Rapport de stage

DEA Réseaux

Année 2004, Option Multimédia et Qualité de Service

Martin Ludovic

Tuteurs Arnaud PIERRE
Vania CONAN

UNIVERSITE PIERRE & MARIE CURIE
LA SCIENCE A PARIS

LIP6

THALES

REMERCIEMENTS

Pour commencer, je remercie tous les membres du laboratoire TAI (Technologies Avancées de l'Information) et en particulier André C., son directeur, pour m'avoir chaleureusement accueilli parmi eux.

Je remercie également chaleureusement Arnaud PIERRE pour sa disponibilité, son écoute et ses conseils. Ses compétences professionnelles et ses qualités humaines ont précieusement contribué au bon déroulement de ce stage qui me fut extrêmement profitable.

Je tiens à remercier tout particulièrement Hervé A. et Corinne S. pour m'avoir permis de mieux comprendre la gestion de réseau par politiques.

Merci encore à Frank W. et à Jean-Etienne G. pour leur précieuse collaboration tout au long de mes travaux de développement.

Je remercie enfin mon amie Frédérique pour la relecture de ce rapport.

TABLE DES MATIERES

Remerciements 3

Table des matières 5

Table des figures 9

Introduction 11

Chapitre 1 : Le groupe Thales 12

1 Le groupe dans son ensemble 12
1.1 Activités 12
1.2 Stratégie 12
1.3 Structure 13
1.3.1 Le pôle Défense 14
1.3.2 Le pôle Aéronautique 14
1.3.3 Le pôle Technologie de l'Information & des Services 14
1.4 Thales et la recherche 15
2 Thales Land & Joint Systems 15
2.1 Activités 16
2.2 Rôle 16
3 Le service TAI 17
3.1 Organisation 17
3.2 Rôle 18

Chapitre 2 : La gestion par politiques 19

1 Introduction au PBNM (Policy-Based Network Management) 19
1.1 Les politiques et les règles 20
1.2 Les objectifs et contraintes 20
1.3 Quelques scénarios d'utilisation 21
2 L'architecture 22
2.1 Modèle trois-tiers 23
2.1.1 Console de gestion 23
2.1.2 Le PDP (Policy Decision Point) 24
2.1.2.1 L'organe décisionnel de l'architecture 24
2.1.2.2 Les autres serveurs 24
2.1.2.3 Le LPDP (Local Policy Decision Point) 25
2.1.3 Les PEPs (Policy Enforcement Point) 26
2.2 Autres modèles 27

2.2.1 Le modèle deux-tiers ... 27
2.2.2 Modèle hybride ... 28
2.2.3 Modèle évolué ... 28
3 Le protocole COPS (Common Open Policy Service) ... 30
3.1 Les caractéristiques principales ... 30
3.1.1 Les objectifs de COPS ... 30
3.1.2 Les extensions de COPS ... 31
3.1.2.1 Les modèles de gestion par politiques ... 31
3.1.2.1.1 Outsourcing ... 31
3.1.2.1.2 Provisioning ... 31
3.1.2.2 Les extensions ... 31
3.1.2.2.1 COPS-RSVP (COPS for RSVP) ... 32
3.1.2.2.2 COPS-PR (COPS for PRovisioning) ... 32
3.2 Les messages ... 32
3.2.1 Les objets ... 32
3.2.1.1 Entête ... 33
3.2.1.2 Types d'objets ... 33
3.2.2 Les messages ... 34
3.2.2.1 Entête ... 34
3.2.2.2 Types de messages ... 35
3.2.2.2.1 Gestion des états ... 35
3.2.2.2.2 Synchronisation des états ... 35
3.2.2.2.3 Gestion de la session ... 35
3.2.2.3 Relations entre les messages et les objets ... 36
3.3 Le déroulement d'une session ... 36
3.3.1 L'initialisation ... 36
3.3.1.1 Ouverture d'un canal client/serveur ... 37
3.3.1.2 Sécurité ... 38
3.3.2 Le fonctionnement normal ... 40
3.3.2.1 La configuration du réseau ... 40
3.3.2.1.1 Création d'un état ... 40
3.3.2.1.2 Modification d'un état ... 42
3.3.2.1.3 Suppression d'un état ... 44
3.3.2.2 Les opérations de maintenance ... 45
3.3.2.2.1 Les rapports ... 45
3.3.2.2.2 Les synchronisations ... 45
3.3.3 Les pannes ... 45
3.3.3.1 Détection ... 46
3.3.3.2 Fonctionnement temporaire ... 46
3.3.3.3 Retour à la normale ... 47
3.3.4 La fermeture ... 47
4 La représentation et le stockage des politiques ... 48
4.1 PCIM (Policy Core Information Model) ... 48
4.1.1 L'origine et les extensions de PCIM ... 48
4.1.2 Un langage plutôt déclaratif ... 49
4.1.2.1 Définitions ... 49
4.1.2.2 Le choix de PCIM ... 49
4.1.3 Un modèle orienté objet ... 49
4.2 Le Policy Repository ... 50
4.2.1 Les contraintes ... 51
4.2.2 Les solutions ... 52
4.2.2.1 Les bases de données relationnelles ... 52
4.2.2.2 Les annuaires ... 53
4.2.2.2.1 LDAP (Lightweight Directory Access Protocol) ... 53
4.2.2.2.2 Avantages et inconvénients ... 55
4.2.2.3 Conclusion ... 56
Chapitre 3 : Le stage ... 57
1 Le projet RHODOS ... 57

1.1 Description du projet ... 57
1.1.1 Objectifs ... 57
1.1.2 Contexte scientifique ... 58
1.1.2.1 Historique de la recherche ... 58
1.1.2.2 Positionnement du projet RHODOS ... 59
1.2 Organisation ... 59
1.2.1 Répartition du projet ... 59
1.2.2 Le rôle de Thales dans RHODOS ... 60
1.2.3 Mon rôle ... 61
2 L'architecture PDP/PEP existante ... 62
2.1 Une architecture calquée sur le modèle trois-tiers ... 62
2.1.1 Présentation ... 62
2.1.2 Les entités ... 63
2.1.2.1 cmdpdp ... 63
2.1.2.2 mysql ... 63
2.1.2.3 pdp ... 64
2.1.2.4 pep ... 64
2.2 Une conception modulaire ... 64
2.2.1 pdp ... 64
2.2.1.1 Les interfaces ... 65
2.2.1.2 Le cœur ... 66
2.2.2 pep ... 66
2.3 Principes de fonctionnement ... 67
2.3.1 Interaction entre les modules ... 67
2.3.1.1 Traitement des commandes de la console de gestion ... 67
2.3.1.2 Traitement des messages non sollicités du PEP ... 70
2.3.2 Le concept de contexte ... 72
2.3.2.1 Définition ... 72
2.3.2.2 Fonctionnement ... 72
2.3.2.2.1 Principe ... 72
2.3.2.2.2 Exemple ... 73
3 Le service contentAdaptation ... 73
3.1 Caractéristiques ... 73
3.1.1 Puissance ... 74
3.1.1.1 Gestion de politiques complexes ... 74
3.1.1.2 Mécanisme de rollback ... 75
3.1.1.3 Modification des politiques sans réinstallation ... 75
3.1.2 Souplesse ... 75
3.1.2.1 Langage des politiques libre ... 75
3.1.2.2 Interfaçage facile avec le PEP ... 76
3.2 Implémentation ... 76
3.2.1 De nouvelles commandes ... 76
3.2.1.1 Commande d'installation ... 76
3.2.1.2 Commande de modification ... 78
3.2.2 Un service PDP complexe ... 79
3.2.2.1 Fonctionnement de la suppression ... 79
3.2.2.2 Fonctionnement de l'installation ... 81
3.2.2.3 Fonctionnement de la modification ... 82
3.2.3 Un client PEP souple ... 86
3.2.3.1 Conception modulaire ... 86
3.2.3.2 Fonctionnement ... 86
3.2.4 Une extension de COPS : COPS-CA ... 87
3.2.4.1 Nouveaux objets ... 87
3.2.4.2 Organisation des objets COPS-CA ... 88
3.2.4.2.1 Message install ... 88
3.2.4.2.2 Message remove ... 88
3.2.4.2.3 Message modify ... 89
3.3 Démonstration ... 89
3.3.1 Plateforme ... 89
3.3.2 Scénarios ... 91

3.3.2.1 Fonctionnement nominal 91
3.3.2.1.1 Installation d'une politique complexe..... 92
3.3.2.1.2 Première modification de la politique..... 93
3.3.2.1.3 Deuxième modification de la politique..... 94
3.3.2.1.4 Suppression de la politique..... 95
3.3.2.2 Erreur pendant une modification 96
3.3.2.3 Erreur pendant un rollback 98
3.3.3 Performances 101

Conclusion 102

Références 103

Glossaire et Acronymes 107

Annexe A Exemples de messages COPS utilisant des objets COPS-CA 109

TABLE DES FIGURES

Figure 1-1 : Répartition du chiffre d'affaire par sites 13
Figure 1-2 : Evolution du chiffre d'affaires de Thales 13
Figure 1-3 : Répartition du chiffre d'affaire par pôles 14
Figure 1-4 : Chiffre d'affaire de Thales Land & Joint Systems 15
Figure 1-5 : Répartition du CA de Thales Land & Joint Systems par domaines d'activités 16
Figure 1-6 : Organigramme de Thales Land & Joint Systems 17
Figure 2-1 : Notion du PBNM 19
Figure 2-2 : Structure d'un PBN (modèle trois-tiers) 23
Figure 2-3 : Exemple de console de gestion avec interface graphique 24
Figure 2-4 : PBN avec LPDP 25
Figure 2-5 : Structure d'un PEP 26
Figure 2-6 : Structure d'un PBN (modèle deux-tiers) 27
Figure 2-7 : Exemple de PBN hétéroclite 28
Figure 2-8 : Exemple d'un PBN évolué 29
Figure 2-9 : Un message COPS 32
Figure 2-10 : Entête d'un objet COPS 33
Figure 2-11 : Entête d'un message COPS 34
Figure 2-12 : Relations entre messages et objets COPS 36
Figure 2-13 : Sessions COPS et connexions TCP 37
Figure 2-14 : Echanges COPS : Ouverture de session 37
Figure 2-15 : Echanges COPS : Négociation des numéros de séquence 39
Figure 2-16 : Echanges COPS : Création d'un état 41
Figure 2-17 : Echanges COPS : Modification d'un état par le PDP 42
Figure 2-18 : Echanges COPS : Modification d'un état par le PEP 43
Figure 2-19 : Echanges COPS : Suppression d'un état par le PDP 44
Figure 2-20 : Echanges COPS : Synchronisation 45
Figure 2-21 : Relations entre les classes PCIM 50
Figure 2-22 : Exemple de topologie d'un *policy repository* distribué 52
Figure 2-23 : Exemple de DIT 54
Figure 2-24 : Exemple d'utilisation du *replication service* de LDAP 55
Figure 2-25 : Exemple d'utilisation du *referral service* de LDAP 55
Figure 3-1 : Les entité de l'architecture PDP/PEP de RTIPA 62
Figure 3-2 : PBN basé sur une structure trois-tiers simplifiée 63
Figure 3-3 : Structure de la base de données du *policy repository* 64
Figure 3-4 : Décomposition modulaire du pdp 65
Figure 3-5 : Décomposition modulaire du pep 66
Figure 3-6 : Interaction entre les modules : installation d'une politique 68
Figure 3-7 : Interaction entre les modules : suppression d'une politique 69
Figure 3-8 : Interaction entre les modules : demande de configuration 71
Figure 3-9 : Exemple d'automate fini associé à un contexte 73
Figure 3-10 : Exemple de fonctionnement d'un automate fini 73
Figure 3-11 : Exemple de politique complexe 74

Figure 3-12 : Utilisation d'une bibliothèque de fonction extérieur au service contentAdaptation 76
Figure 3-13 : Automate fini du service contentAdaptation : Suppression..... 79
Figure 3-14 : Exemple de suppression d'une politique sans erreur..... 80
Figure 3-15 : Exemple de suppression d'une politique avec erreur 80
Figure 3-16 : Automate fini du service contentAdaptation : Installation 81
Figure 3-17 : Automate fini du service contentAdaptation : Modification 83
Figure 3-18 : Interaction entre les modules : modification d'une politique 85
Figure 3-19 : Conception modulaire du client contentAdaptation 86
Figure 3-20 : Interaction entre les modules du client contentAdaptation..... 87
Figure 3-21 : Structure d'un objet contentAdaptation 88
Figure 3-22 : Organisation des objets COPS-CA dans un message COPS DEC *install* 88
Figure 3-23 : Organisation des objets COPS-CA dans un message COPS DEC *remove* 89
Figure 3-24 : Organisation des objets COPS-CA dans un message COPS DEC *modify* 89
Figure 3-25 : Plateforme de développement et de démonstration du service TAI 90
Figure 3-26 : Plateforme de démonstration..... 91
Figure 3-27 : Réseau virtuel de la démonstration..... 91
Figure 3-28 : Scénario 1, étape 1 : Configuration du réseau virtuel..... 92
Figure 3-29 : Scénario 1, étape 1 : Messages COPS envoyés 93
Figure 3-30 : Scénario 1, étape 2 : Configuration du réseau virtuel..... 93
Figure 3-31 : Scénario 1, étape 2 : Messages COPS envoyés 94
Figure 3-32 : Scénario 1, étape 3 : Messages COPS envoyés 95
Figure 3-33 : Scénario 1, étape 4 : Messages COPS envoyés 95
Figure 3-34 : Scénario 2, installation 96
Figure 3-35 : Scénario 2, modification : fonctionnement nominal jusqu'à erreur 97
Figure 3-36 : Scénario 2, modification : *rollback* 98
Figure 3-37 : Scénario 3, installation 99
Figure 3-38 : Scénario 3, modification : fonctionnement nominal jusqu'à erreur 99
Figure 3-39 : Scénario 3, modification : *rollback* jusqu'à erreur 100
Figure 3-40 : Scénario 3, modification : suppression brutale de la politique 100

INTRODUCTION

Ce rapport présente les travaux que j'ai réalisés pendant mon stage de DEA. L'objectif de celui-ci a été la mise en place d'un système permettant de gérer des modules applicatifs disséminés dans un réseau. Ce stage s'insère dans le projet RNRT RHODOS où ces modules serviront à adapter le contenu de flux multimédia, à la demande des clients.

Le principal sujet de ce stage a donc été la gestion de réseau par politique, également appelée PBNM (Policy-Based Network Management). En effet, ce système de gestion est parfaitement adapté aux contraintes imposées par le projet RHODOS.

