema
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RSVP	Resource ReSerVation Protocol
SANE	Secure Active Network Environment
SLA	Service Level Agreement
SLS	Service Level Specification
SMI	Structure of Management Information
SMTP	Simple Message Transport Protocol
SNMP	Simple Network Management Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol

SSL	Secure Socket Layer
SWRL	Semantic Web Rule Language
URI	Uniform Resource Identifier
URL	Unique Reference Locator
UDDI	Universal Discovery Description and Integration
VE	Virtual Environment
WBEM	Web Based Enterprise Management
WSE	Web Service Enhancements
XML	Extensible Markup Language