Ainsi, dans un premier temps, j'ai effectué une importante recherche bibliographique sur ce sujet, en étudiant plus particulièrement le protocole COPS (Common Open Policy Service). Ce travail s'est achevé par la rédaction d'un rapport bibliographique.

La suite du stage a consisté à développer un service applicatif, baptisé « contentAdaptation », basé sur le protocole COPS et son architecture PDP/PEP (Policy Decision Point / Policy Enforcement Point). Cette architecture ayant déjà été mise en place par TAI pour d'autres projets, j'ai commencé par lire son code afin de comprendre son fonctionnement. Ceci fait, je l'ai étendue pour y intégrer le service contentAdaptation.

Ce rapport est divisé en trois chapitres. Le premier présente le groupe Thales et son service TAI qui m'a accueilli pendant six mois.

Le deuxième chapitre traite de la gestion par politiques. Il s'agit d'une version légèrement allégée de mon rapport bibliographique. Il introduit les notions essentielles de la gestion par politiques, montre les architectures possibles d'un réseau ainsi géré, détaille le fonctionnement du protocole COPS et expose quelques solutions pour représenter et stocker des politiques.

Le troisième chapitre parle du stage proprement dit. Il présente le projet RHODOS en précisant le rôle de Thales et le mien. Puis il décrit l'architecture PDP/PEP développée par TAI. Enfin, il s'intéresse au service que j'ai développé, d'une part en détaillant ses caractéristiques et son implémentation, d'autre part en présentant la démonstration que j'ai exposée à TAI.

Chapitre 1 : Le groupe Thales

1 Le groupe dans son ensemble

1.1 Activités

La mission de Thales est d'apporter à ses clients des solutions spécifiques de haute technologie pour répondre à leurs besoins de sécurité. Ses activités sont donc centrées autour de la sécurité qui représente 80% de son activité.

La sécurité peut être aussi bien celle des personnes, des Etats ou des biens. Ainsi Thales assure la sécurité :

- Dans les domaines militaires : équipements des soldats, armements, systèmes de surveillance…
- Des transports, principalement aériens : simulateurs, systèmes de navigations, contrôles aériens…
- Des sites et des personnes : systèmes de contrôle d'accès et de vidéosurveillance ou encore cartes d'identité infalsifiables…
- Des communications et des transactions électroniques

1.2 Stratégie

Thales poursuit depuis quelques années avec constance une stratégie fondée sur trois axes essentiels :

- se positionner sur des métiers de très haute technologie et à forte valeur ajoutée ;
- faire jouer les synergies entre ses activités civiles et militaires grâce aux technologies duales ;
- renforcer de manière significative sa dimension internationale en implantant des sites sur tous les continents.

Ce qui sous-tend cette stratégie est la volonté de l'ensemble du groupe de toujours mieux satisfaire ses clients, d'une part en adaptant ses offres à leurs nouveaux besoins, d'autre part en exerçant ses activités dans les pays où ils se trouvent.

Aujourd'hui Thales est implanté dans 50 pays et près de 50 % de ses 61500 collaborateurs travaillent à l'étranger.

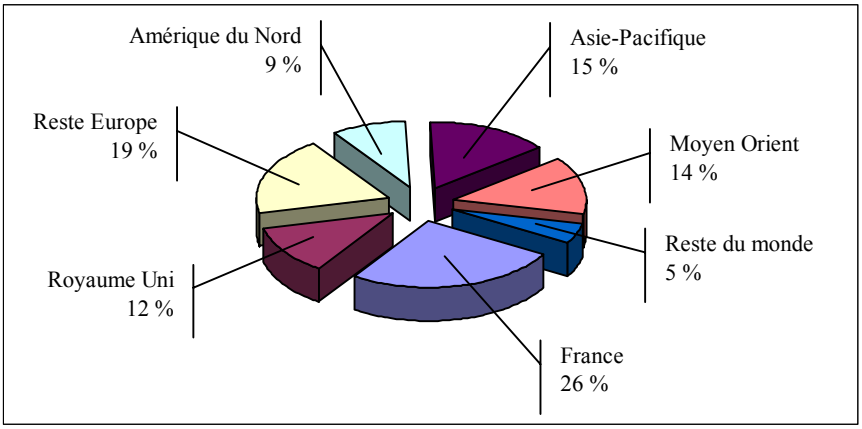


Figure 1-1 : Répartition du chiffre d'affaire par sites

De toute évidence, cette stratégie est payante puisqu'elle permet à Thales de rencontrer des succès

- en termes de croissance et de résultats,
- en termes de développement international et
- en termes de succès technologiques et commerciaux.

L'évolution du chiffre d'affaire au cours de ces dernières années illustre bien la bonne santé de Thales.

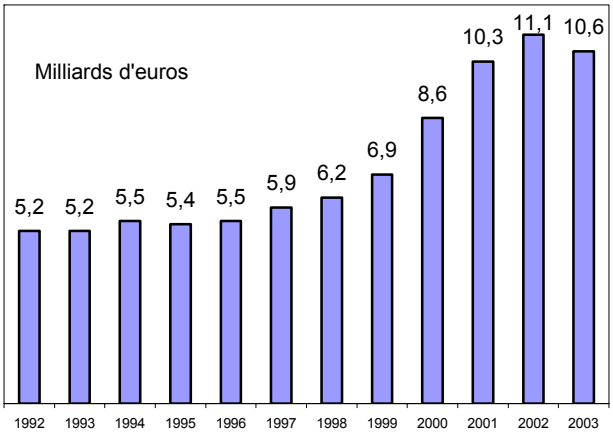


Figure 1-2 : Evolution du chiffre d'affaires de Thales

Malgré une légère régression, les résultats de 2003 sont très bons avec un carnet de commande qui s'élève à 19 milliards d'euros soit près de deux ans de chiffre d'affaire et un bénéfice net de 111 millions d'euros.

1.3 Structure

Thales est structuré autour de trois grands pôles :

- La Défense
- L'Aéronautique
- Les Technologie de l'Information & des Services

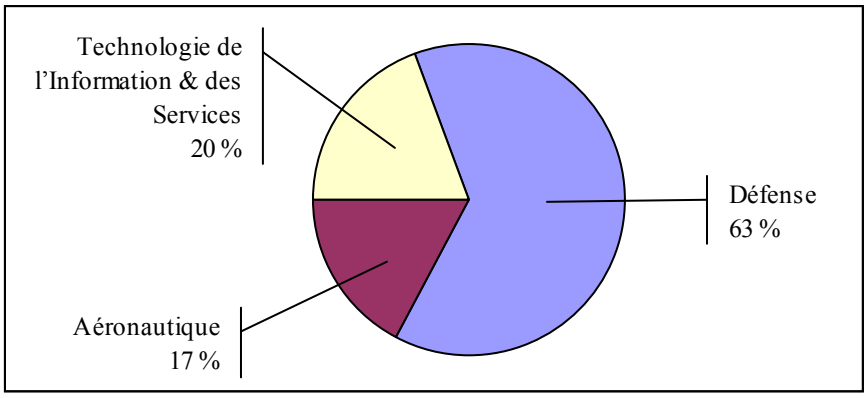


Figure 1-3 : Répartition du chiffre d'affaire par pôles

1.3.1 Le pôle Défense

Les récents conflit ont montré combien les systèmes électroniques modifie « l'art de la guerre ». A tel point que l'on parle aujourd'hui de révolution dans la conduite des opérations militaires. De ce fait, le pôle Défense est en croissance constante depuis plusieurs années grâce à sa maîtrise technologique, sa maîtrise d'œuvre de grands projets, sa capacité à interconnecter différentes forces et sa stratégie multi-domestique.

Le pôle Défense est présent dans les domaines :

- terrestres : systèmes d'information et de communications, systèmes optroniques, armement, simulation…
- navales : systèmes de surveillance, modernisation de bâtiments existants, activité sous-marine, communication…
- aériens : drones, systèmes optroniques, communication, identification, armement…
- spatiales.

1.3.2 Le pôle Aéronautique

Thales est le seul groupe au monde couvrant l'ensemble de la chaîne de la sécurité aérienne. Les compétences du pôle Aéronautique vont des systèmes électroniques de vol jusqu'à la gestion du trafic aérien en passant par l'entraînement des équipages.

Le pôle Aéronautique travaille aussi bien dans le domaine militaire que civil.

1.3.3 Le pôle Technologie de l'Information & des Services

Le pôle Technologie de l'Information & des Services (*Information Technologies & Services*, IT&S) regroupe les activités du groupe qui s'adressent prioritairement aux besoins des marchés civils de haute technologie ; finance, industrie, administrations…

Dans ce pôle, Thales privilégie le développement d'activités à fort contenu technologique valorisant des synergies importantes avec les pôles de Défense et Aéronautique. Le pôle se concentre ainsi sur les activités dont il estime pouvoir garantir la croissance et la rentabilité, notamment grâce aux technologies duales civiles/militaires :

- Sécurité : des réseaux et communications, des paiements et transactions électroniques, des personnes et des sites…
- Solution de positionnement : navigation, télématique, géosolutions…
- Composants de communication
- ...

1.4 Thales et la recherche

Afin de conserver son avance dans les technologies et services clefs, et pour mieux répondre aux besoins des marchés, Thales maintient un niveau élevé de recherche & développement. Il y consacre 17 % de son chiffre d'affaire, dont 23 % sont autofinancés.

Fort d'un portefeuille de 15000 brevets, Thales protège chaque année, quelques 250 nouvelles inventions. Cependant, il accorde également de plus en plus d'importance à l'utilisation et la mise en place de standards afin de faciliter l'interopérabilité des systèmes.

La recherche de Thales est ouverte sur l'extérieur grâce à des partenariats avec des centres universitaires et de recherche prestigieux du monde entier : Polytechnique, CNRS, CEA, INRIA… en France, QINETIQ au Royaume Uni… Thales participe également très activement aux programmes de recherche français et européens.

2 Thales Land & Joint Systems

La section Thales Land and Joint Systems est apparue en mars 2004. Elle regroupe principalement Thales Communication et Thales Optronique et se situe au sein du pôle Défense.

Il s'agit d'une section très importante de Thales puisqu'elle emploie 12000 personnes dans 19 pays et devrait réaliser un chiffre d'affaire de 2.4 milliards d'euros en 2004.

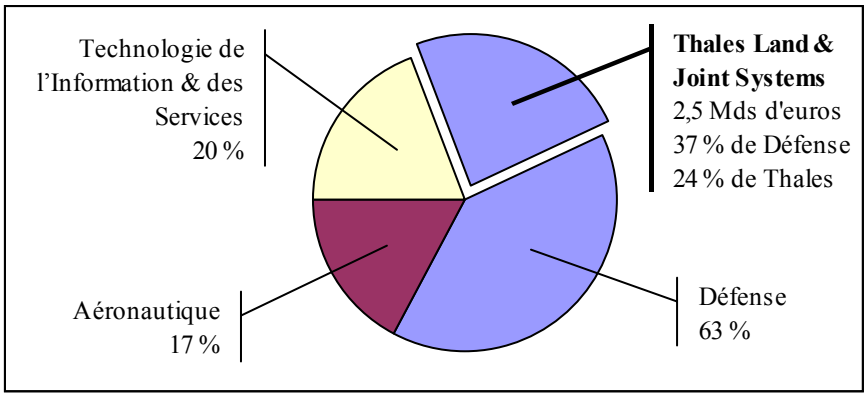


Figure 1-4 : Chiffre d'affaire de Thales Land & Joint Systems

2.1 Activités

Les activités de cette section sont structurées en cinq domaines :

- Systèmes terrestres : Maîtrise d'œuvre et intégration système pour des programmes terrestres majeurs : systèmes soldat, systèmes véhicule, robots terrestres, armement et numérisation de l'espace de travail…
- Systèmes interarmées : Maîtrise d'œuvre et intégration système pour de grands programmes interarmées : réseaux d'infrastructure, communications spatiales, systèmes de commandement, renseignement, surveillance, reconnaissance…
- Equipements et sous-systèmes de communication et optronique : Communication, armement, navigation, identification… pour les armées Terre, Mer et Air.
- Services : Eventail complet de services à valeur ajoutée pour aider le client à optimiser le fonctionnement de leurs systèmes sur le terrain.

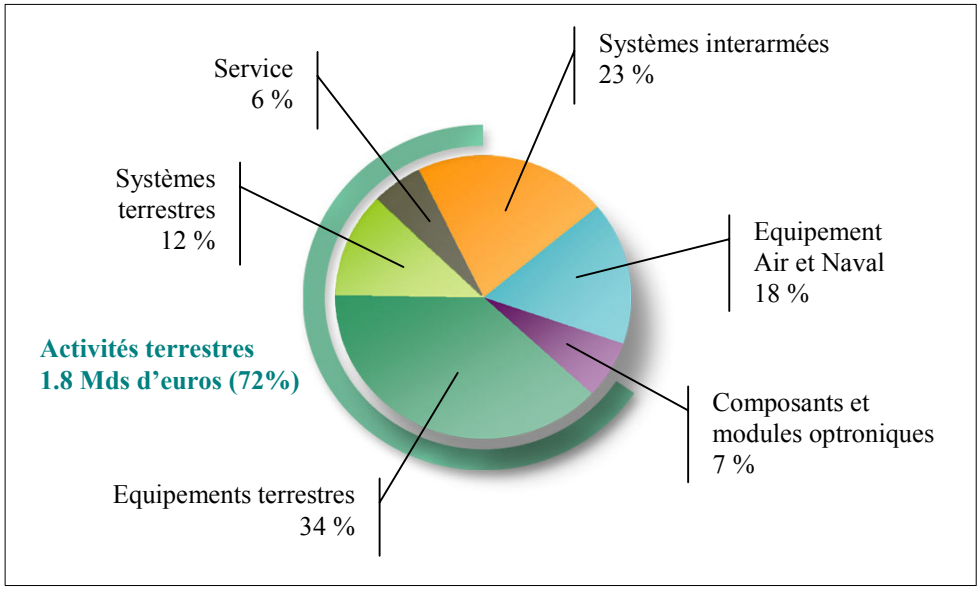


Figure 1-5 : Répartition du CA de Thales Land & Joint Systems par domaines d'activités

2.2 Rôle

L'objectif Thales Land and Joint Systems est de réunir dans le même organisme toutes les compétences pour traiter le concept de C4ISR (*Command, Control, Communication, Computing, Intelligence, Surveillance, Reconnaissance*). Ce concept est apparu il y a environ cinq ans et constitue une révolution majeur dans les affaires militaires. Il permet une meilleur interopérabilité entre alliés, des interventions plus rapides et plus précises. On comprend donc que Thales mette tout en œuvre pour mettre en valeur ses compétences dans ce domaine devenu crucial pour ses clients.

3 Le service TAI

3.1 Organisation

Le service Technologies Avancées de l'Information (TAI) se situe au sein de l'entité Thales Land & Joint Systems.

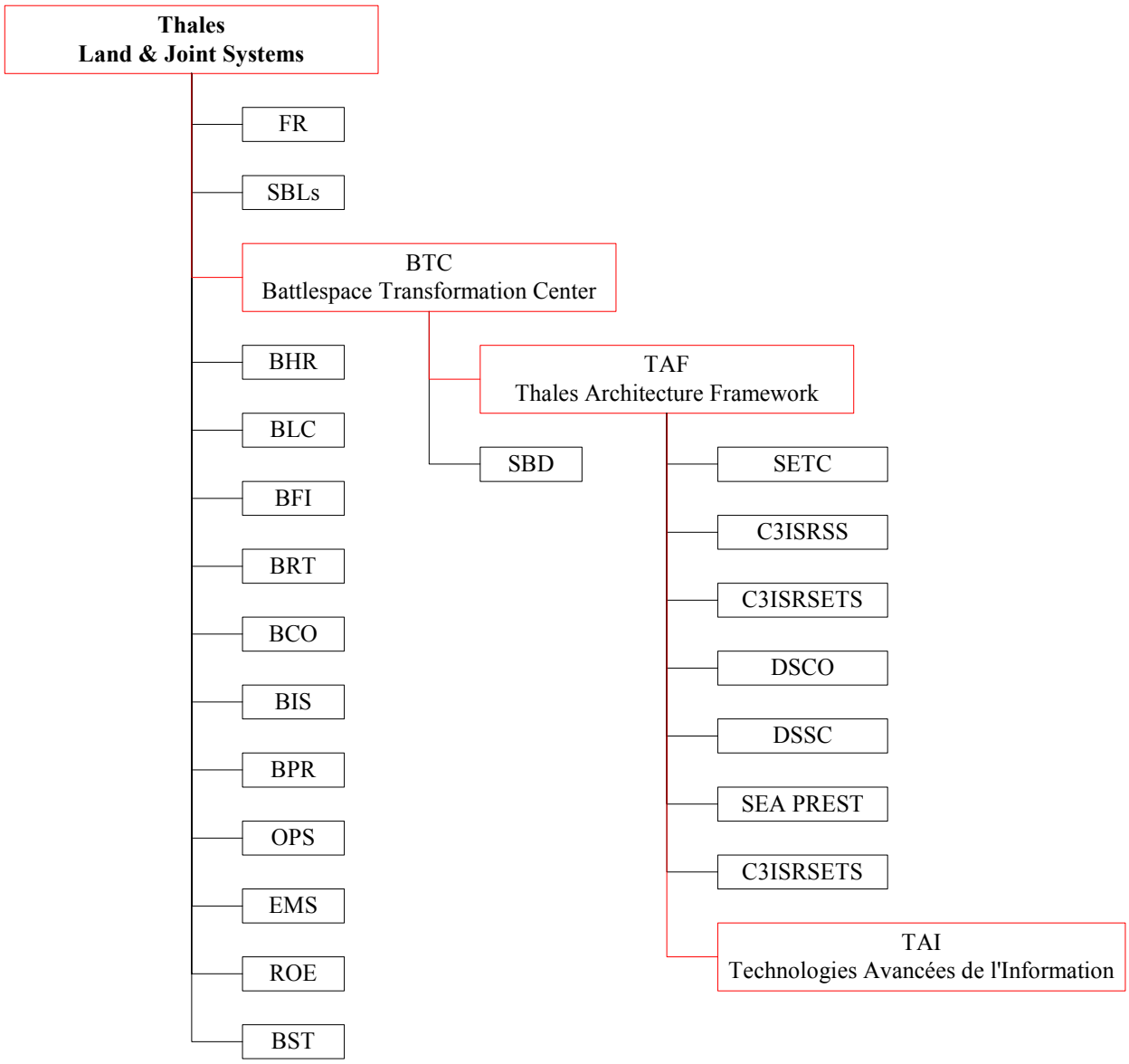


Figure 1-6 : Organigramme de Thales Land & Joint Systems

TAI occupe une place particulière dans Thales Land & Joint Systems car il ne s'agit pas d'un service de production mais de recherche. Il agit en amont du service décisionnel des orientations techniques pour les différents produits proposés par Thales Land & Joint Systems.

3.2 Rôle

Le rôle de TAI est triple :

- Premièrement, ce service doit être l'éclaireur de la société sur le terrain des nouvelles technologies de l'information, naissantes au sein de laboratoires académiques ou industriels.
- Deuxièmement, ce service doit être un acteur clé, en étant un intermédiaire de choix entre l'entreprise et le monde extérieur afin de favoriser le partage des connaissances dans le domaine des hautes technologies. De cette manière, le groupe apporte sa contribution à l'effort de recherche collectif.
- Troisièmement, TAI doit avoir, pour le service décisionnel des orientations techniques, le même rôle qu'un bureau d'étude, en fournissant des rapports structurés et concis sur ses différents domaines d'investigation.

Ainsi, le service « Technologies Avancées de l'Information » repère, sélectionne, teste et développe différentes technologies de l'Information et des Communications répondant aux besoins des nouveaux produits et systèmes élaborés par Thales.

Dans ce but, le service TAI multiplie ses participations aux grands projets de la recherche européenne et française, notamment RNRT.

Actuellement, les champs d'investigations du service TAI sont articulés autour de quatre thèmes majeurs : les applications multimédia, les futurs réseaux en cours de définition par l'IETF (Internet Engineering Task Force), les environnements mobiles et la standardisation, la sécurité.

Chapitre 2 : La gestion par politiques

1 Introduction au PBNM (Policy-Based Network Management)

Jusqu'à récemment, l'administration d'un réseau se plaçait d'un point de vue ingénierie. Les outils mis à la disposition de l'administrateur, comme SNMP (Simple Network Management Protocol), étaient destinés à détecter et réparer les problèmes dans le réseau.

La gestion par politiques, ou PBNM, marque une rupture avec cette optique. On considère cette fois que le réseau est fonctionnel. Le rôle de l'administrateur est alors de le régler afin d'atteindre certains objectifs (QoS, sécurité…) définis à l'avance.

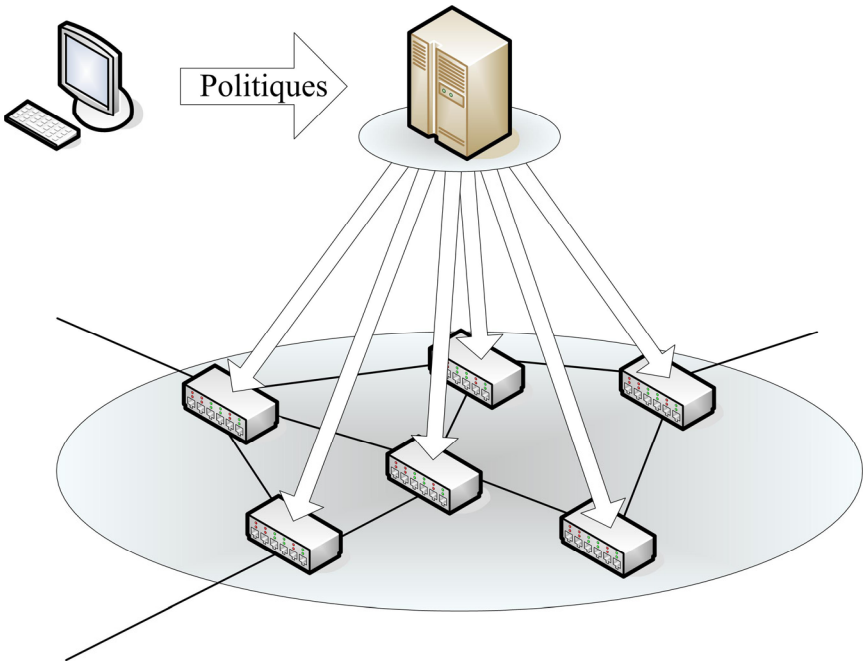


Figure 2-1 : Notion du PBNM

Figure 2-1 donne une notion très abstraite du PBNM. Le principe est le même pour n'importe quel type de service (QoS, sécurité…) :

- Lorsqu'un flux arrive dans un nœud actif du réseau, ce dernier interroge une entité qui a une vue globale du réseau pour savoir ce qu'il doit faire.
- Cette entité décisionnelle s'appuie sur des règles pour répondre aux questions que les nœuds actifs lui posent.
- Ces règles sont mises en place par l'administrateur du réseau via une console de gestion conviviale.

L'élément principal et novateur du PBNM est donc la notion de règles de politiques que nous allons détailler dans la section suivante.

1.1 Les politiques et les règles

Une politique est un ensemble de règles qui permet de fixer le comportement des éléments du réseau qu'elle administre. Par élément, on entend service, utilisateur ou équipement.

Une règle est typiquement exprimées sous la forme de couples (condition , action). Ainsi on peut imaginer une règle limitant le trafic Internet pendant les heures de travail à 2 Mbps pour les commerciaux :

```
SI Groupe-utilisateur = Commerciaux ET Heure = 8:00-18:00
ALORS Bande-passante-Internet = 2Mbps
```

Il faut bien remarquer que cette règle ne dit pas comment limiter la bande passante. Elle se contente de spécifier ce qu'il faut faire. Il s'agit d'une grande différence (et d'un grand progrès) par rapport à l'administration traditionnelle où un opérateur devait traduire les objectifs en commandes machine à la main (beaucoup de risques d'erreur et de perte de temps).

Bien sûr, il n'est pas possible de spécifier ces règles avec le langage humain trop ambigu comme cela a été fait dans l'exemple ci-dessus. En fait, il existe des langages spécialisés: PCIM [28], Ponder [32], PFDL [33]… Nous étudierons PCIM dans la section 4.1 page 48.

Bien que très succincte, cette présentation des règles de politique laisse entrevoir à quel point le PBNM est un outil d'administration puissant. Cela n'est pas surprenant, compte tenu des objectifs imposés par [6], énoncés ci-dessous.

1.2 Les objectifs et contraintes

Les mécanismes et les protocoles conçus pour fournir une gestion basée sur des politiques obéissent aux objectifs et contraintes suivantes :

- Il n'y a aucune limitation sur les politiques envisageables.
- Les mécanismes doivent supporter la préemption. C'est à dire qu'il doit être possible de supprimer ou modifier des états précédemment installés à n'importe quel moment pour modifier la configuration du réseau.
- Ils doivent fournir des outils de surveillance (monitoring) notamment pour la comptabilité et la facturation.

- Ils doivent être résistants aux failles (défaillances d'un PDP, PEP ou lien de communication). C'est à dire qu'ils doivent pouvoir continuer à fonctionner, au moins temporairement, pendant une défaillance et pouvoir reprendre leur fonctionnement normal une fois celle-ci terminée.
- Il n'est pas indispensable que tous les nœuds du réseau soient actifs. Les nœuds inactifs sont appelés des PINs (Policy Ignorant Node).
- Le passage à l'échelle est un des points les plus importants. Si celui-ci est limité, cela ne doit pas venir de la gestion par politique mais des techniques utilisées pour le service rendu (RSVP, IntServ, DiffServ, Ipsec…). Le multicast doit notamment être pris en compte, au moins autant que le service rendu.
- Enfin, les mécanismes doivent être sécurisés. Ils doivent en particulier minimiser les risques de vol d'identité et de dénis de service.

Les objectifs du PBNM sont donc très ambitieux. Bien évidemment, toutes ces contraintes engendrent un surcoût dans la complexité du réseau. Il est légitime, dans ce cas, de se demander dans quels scénarios cette augmentation de la complexité se justifie par les avantages qu'elle apporte.

1.3 Quelques scénarios d'utilisation

Voici quelques exemples de scénarios rendus possibles, facilités ou améliorés grâce au PBNM. Dans certains de ces scénarios, il reste des problèmes non résolus par les politiques. Ceux-ci seront également signalés.

♦ *Accords bilatéraux entre ISPs (Internet Service Provider)*

Jusqu'à récemment, les accords entre ISPs concernant le trafic traversant leur frontière étaient très simples. Généralement, les deux ISPs acceptaient tout le trafic sans aucun système de comptabilité ou de paiement. Cependant, avec l'apparition des mécanismes de QoS basés sur DiffServ ou IntServ, il devient important de différencier les sources des flux. Cela devient même vital si les deux ISPs voisins gèrent des clientèles différentes qui exigent des QoS différentes (par exemple, les entreprises sont prêtes à payer cher pour avoir une QoS élevée, tandis que les particuliers se satisfont d'une QoS moindre si cela réduit le prix du service). En effet, l'absence de système de comptabilité suffisamment sophistiqué entre de tels ISPs peut mener à des allocations de ressource inefficaces.

Le PBNM permet un tel système de comptage même si cela peut nécessiter l'ajout d'informations complémentaires par exemple dans des messages RSVP.

♦ *Gestion fine des priorités*

Le PBNM permet de façon assez évidente de gérer finement les priorités. Par exemple, pour une priorité donnée, réserver des ressources pour une durée limitée.

La vue globale de l'élément décisionnel rend possible d'accepter une réservation en toute connaissance de cause. Ce n'est pas parce qu'il y a suffisamment de ressource dans un nœud donné que c'est aussi le cas dans le reste du réseau. De plus, en cas de mécanisme préemptif, il est intéressant d'interrompre des flux moins prioritaires dans tout le réseau de la façon la plus équitable possible.

Cependant, il apparaît ici une difficulté que le PBNM ne résout pas : comment gérer cette notion d'équité ? Il est en effet, facile de gérer les priorités lorsque les flux sont homogènes (bande passante, délai, mémoire…).

En revanche, lorsqu'ils sont hétérogènes, il n'y a plus de manière simple de procéder. Par exemple, faut-il accepter un seul flux qui demande beaucoup de ressources en détriment d'un grand nombre de flux moins prioritaires qui en demandent peu ? Des éléments de réponse sont donnés dans [15].

♦ *Appels par carte pré-payée*

Dans les réseaux téléphoniques, le modèle des cartes pré-payées, ou carte à points, est de plus en plus populaire. Ce modèle est transposable dans les réseaux informatiques, par exemple pour les bornes publiques Wi-Fi. Le principe en est assez simple : l'utilisateur qui veut obtenir certaines ressources indique l'identifiant de sa « carte », le nœud actif du réseau recevant la demande transmet celle-ci à l'organe décisionnel qui vérifie si le crédit lié à l'identifiant n'est pas épuisé et valide la réservation si ce n'est pas le cas. Une fois la réservation établie, le crédit de la « carte » est diminué selon un algorithme quelconque.

Bien que le principe soit très simple dans un PBN (Policy-Based Network), sa réalisation pose quelques problèmes. Il faut résoudre certaines questions :

- Quelle est l'étendue de la facturation?
- Quand le crédit commence-t-il à être décompté ?
- Comment savoir quand est épuisé le crédit ?

Pour la première question, il faut distinguer deux cas différents si la réservation concerne plusieurs ADs (Administrative Domain). Dans un cas ces ADs ont passé des accords entre eux, seul le réseau sur lequel se trouve l'utilisateur modifiera le crédit de la « carte ». Dans le cas contraire, tous les réseaux concernés chercheront à accéder, en concurrence, à la base de données des cartes pour décrémenter le crédit de celle-ci. Les problèmes relatifs à la localisation et la gestion de cette base de données n'entre pas dans le cadre du PBNM.

La réponse à la deuxième question est laissée à l'initiative de l'opérateur. Cependant, le PBNM met à la disponibilité de celui-ci les outils pour faire commencer le décompte aussi bien dès la demande, qu'après la réservation effective des ressources, du moins tant que celle-ci ne concerne qu'un seul AD. Si elle concerne plusieurs ADs, le PBNM ne fournit aucun système pour notifier au réseau d'où provient l'appel, à quel instant toutes les réservations ont été mises en place et donc à quel moment commencer le décompte du crédit.

Enfin, pour ce qui est de la dernière question, le PBNM ne permet pas de mettre en place un système qui supprimerait la réservation dès que le crédit est épuisé (c'est pourtant ce qui intéresserait l'opérateur). Le mieux qu'il est possible de faire, est que l'organe décisionnel consulte régulièrement la base de données des crédits pour savoir si la « carte » de l'utilisateur n'est pas épuisée.

2 L'architecture

C'est le groupe de travail RAP (Ressource Allocation Protocol) de l'IETF qui est à l'origine de la structure des PBNs. Elle est décrite dans [6] et se caractérise par sa simplicité puisqu'on n'y trouve que deux composants principaux :

- Le PDP qui est l'organe décisionnel.
- Les PEPs qui sont les points où sont appliqués les décisions.

2.1 Modèle trois-tiers

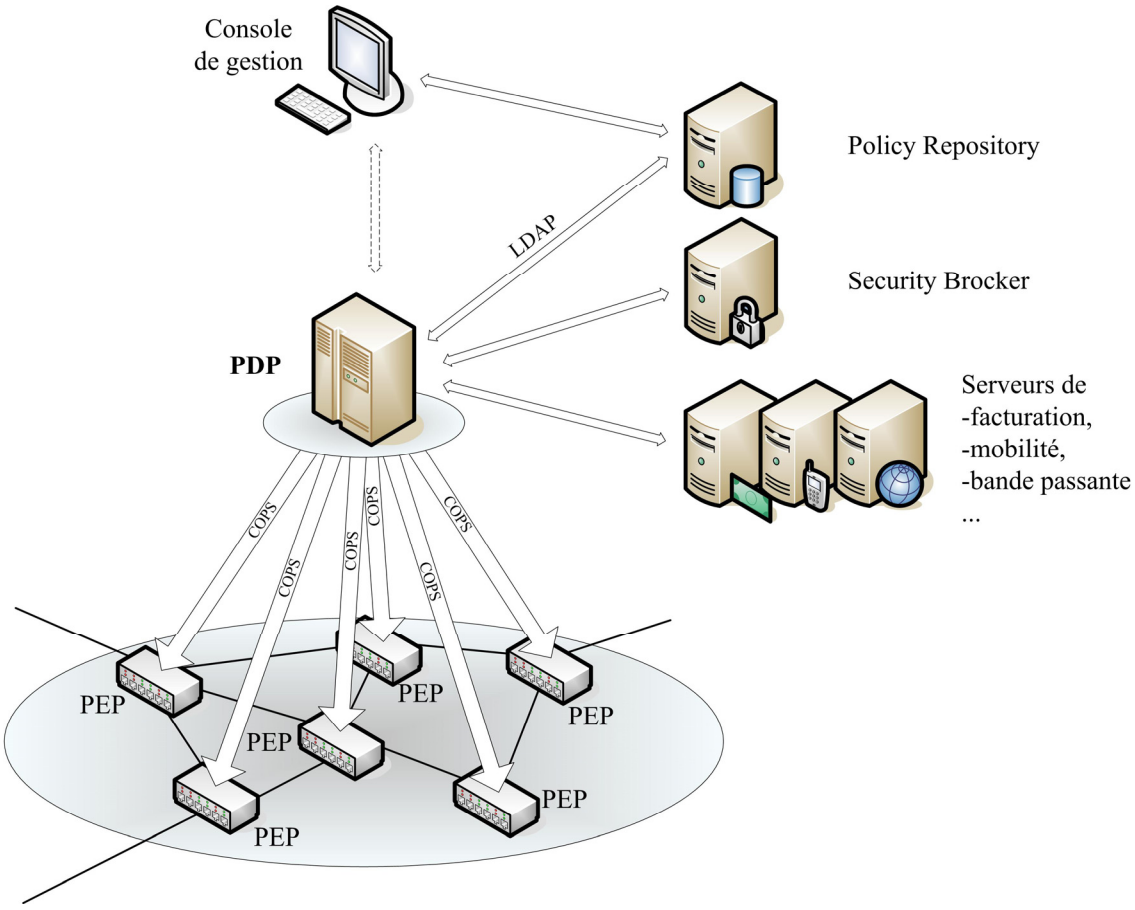


Figure 2-2 : Structure d'un PBN (modèle trois-tiers)

Figure 2-2 introduit le modèle dit « trois-tiers ». La section 2.2 page 27 en présente d'autres, mais celui-ci est le plus simple. C'est donc lui qui servira de référence pour décrire les composants d'un PBN puisque ces composants sont communs à tous les modèles.

2.1.1 Console de gestion

La console de gestion possède une interface conviviale (par exemple sous forme d'une page HTML) pour permettre à l'administrateur d'éditer, modifier et consulter les règles s'appliquant à son réseau.

Ces règles peuvent être :

- stockées dans une base de données
- ou bien directement installées dans le PDP, par exemple si on veut forcer un changement immédiat du comportement du réseau ou encore s'il n'y a, tout simplement, pas de base de données.

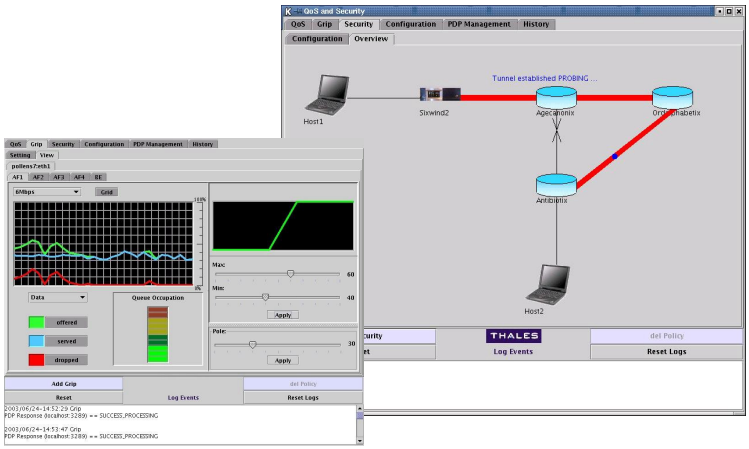


Figure 2-3 : Exemple de console de gestion avec interface graphique

Optionnellement, la console de gestion peut aussi gérer les conflits entre les règles. En effet, dans un système complexe, éventuellement administré par plusieurs personnes, il est impossible de garantir que toutes les règles seront compatibles entre elles. Il existe plusieurs types de stratégies pour détecter et/ou résoudre des règles conflictuelles, cependant aucune solution n'est exempte de défauts.

Il n'y a pas de protocole de communication normalisé entre la console et le PDP ou entre la console et la base de données.

2.1.2 Le PDP (Policy Decision Point)

2.1.2.1 L'organe décisionnel de l'architecture

Figure 2-2 met en avant le rôle central du PDP. Comme cela a déjà été dit plusieurs fois, il s'agit de l'organe décisionnel du PBN. Les PEPs doivent se référer à lui dès qu'un changement intervient dans un de leurs états (une définition précise d'un état sera donnée lors de la description de COPS).

Pour prendre sa décision, le PDP se base sur :

- Le changement d'état du PEP : sa configuration a été modifiée (par exemple une de ses interfaces est tombée en panne), un nouveau flux RSVP essaie de pénétrer dans le réseau…
- L'état du réseau : est-il congestionné ?
- D'autres informations dynamiques : l'heure, le crédit restant sur un compte…

Ayant recueilli toutes ces informations, le PDP recherche la règle dont la condition correspond à celles-ci et retourne au PEP l'action associée.

2.1.2.2 Les autres serveurs

En tant qu'élément central de l'architecture, le PDP s'appuie sur plusieurs autres serveurs aussi bien pour recueillir des informations complémentaires, que pour rechercher la politique à appliquer ou encore pour déclencher des processus après la prise de décision (pour la facturation par exemple).

Il peut donc être intéressant de joindre au PDP des serveurs supplémentaires :

- La base de données des politiques (*policy repository*) est en charge du stockage des règles de QoS (ou autre…). L'IETF préconise l'utilisation d'annuaires pour implémenter le *policy repository* et celle du protocole LDAP (Lightweight Directory Access Protocol) pour y accéder.

- Le gestionnaire de bande passante (*bandwidth broker*) connaît la topologie et les caractéristiques du réseau ce qui lui permet de distribuer les ressources à bon escient.
- Le gestionnaire de sécurité (*security brocker*) est généralement un serveur AAA (Authentification, Autorisation and Accounting).
- Le serveur de mobilité (m*obility brocker*) peut gérer la continuité du service (comme la QoS) pendant un déplacement.
- Le serveur de facturation sera essentiel lorsque les réseaux feront de la QoS car sans différenciation tarifaire entre les CoS (Class of Service) il est impossible de mettre en place une quelconque qualité de service.

Bien sûr aucun de ces serveurs n'est indispensable. Il est possible de construire un PBN sans aucun de ces serveurs, de même qu'il est possible que, dans le futur, d'autres serveurs deviennent utiles pour de nouveaux services. Le seul serveur qui soit, peut être, plus important que les autres, est le *policy repository*.

2.1.2.3 Le LPDP (Local Policy Decision Point)

Dans certains cas, il est possible qu'un nœud du réseau ait suffisamment de ressources (processeur, mémoire…) pour prendre lui-même une première décision. On lui ajoute alors un LPDP.

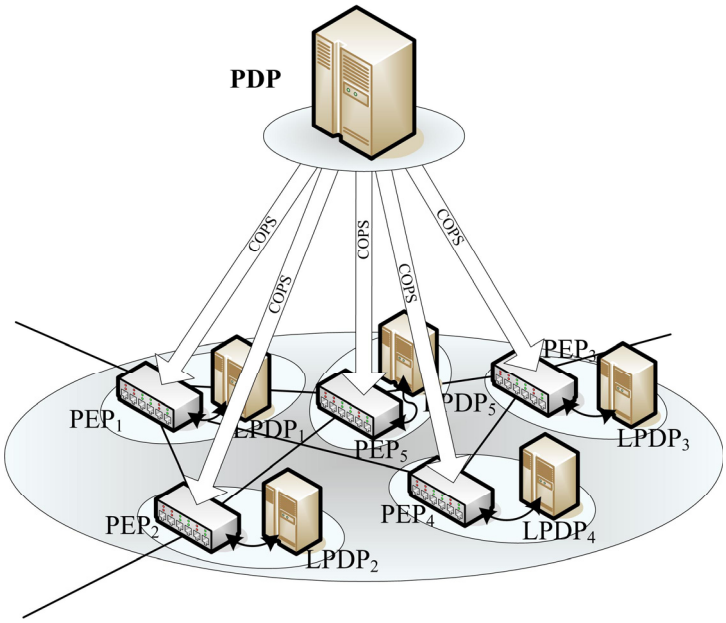


Figure 2-4 : PBN avec LPDP

Cette entité logique n'a, *a priori*[1], pas accès aux serveurs complémentaires. Elle possède cependant une copie partielle du *policy repository* pour prendre ses décisions. Ces dernières ne sont pas basées, comme c'est le cas avec le PDP, sur des informations globales puisque le LPDP n'a pas accès au *bandwidth broker* par exemple, mais uniquement sur des informations locales.

Lorsqu'un nœud est équipé d'un LPDP, il interroge d'abord celui-ci. Après avoir récupérer la réponse du LPDP, deux possibilités peuvent se présenter :

- Soit le PDP est injoignable à cause d'une panne. Le PEP applique alors cette décision.

- Soit le réseau fonctionne correctement. Dans ce cas, le PEP envoie au PDP la décision du LPDP accompagnée de la question d'origine. Le PDP prend en compte cette décision et d'autres informations globales puis retourne la décision finale prévalant sur celle du LPDP.

L'intérêt du LPDP est double :

- En temps normal, on peut imaginer que le LPDP dispose d'informations locales inaccessibles au PDP. C'est pourquoi le PDP doit tenir compte de la décision du LPDP même s'il n'est pas tenu de la suivre.
- Lorsque le PEP ne peut plus joindre de PDP, son LPDP lui permet de continuer à fonctionner de façon autonome au moins pendant un certain temps. Cela suppose cependant que les politiques que le LPDP a en mémoire sont à jour par rapport à celles du PDP. C'est d'ailleurs un autre intérêt de toujours passer par le LPDP même lorsque le réseau fonctionne : cela maintient une certaine synchronisation entre le PDP et le LPDP.

2.1.3 Les PEPs (Policy Enforcement Point)

Le PEP est une entité logique qui met en application les décisions du PDP. En pratique, il y en a un par nœud actif du réseau. Il peut gérer plusieurs interfaces et s'occuper de plusieurs services (QoS, sécurité…).

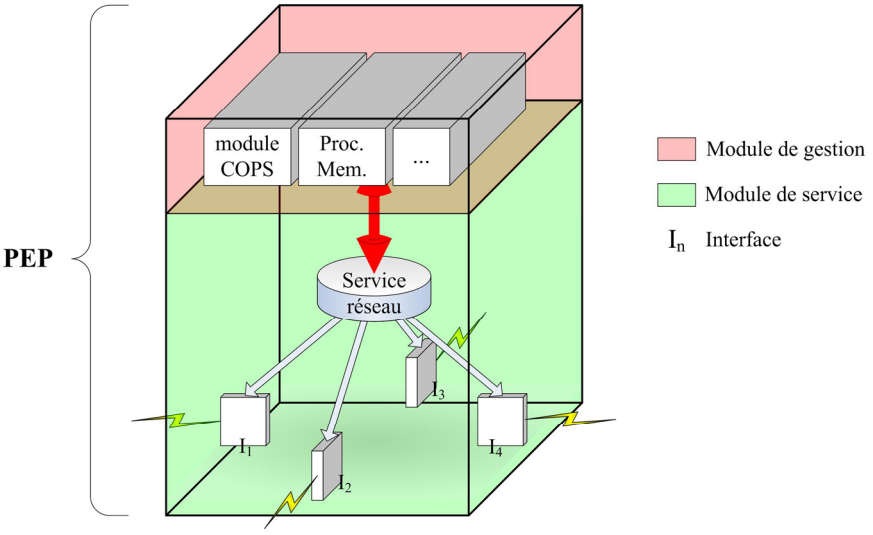


Figure 2-5 : Structure d'un PEP

Les PEPs sont généralement installés sur des routeurs d'accès ou des serveurs du réseau. Afin d'appliquer les règles de politique, suivant le matériel auquel ils sont associés, ils peuvent faire du filtrage, marquer les paquets, contrôler le débit, remettre en forme les flux…

Comme indiqué dans Figure 2-2, le PEP communique avec le PDP par le protocole COPS (Common Open Policy Service) ou une de ses extensions.

[1] En réalité, aucun document que j'ai lu ne permet réellement d'affirmer ou non ce fait. Ce qu'on peut dire c'est que tout laisse à penser que les LPDP n'ont pas pour rôle d'accéder directement aux serveurs supplémentaires, mais rien n'interdit de leur donner ce pouvoir.

2.2 Autres modèles

2.2.1 Le modèle deux-tiers

Dans le modèle trois-tiers (Figure 2-2), les trois composantes principales sont constituées du PDP, du PEP et du *policy repository*. Dans ce modèle, les deux composants principaux sont le *policy repository* et les clients de politiques (*policy client*) qui regroupent PEP et PDP en un seul nœud. Ce choix est motivé par le fait que certains éléments du réseau sont suffisamment souples et intelligents pour prendre leurs propres décisions de politique et les appliquer. Typiquement ces éléments de réseau sont des serveurs ou des routeurs possédant un processeur dédié aux opérations de contrôle.

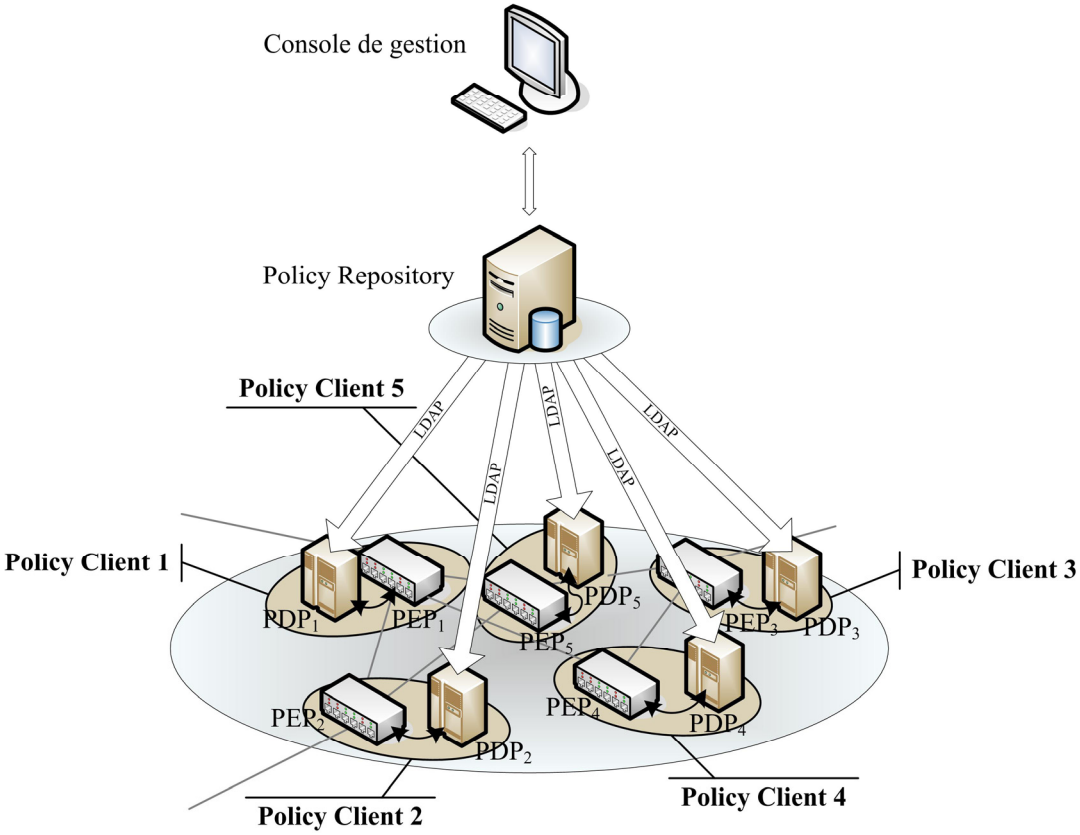


Figure 2-6 : Structure d'un PBN (modèle deux-tiers)

Cette architecture offre de multiples avantages :

- Elle améliore la réactivité des nœuds actifs du réseau. En effet, le jeu de question-réponse avec un PDP central peut engendrer des délais assez coûteux notamment en cas de surcharge du PDP ou du réseau.
- Elle décentralise l'entité de prise de décision. Cette décentralisation améliore grandement le passage à l'échelle du dispositif et peut même être nécessaire pour des raisons stratégiques.

Pourtant le modèle deux-tiers est rarement évoqué car il présente aussi quelques inconvénients :

- Conserver une vision globale et cohérente du réseau dans tous les PDPs est très complexe.
- Le PDP étant un dispositif sensiblement plus complexe que le PEP, sa multiplication engendre un surcoût matériel non négligeable.

2.2.2 Modèle hybride

Signalons brièvement qu'il n'est pas obligatoire que tous les nœuds actifs d'un réseau fonctionnent de la même manière. Il est possible d'imaginer des réseaux où les nœuds ayant des ressources limitées communiquent avec un PDP central et les autres prennent leur propre décision.

Figure 2-7 donne un exemple de ce à quoi peut ressembler un tel PBN.

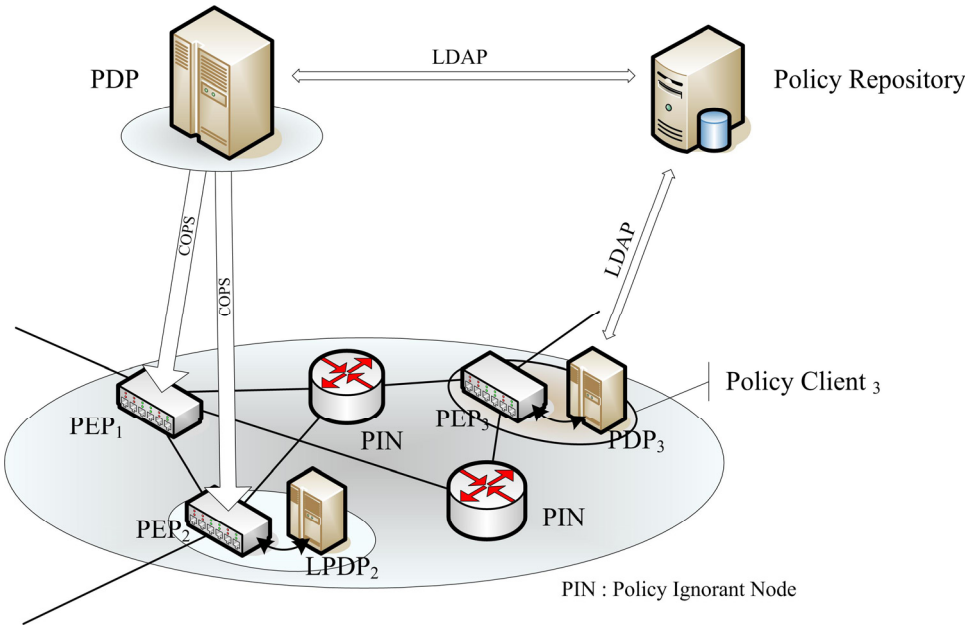


Figure 2-7 : Exemple de PBN hétéroclite

Il est évident qu'une telle hétérogénéité dans un réseau est quelque peu exagérée et qu'elle peut poser des difficultés dans sa mise en place, cependant elle est tout de même envisageable.

2.2.3 Modèle évolué

Les deux modèles vu précédemment ont chacun leurs défauts : faibles performances pour le modèle trois-tiers et complexité de mise en oeuvre pour le modèle deux-tiers. Il ont également un défaut commun : faible résistance aux pannes. Donc ces modèles ne répondent pas aux contraintes fixées dans la section 1.2 page 20.

Ces modèles sont en effet trop simples : pour répondre aux besoins de robustesse face aux pannes et de passage à l'échelle, il faut pratiquer judicieusement la redondance ce qu'aucun des modèles ne fait.

Deux entités en particulier doivent être dupliquées et /ou partitionnées : le PDP et le *policy repository* car il s'agit d'éléments indispensables au bon fonctionnement du réseau et dont la charge croit au moins linéairement avec la taille du réseau (probablement plus en fonction des services offerts).

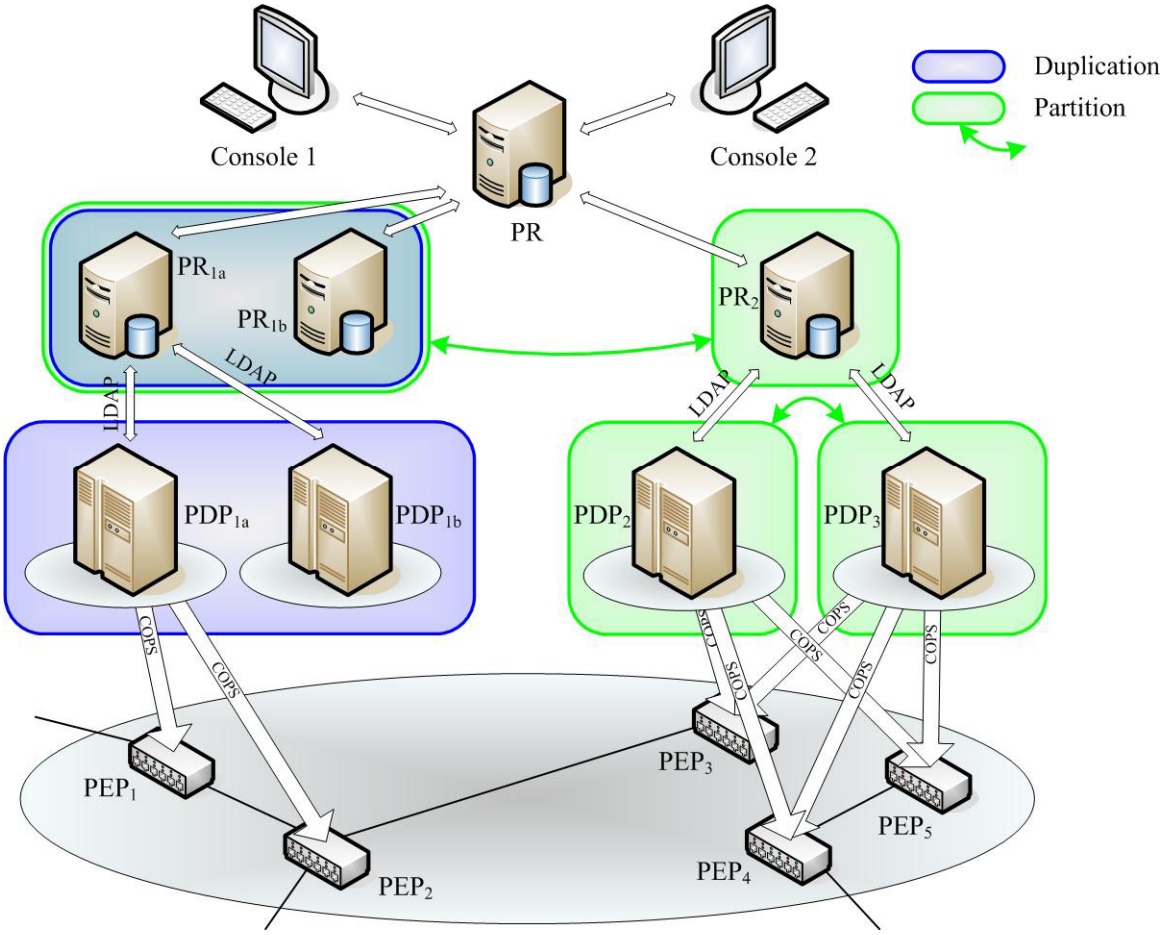


Figure 2-8 : Exemple d'un PBN évolué

L'exemple de réseau ci-dessus comprend des duplications et des partitions de PDPs et de *policy repositories* :

- PR_{1b} est la réplique exacte de PR_{1a} au cas où celui-ci tomberait en panne.
- Les politiques de PR sont réparties entre PR_{1a} et PR_2 parce que certaines ne concernent que PEP_1 et PEP_2 alors que les autres ne concernent que PEP_3, PEP_4 et PEP_5.
- PDP_{1b} est une PDP de secours au cas où PDP_{1a} tomberait en panne.
- PDP_2 et PDP_3 forment sorte de partition des services, par exemple parce que PDP_2 s'occupe de la QoS tandis que PDP_3 se charge de la sécurité.

D'autres part, sur ce même exemple, deux consoles de gestion sont représentées. La duplication des consoles de gestion n'apporte pas de gain de performance, mais elle est indispensable dans un grand réseau. Dans ce cas, gérer les conflits entre les règles n'est plus optionnel mais nécessaire (et difficile).

Pour finir, signalons que si ce modèle évolué est réellement performant, l'optimisation de la redondance des PDPs et autres serveurs auxiliaires est encore un domaine de recherche actif.

3 Le protocole COPS (Common Open Policy Service)

Le groupe de travail RAP de l'IETF (celui qui a défini le cadre de travail pour le PBNM [6]) a conçu le protocole COPS pour les communications entre le PEP et le PDP après avoir constaté que les protocoles de gestion de réseau traditionnels, comme par exemple SNMP[1], étaient incapables de s'intégrer complètement dans un PBN. COPS a l'avantage d'être parfaitement compatible avec le cadre de travail du PBNM mais il n'est pas interdit d'utiliser un autre protocole.

Très récemment d'ailleurs un nouveau groupe de travail est apparu au sein de l'IETF, le Network Configuration WG, pour mettre en place un protocole de configuration basé sur XML : Netconf. Ce protocole qui n'est encore qu'à l'état de draft [24], pourrait, à terme, remplacer COPS d'autant plus que certains acteurs importants du monde des réseaux, en particulier Cisco, sont très peu favorable au déploiement de COPS.

En dépit de ce sombre avenir (certains annoncent la mort de COPS depuis plus d'un an) ce document ne présente que COPS car il s'agit du protocole le plus abouti et qu'il existe des implémentations, en licence libre qui plus est, de COPS ce qui n'est pas le cas de NetConf.

3.1 Les caractéristiques principales

3.1.1 Les objectifs de COPS

L'objectif principal de COPS est d'être simple et extensif. Il doit permettre de mettre en place des services qui n'existent pas encore.

Ses principales caractéristiques sont :

- Le protocole emploie un modèle de type client/serveur où le PEP (le client) envoie des requêtes, des mises à jour et des suppressions au PDP (le serveur) et le PDP lui retourne des décisions.
- Le protocole utilise TCP comme protocole de transport afin de fiabiliser les échanges de messages entre le PEP et le PDP.
- Le protocole est extensible dans la mesure où il prend en compte des objets contenant leur propre définition. Il a été créé pour l'administration, la configuration et l'application de règles de politique. C'est un protocole générique dont les définitions peuvent être étendues dans un contexte d'application particulier (la sécurité, la QoS avec IntServ, la QoS avec DiffServ…)
- Le protocole fournit une sécurisation des messages : authentification, protection contre le rejoue et contrôle d'intégrité. Il peut également s'appuyer sur d'autres protocoles comme IPSEC ou TLS (Transaction Layer Security) pour sécuriser le canal entre le PEP et le PDP.

[1] A l'heure actuelle, SNMP est, malgré tout, plus souvent utilisé que COPS, probablement car il est plus mature, mieux maîtrisé et offre certaines fonctions similaires à COPS. Cependant, il faut ajouter que beaucoup de produits commerciaux (de Intel, Cisco…) proposant COPS existent déjà.

- Le protocole est *stateful*. C'est à dire qu'il mémorise des informations d'états de configuration. Ces états représentent un ensemble (qui peut être vide) de règles que le PEP doit appliquer. Le protocole les gère de sorte que :
  - Les états sont partagés entre le PDP et le PEP tant que celui-ci ne les a pas explicitement effacés.
  - Le PDP peut répondre à une nouvelle requête différemment en fonction d'états déjà établis relatifs avec cette requête.
  - Le PDP peut mettre en place dans le client des informations de configuration et les effacer quand elles ne sont plus applicables.

3.1.2 Les extensions de COPS

3.1.2.1 Les modèles de gestion par politiques

Il existe à l'heure actuelle deux modèles de gestion de politiques qui ont fait l'objet de beaucoup de recherche : l'*outsourcing* que l'on peut traduire par modèle par « externalisation » et le *provisioning* que l'on peut traduire par modèle par « approvisionnement » ou encore par « configuration ».

3.1.2.1.1 Outsourcing

L'*outsourcing* est le principal modèle de gestion par politiques, en ce sens que le PBNM a d'abord été envisagé sous cet angle. Dans ce modèle, le PEP externalise toutes ses décisions vers le PDP, c'est à dire qu'il ne prend jamais de décision sans s'être référé au PDP.

Cela est parfaitement adapté à des réseaux utilisant des protocoles de signalisation comme RSVP. Lorsqu'un utilisateur extérieur au réseau souhaite réserver un chemin à travers ce réseau, il envoie un message RSVP à un nœud d'entrée du réseau spécifiant ses besoins. Si ce nœud est un PEP, il intercepte le message et demande au PDP s'il doit accepter la réservation ou non. Quelque soit la décision du PDP, il est tenu de l'appliquer.

Ce modèle est parfois qualifié de mode « tiré » ou « réactif », dans la mesure où le PEP « tire » les décisions du PDP et le PDP « réagit » aux évènements du PEP.

3.1.2.1.2 Provisioning

Dans le modèle de *provisioning*, le PEP est « approvisionné » par le PDP en règles de politiques qui lui permettront de prendre tout seul des décisions. En retour le PEP peut faire des rapports sur les décisions qu'il applique par exemple pour mettre en place un service de facturation.

Le *provisioning* est donc fondamentalement différent de l'*outsourcing*. Il est particulièrement bien adapté à des réseaux non fondés sur des protocoles de signalisation, comme des réseaux offrant une QoS par DiffServ.

3.1.2.2 Les extensions

Ce document n'entrera pas dans les détails des extensions de COPS. Cependant, il faut signaler qu'il existe pour chacun des modèles de gestions de politiques décrits ci-dessus une extension de COPS normalisée dans un ou plusieurs RFCs (Request For Comment).

3.1.2.2.1 COPS-RSVP (COPS for RSVP)

COPS-RSVP [14] est une extension de COPS adaptée au modèle de l'*outsourcing* et plus particulièrement au réseaux utilisant RSVP [12] comme protocole de signalisation. La principale amélioration de COPS-RSVP est de pouvoir intégrer dans les messages COPS des objets RSVP entiers.

3.1.2.2.2 COPS-PR (COPS for PRovisioning)

COPS-PR [17] est une extension de COPS adaptée au modèle du *provisioning*. Il ne fournit pas de mécanisme de gestion de protocole de signalisation mais permet le transport de règles de politiques de tout type. Ces règles sont exprimées dans une structure, compréhensible par tout matériel, appelée SPPI (Structure of Policy Provisioning Information) [18] et sont stockées dans une PIB (Policy Information Base) [19], sorte de base de données de politiques chez le PEP et le PDP.

Il faut noter qu'il est aussi envisagé [20] d'utiliser des PIBs dans le modèle de l'*outsourcing*. Le transport des informations destinées à ces PIBs se ferait alors via COPS-PR utilisé parallèlement à COPS-RSVP.

3.2 Les messages

COPS n'est, a priori, pas utilisable immédiatement. Il est nécessaire de l'étendre pour l'utiliser dans un PBN (COPS-PR, COPS-RSVP et COPS-ODRA sont des exemples d'extensions). Il fournit néanmoins les messages et les objets minimaux pour assurer complètement le dialogue entre le PEP et le PDP.

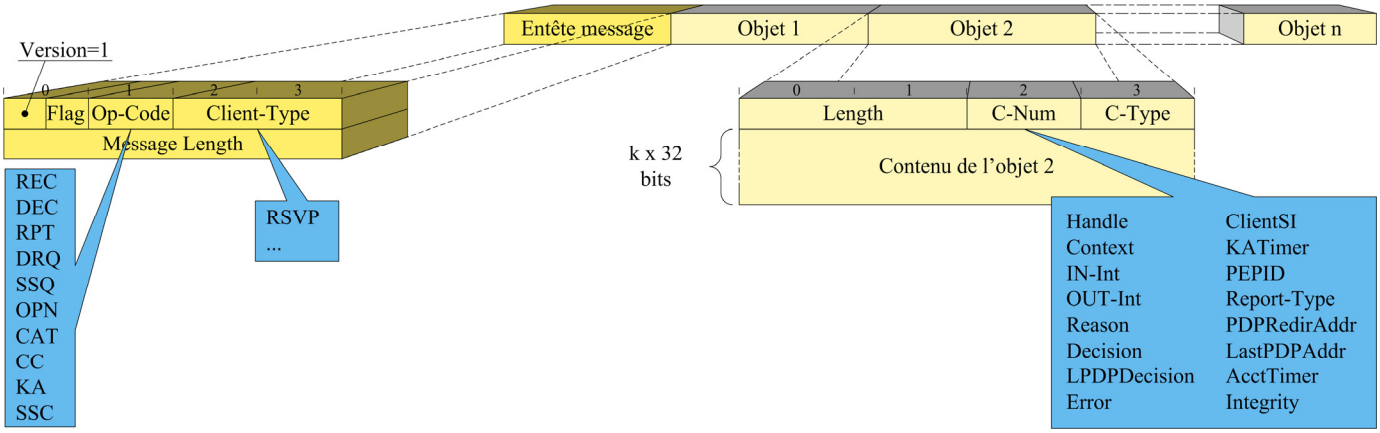


Figure 2-9 : Un message COPS

Tous les messages de COPS (et de ses extensions) ont la même structure définie dans Figure 2-9. Les sections suivantes détaillent brièvement les différents objets et messages existants.

3.2.1 Les objets

Les objets sont des briques de $k \times 32$ bits ($k \in \aleph$) qui constituent le contenu d'un message et lui donnent un sens. Extrait de Figure 2-9, Figure 2-10 donne l'entête commun à tous les objets de COPS.

3.2.1.1 Entête

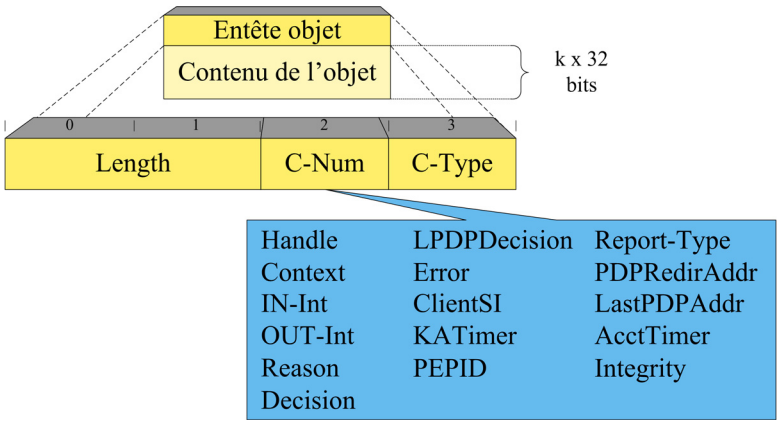


Figure 2-10 : Entête d'un objet COPS

Length est une valeur de deux octets qui décrit le nombre d'octets (y compris l'entête) composant l'objet. Si cette longueur n'est pas un multiple de 4 octets alors il est obligatoire de rajouter à la fin de l'objet suffisamment de bits à 0 pour atteindre la bonne taille (cependant ces bits de bourrage ne sont pas pris en compte dans Length).

C-Num identifie le type d'information contenue dans l'objet (voir plus bas).

C-Type dépend de C-Num.

3.2.1.2 Types d'objets

Il existe en tout 16 types d'objets COPS :

- Handle : Identifie de manière unique un état installé. Pour faire simple, disons qu'un état correspond à une décision 'initiale' du PDP. Ainsi, tous les messages échangés par la suite concernant cette décision utilise le même Handle.
- Context : Indique le type d'événement qui a déclenché la requête (arrivée/sortie d'un message, besoin d'allocation de ressources, besoin de configuration).
- IN-Int (IN-Interface) : Indique l'interface d'entrée concernée par le message.
- OUT-Int (OUT-Interface) : Indique l'interface de sortie concernée par le message.
- Reason : Indique la raison pour laquelle un état n'est plus applicable par le PEP émetteur.
- Decision : Contient la décision prise par le PDP. Une décision peut être répartie sur plusieurs objets Decision.
- LPDPDecision : Contient la décision prise par le LPDP pour aider le PDP à prendre sa décision finale.
- Error : Identifie une erreur survenue dans un PDP ou un PEP
- ClientSI (Client Specific Information) : Contient des informations spécifique à un *client-type*. Dans la terminologie COPS, un *client-type* désigne un service (QoS par DiffServ, QoS par IntServ, sécurité…) tandis qu'un *client* désigne un PEP (par référence à l'architecture client/serveur des PBNs).
- KATimer (Keep-Alive Timer) : Indique l'intervalle maximum entre l'émission ou la réception de deux messages COPS. Au-delà de cet intervalle, la connexion entre le PEP et le PDP est considérée comme interrompue.

- PEPID (PEP Identification) : Identifie, de manière unique au sein d'un domaine de politiques, l'identité du PEP auquel le message s'adresse.
- Report-Type : Indique la nature d'un rapport d'un PEP (succès, échec, comptabilité).
- PDPRedirAddr (PDP Redirect Address) : Indique au PEP récepteur un PDP alternative (car l'actuel PDP ne peut pas le prendre en charge).
- LastPDPAddr (Last PDP Address) : Contient l'adresse du dernier PDP auquel le PEP émetteur a été connecté. Cette information peut être utile après une panne.
- AcctTimer (Accounting Timer) : Indique l'intervalle minimum entre deux rapports de comptabilité du même PEP pour éviter de surcharger le réseau inutilement.
- Integrity : Authentifie le message COPS et garantie son intégrité.

Les objets les plus utiles pour étendre COPS, et donc créer un nouveau service, sont les objets Decision et ClientSI. En effet, les extensions de COPS ne définissent a priori pas de nouveaux objets. Cela n'est pas strictement interdit, mais c'est plutôt inutile car il suffit généralement d'introduire de nouveaux *sous*-objets aux objets Decision et/ou ClientSI pour ajouter de nouvelles fonctionnalités (voir 3.2.4 du chapitre 3). C'est ce qui a été fait dans COPS-PR, COPS-RSVP et dans l'extension que j'ai mise en place pendant ce stage.

3.2.2 Les messages

Les messages COPS sont constitués d'un entête (Figure 2-11) et d'un certain nombre d'objets définis ci-dessus. L'ordre des objets constituant le message n'est pas imposé excepté pour l'objet Integrety qui doit toujours se trouver à la fin.

3.2.2.1 Entête

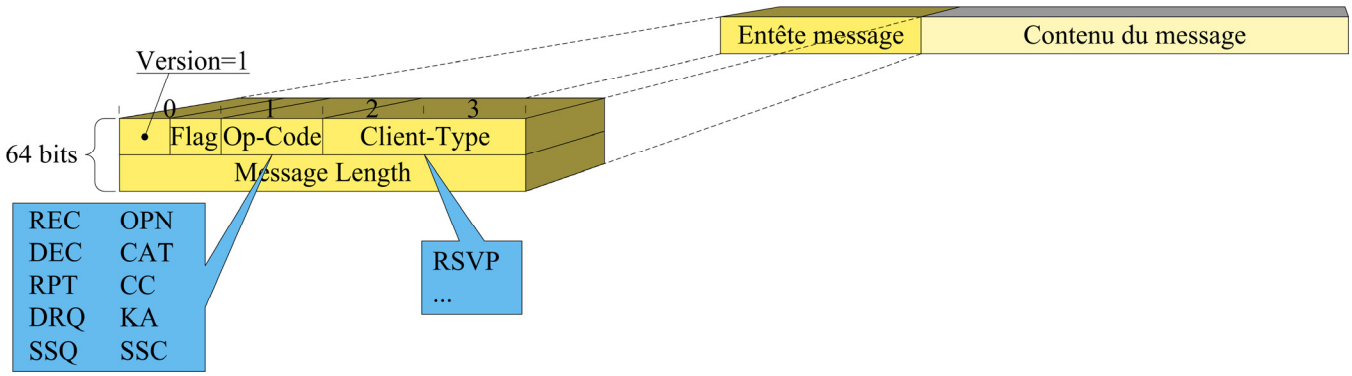


Figure 2-11 : Entête d'un message COPS

L'entête est composé de cinq champs :

- Version : Numéro de version de COPS (actuellement Version=1).
- Flags : Ce champ est mis à 0x1 si ce message a été sollicité par un autre message COPS. Autrement il doit être laissé à 0x0.
- Op Code : Type du message (voir plus bas).
- Client-Type : Identifie le *client-type* auquel le message est destiné. Aujourd'hui seul le *client-type* RSVP est normalisé (sa valeur est 0x1). Cependant, les numéros 0x8000 à 0xFFFF peuvent être

utilisés librement pour des implémentations propriétaires de COPS afin de répondre à des besoins très spécifiques.

- Message Length : Taille en octets du message. Cette longueur comprend l'entête du message et les objets qu'il contient. Les éventuels bits de bourrage sont aussi comptés (pour rappel, ils ne sont pas pris en compte dans le champ Length de l'entête des objets).

3.2.2.2 Types de messages

On peut classer les dix types de messages en fonction de leur utilisation.

3.2.2.2.1 Gestion des états

La gestion des états (installation, suppression et modification des politiques) est réalisée à l'aide de quatre types de messages :

- Request (REQ) PEP → PDP : Requête du PEP pour savoir comment réagir face à une situation précise.
- Decision (DEC) PDP → PEP : Décision du PDP, généralement en réponse à une requête REQ, indiquant au PEP ce qu'il doit faire.
- Report State (RPT) PEP → PDP : Rapport du PEP indiquant si la décision du PDP a pu être mise en place.
- Delete Request State (DRQ) PEP → PDP : Message indiquant au PDP que le PEP émetteur a du supprimer un état installé (par exemple à cause d'une panne d'interface).

3.2.2.2.2 Synchronisation des états

Les messages précédents sont suffisants pour gérer les états tant qu'aucune défaillance (panne réseau, anomalie dans le fonctionnement du PDP ou du PEP…) ne survient. Deux messages supplémentaires sont nécessaires pour récupérer après une erreur :

- Synchronize State Request (SSQ) PDP → PEP : Demande du PDP de synchronisation des états. Le PEP envoie ses états à synchroniser dans des messages REQ.
- Synchronize State Complete (SSC) PEP → PDP : Message indiquant au PDP que la synchronisation qu'il a demandée est terminée.

3.2.2.2.3 Gestion de la session

Tous les messages précédents sont échangés au sein d'une session. Une session est un espace de dialogue, de niveau applicatif, entre un PEP et un PDP. Un PEP doit ouvrir autant de sessions qu'il supporte de *client-types*, mais il n'est pas nécessaire que toutes ces sessions soient ouvertes avec le même PDP.

La gestion des sessions requière quatre messages spécifiques :

- Client-Open (OPN) PEP → PDP : Demande d'ouverture de session pour un *client-type* donné.
- Client-Accept (CAT) PDP → PEP : Message indiquant au PEP que sa demande d'ouverture de session a été acceptée.
- Client-Close (CC) PEP ↔ PDP : Fermeture de session pour in *client-type* donné. Si ce message vient du PDP, celui-ci peut préciser au PEP à quel autre PDP il peut se reconnecter.

- Keep-Alive (KA) PEP ↔ PDP : Message permettent de s'assurer que la connexion TCP, supportant une ou plusieurs sessions, fonctionne correctement.

3.2.2.3 Relations entre les messages et les objets

Figure 2-12 montre les relations entre messages et objets COPS. Du point de vue des messages, on peut voir de quel(s) objet(s) ils sont constitués. Du point de vue des objets, on peut voir dans quel(s) message(s) ils apparaissent.

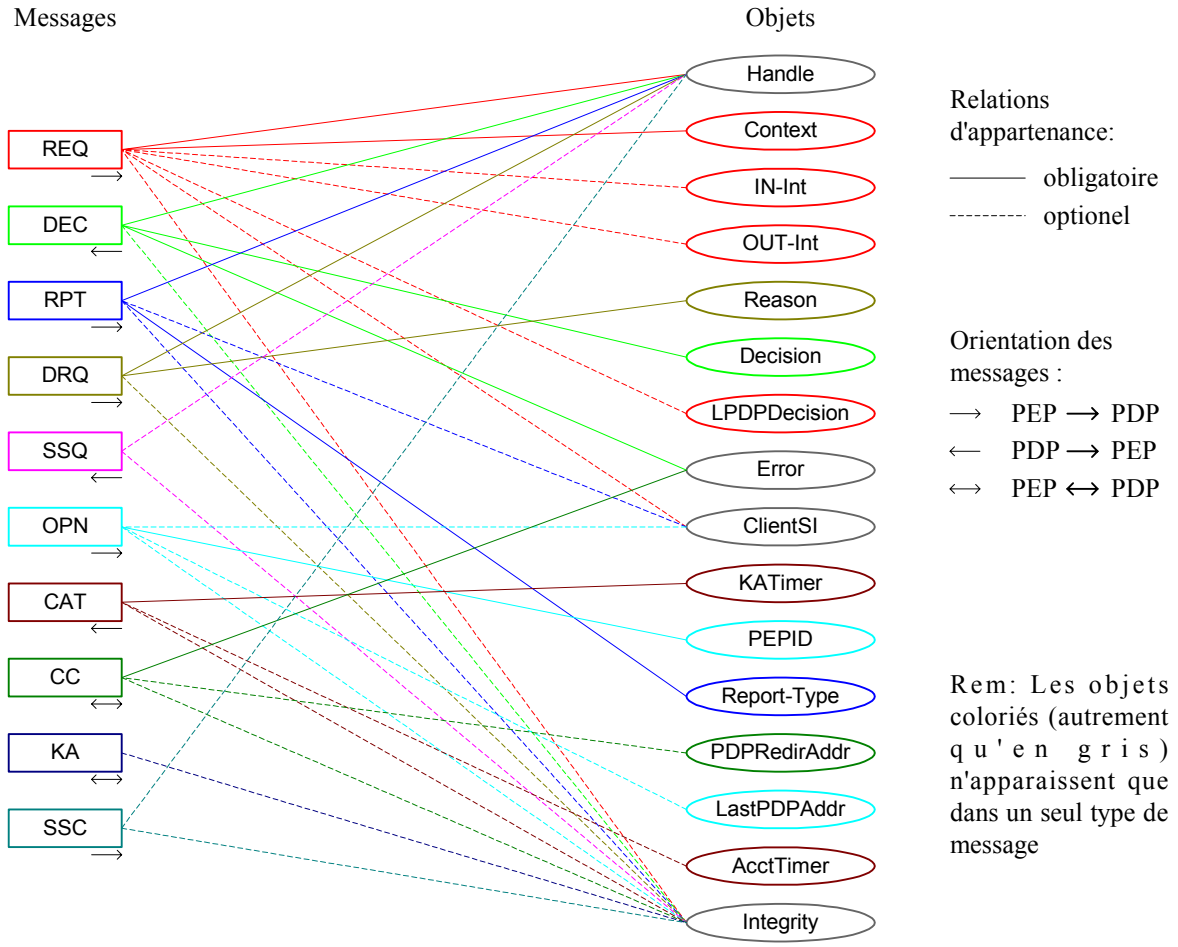


Figure 2-12 : Relations entre messages et objets COPS

Pour avoir des informations plus précises sur la manière dont les messages sont constitués, se reporter à [8].

3.3 Le déroulement d'une session

Après avoir listé tous les messages COPS, il est intéressant de savoir comment ils sont utilisés pendant le déroulement d'une session.

3.3.1 L'initialisation

L'initialisation consiste toujours à ouvrir un canal par *Client-type* entre le PEP et le PDP. La sécurisation de ce canal est optionnelle et exige une étape supplémentaire.

3.3.1.1 Ouverture d'un canal client/serveur

♦ *Connexion TCP*

COPS est basé sur TCP. Donc avant d'ouvrir une session COPS, il faut d'abord ouvrir une connexion TCP comme le montre Figure 2-13.

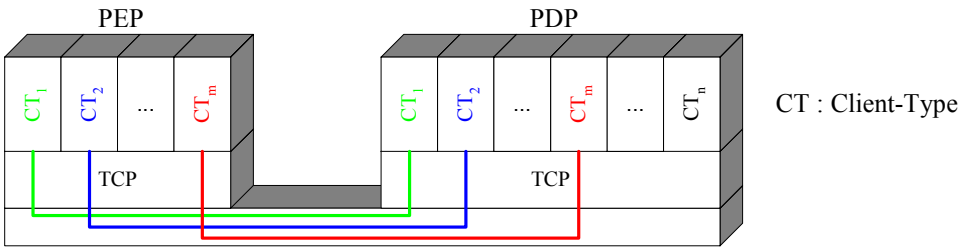


Figure 2-13 : Sessions COPS et connexions TCP

Le PEP est responsable de l'initialisation de la connexion TCP. Les PDPs doivent toujours écouter un port défini à l'avance. L'IANA (Internet Assigned Numbers Authority) a réservé le port 3288 à COPS, cependant il n'est pas interdit d'en utiliser d'autres par exemple pour faire tourner deux PDPs sur la même interface.

La localisation du PDP peut être configurée manuellement ou obtenue dynamiquement par un mécanisme de localisation [11]. Cependant, aucune méthode de découverte des PDPs n'est définie dans COPS.

♦ *Client-Types*

Une fois la connexion TCP établie et éventuellement après que la sécurité ait été négociée (voir la section suivante), le PEP ouvre autant de sessions qu'il supporte de *Client-types*. Cela se fait par l'intermédiaire des messages OPN. Le PEP envoie un message OPN pour chaque *Client-type*.

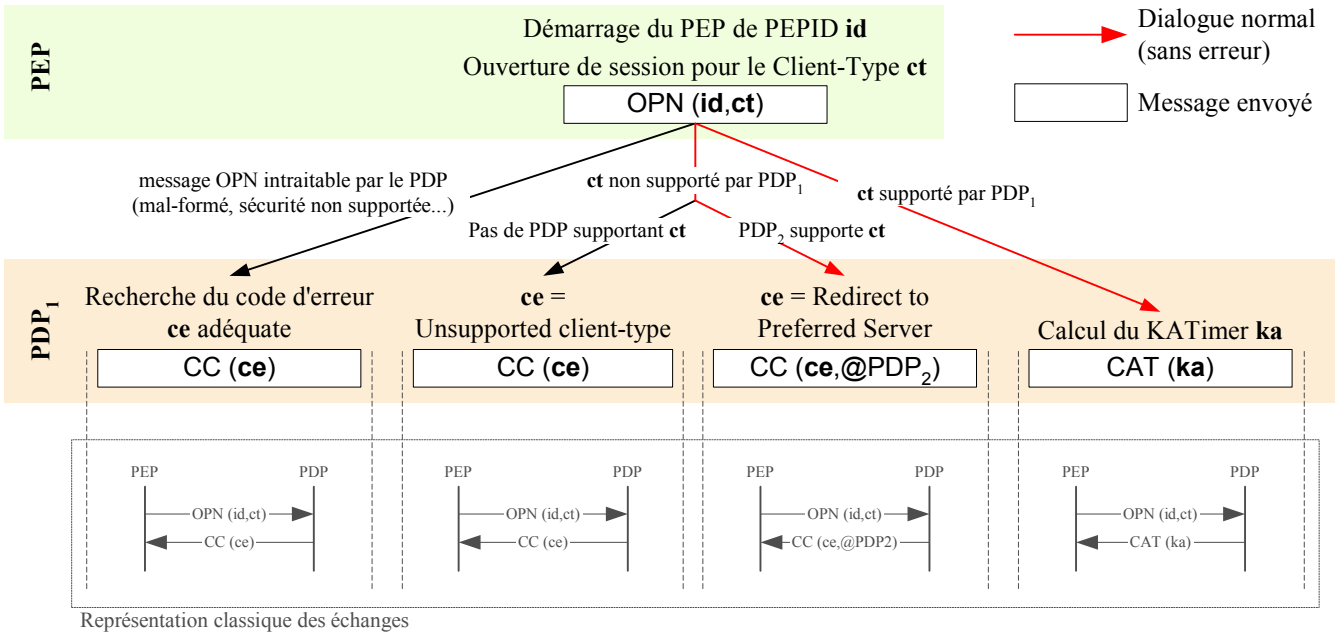


Figure 2-14 : Echanges COPS : Ouverture de session

Pour chaque message OPN, s'il est bien formé et que le PDP supporte ce *Client-type*, le PDP retourne au PEP un message CAT signifiant qu'il accepte l'ouverture de la session. Dans le cas contraire, il retourne un

message CC en indiquant éventuellement l'adresse d'un autre PDP qui pourrait répondre favorablement au message OPN.

3.3.1.2 Sécurité

COPS intègre un mécanisme pour sécuriser les sessions à travers les objets Integrity qui permettent l'authentification, la vérification de l'intégrité des messages et empêchent les attaques par rejoue. Bien que toutes les entités COPS doivent supporter les objets Integrity, l'utilisation de la sécurité de COPS est facultative. Il est tout à fait possible d'utiliser d'autres mécanismes de sécurité, éventuellement à des niveaux inférieurs.

Pour ne pas utiliser la sécurité de COPS, il suffit de ne rien faire : ne pas initialiser les mécanismes de sécurité et ne pas utiliser les objets Integrity.

♦ *Sécurité et connexion TCP*

Dans le cas où le PEP et le PDP veulent utiliser la sécurité de COPS, ils doivent impérativement la mettre en place avant d'ouvrir la première session, c'est à dire lors du premier échange de messages OPN et CAT. En effet, la sécurité est négociée une seule fois pour chaque connexion TCP et protège toutes les sessions qui seront mises en place au-dessus de cette connexion. Si le PDP reçoit une demande de sécurisation (message OPN avec Client-Type=0) alors qu'au moins une session a déjà été établie, il doit refuser cette demande (envoi d'un message CC avec Client-Type=0 au PEP).

Puisque la sécurité est établie sur une connexion TCP donnée et non sur une session particulière, les messages relatifs à la sécurité ont toujours un champ Client-Type valant 0.

Une fois que la sécurité est mise en place, tous les messages échangés sur cette connexion TCP doivent comporter un objet Integrity (en dernière position).

♦ *Négociation des numéros de séquence*

Les numéros de séquence servent à éviter les attaques par rejoue.

Leur utilisation est assez simple. Chaque entité (PEP et PDP) gère son propre numéro de séquence évoluant indépendamment du numéro de séquence de l'autre entité. Pour une entité donnée, une fois le numéro initial défini, il suffit d'incrémenter de un ce numéro à chaque message envoyé par cette entité. Lorsque le numéro a atteint sa valeur maximale (OxFFFFFFFF), le prochain incrément le ramène à 0 puis il continue à être incrémenté normalement.

Pour qu'un numéro de séquence soit efficace, il est indispensable que sa valeur initiale soit choisie aléatoirement par l'entité qui va l'incrémenter. Ces valeurs initiales sont communiquées lors de l'échange de messages OPN et CAT comme le montre Figure 2-15.

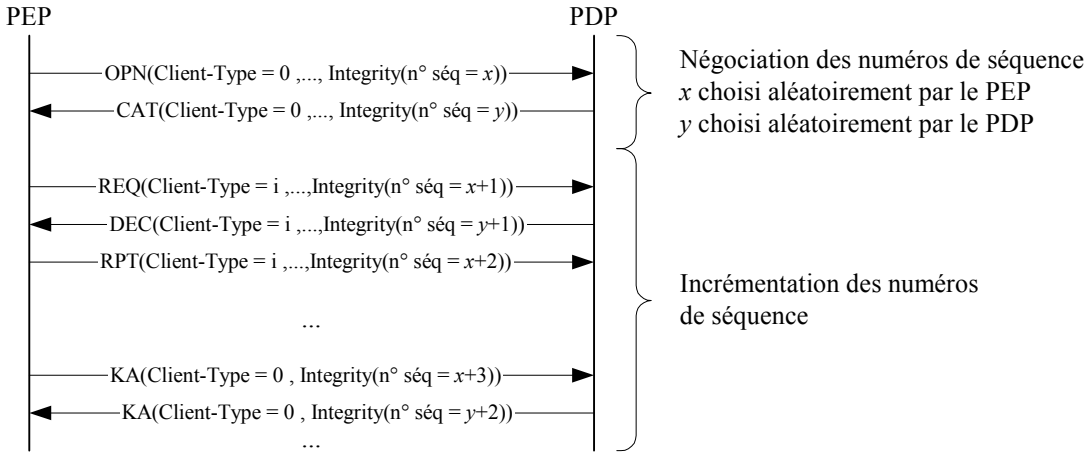


Figure 2-15 : Echanges COPS : Négociation des numéros de séquence

♦ *Maintenance des clefs et des fonctions de hachage*

Afin de calculer le Digest de l'objet Integrity, on utilise une fonction de hachage φ associée à une certaine clef χ. L'entité qui reçoit le message contenant l'objet Integrity doit connaître ce couple (φ,χ) pour pouvoir elle-même calculer le Digest et vérifier ainsi l'authenticité du message et son intégrité. Il est donc nécessaire que le PEP et le PDP partagent le couple (φ,χ) et que celui-ci ne puisse pas être découvert par une entité malveillante. Pour éviter les attaques par force brute, il est également nécessaire de changer régulièrement la clef et même éventuellement l'algorithme de hachage utilisé. Les deux entités partagent donc, non pas un couple (φ,χ), mais un ensemble $\{(\varphi_i,\chi_i) \mid i\in[1,n]\}$ de différentes clefs et fonctions de hachage. Il est alors nécessaire de pouvoir indiquer dans les objets Integrity le couple (φ_k,χ_k) permettant de calculer le Digest. Pour cela, le PEP et le PDP mettent en place une liste commune (connue d'eux seuls) associant à chaque couple (φ_i,χ_i) un identificateur λ_i. C'est cet identificateur λ_i qui est placé dans le champ Key ID de l'objet Integrity.

Une grande partie de la sécurité de COPS repose sur la gestion des couples (φ_i,χ_i) et des identificateurs λ_i. En effet, si une entité malveillante parvient à connaître les associations $\lambda_i \leftrightarrow(\varphi_i,\chi_i)$, elle peut usurper l'identité d'un PEP, ou plus probablement d'un PDP, et perturber fortement le réseau.

Malheureusement, COPS ne fournit aucun mécanisme pour gérer cela. Tout au plus la norme donne les recommandations suivantes :

- Il faut associer à chaque couple (φ_i,χ_i) une période de validité en dehors de laquelle les messages l'utilisant sont rejetés.
- Le mécanisme de changement de couples (φ_i,χ_i) doit permettre une période de transition pendant laquelle plusieurs couples peuvent être utilisés afin de pouvoir passer en douceur d'un couple (φ_k,χ_k) à un couple (φ_l,χ_l).

♦ *Les échecs*

La négociation de la sécurité peut échouer si l'un des participants désire la mettre en place mais pas l'autre. Dans ce cas, celui qui demandait la sécurité renvoie toujours à l'autre un message CC avec Client-Type = 0.

Une fois que la sécurité a été négociée, les messages reçus contenant un mauvais numéro de séquence, un identificateur λ_i inconnu, un Digest invalide ou n'ayant pas d'objet Integrity entraînent systématiquement l'envoi

d'un message CC de Client-Type=0, contenant un objet Integrity valide et spécifiant le code d'erreur approprié. Bien que non obligatoire, il est aussi recommandé d'interrompre la connexion TCP.

3.3.2 Le fonctionnement normal

Une fois qu'une session a été établie, les principaux messages échangés sont de type REQ (Request), DEC (Decision), RPT (Repport) et DRQ (Delete Request State) afin de configurer le réseau. Occasionnellement, il peut aussi y avoir des messages KA (Keep-Alive), SSQ (Synchronize State Request) et SSC (Synchronize State Complete) pour certaines opérations de maintenance.

Rem : Dans cette section, les échanges de messages sont décrits sans entrer dans les détails de leur contenu propre à chaque modèle de gestion de politiques (*provisioning* ou *outsourcing*).

3.3.2.1 La configuration du réseau

Pour éviter toute confusion, il faut signaler que la section 4.2 présente COPS-PR, une extension de COPS se basant sur le modèle du *provisioning* que l'on traduit en français par modèle d'approvisionnement ou encore de *configuration*. Le terme de configuration aura alors une signification particulière mais dans la section présente, il est utilisé dans une approche très généraliste.

Configurer le réseau consiste ici à modifier certains de ses paramètres afin de contrôler sa manière de réagir à certains évènements (la définition d'un événement dépend du modèle de gestion de politiques utilisé). Dans la terminologie COPS, cela consiste en fait à mettre en place des états et à les faire évoluer.

3.3.2.1.1 Création d'un état

Figure 2-16 indique les échanges entre le PEP et le PDP nécessaires pour mettre en place un état. La figure présente toutes les possibilités qui peuvent survenir pendant le dialogue.

Dans ces échanges, il est intéressant de remarquer les deux réactions possibles du PEP face à des décisions (messages DEC) qu'il ne peut pas appliquer :

- Lorsqu'il s'agit d'un problème protocolaire (message DEC mal-formé ou contenant des objets inconnus), le PEP considère que l'origine du problème vient de l'état **s** (conformément aux notations de Figure 2-16) qui a été installé dans le PDP. Il décide donc de l'effacer (message DRQ) et de libérer le Handle associé. Il pourra par la suite essayer de réinstaller cet état, éventuellement après lui avoir apporter quelques modifications.
- Lorsqu'il s'agit de décisions qu'il ne peut pas mettre en œuvre pour des raisons techniques (manque de ressources par exemple), il se contente d'envoyer un rapport (message RPT) au PDP en précisant éventuellement le problème qu'il a rencontré. Il ne supprime pas l'état puisque a priori celui-ci est valide.

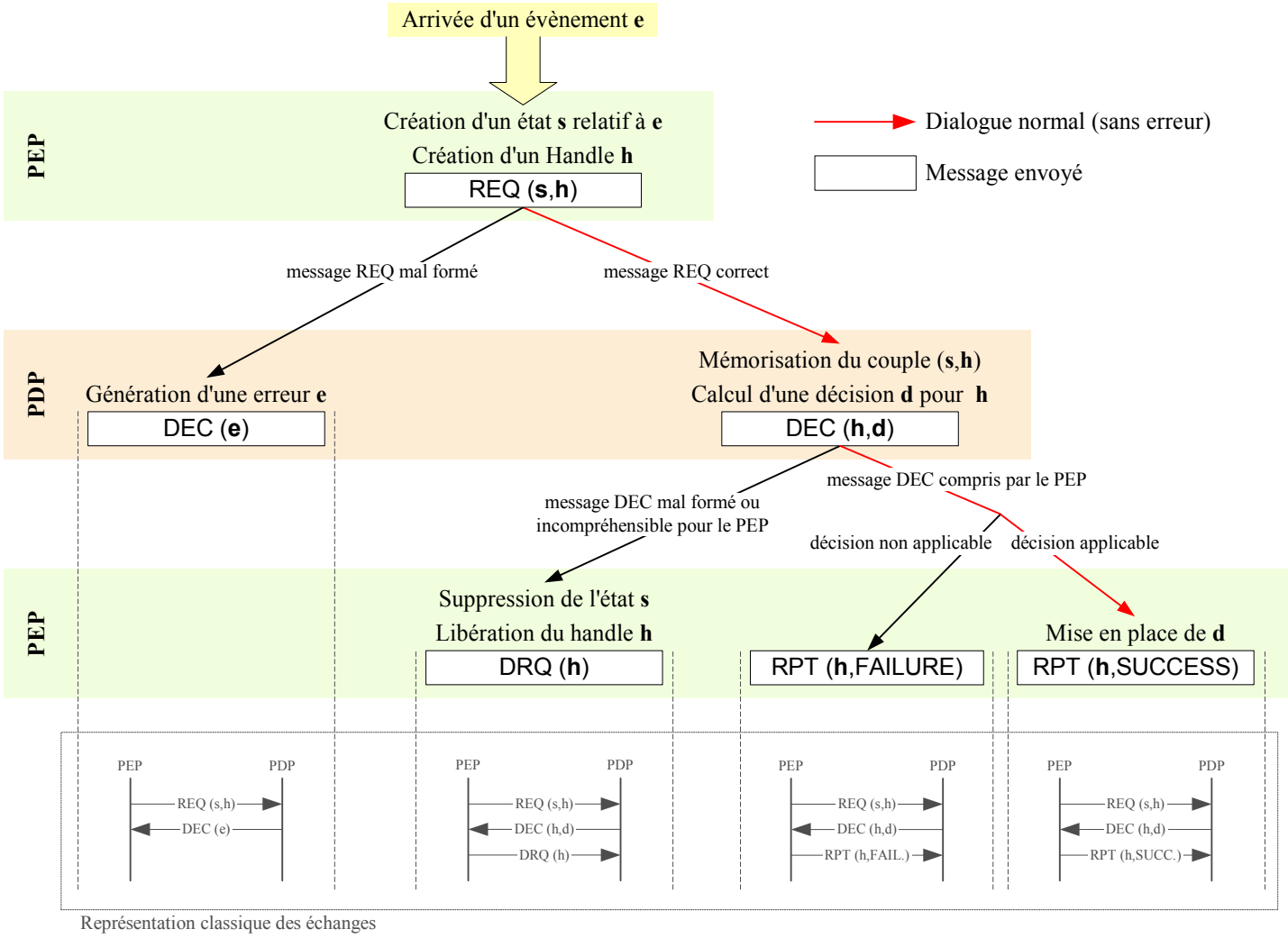


Figure 2-16 : Echanges COPS : Création d'un état

3.3.2.1.2 Modification d'un état

♦ *Par le PDP*

Il est possible que dans certaines conditions, le PDP désire changer une décision prise précédemment. L'évènement provoquant ce changement peut être :

- le déclenchement d'une horloge,
- le fait que le réseau entre dans une situation de congestion,
- une modification manuelle des politiques effectuée par l'administrateur,
- …

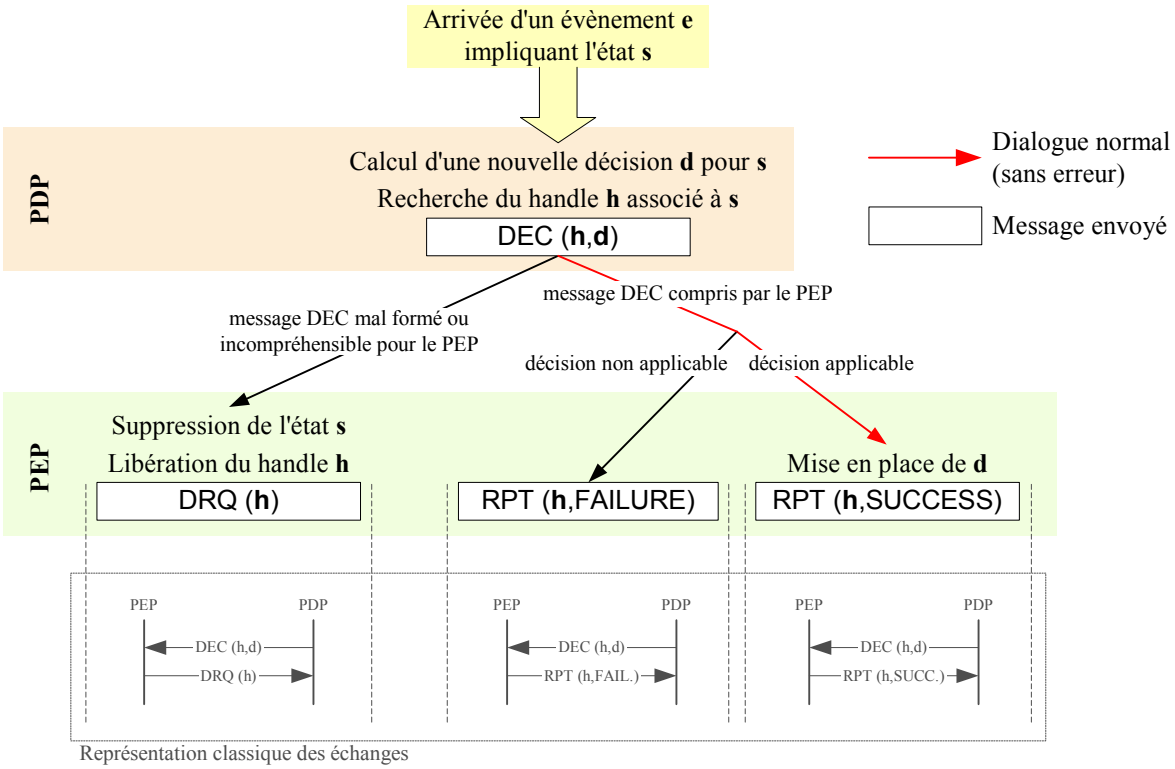


Figure 2-17 : Echanges COPS : Modification d'un état par le PDP

♦ *Par le PEP*

Dans certaines conditions, le PEP doit modifier un état installé afin de refléter une modification et de demander au PDP une nouvelle prise de décision. L'événement provoquant cela peut être :

- une panne sur une interface,
- une extension de ses capacités (par exemple, augmentation de sa mémoire),
- une modification d'un flux
- …

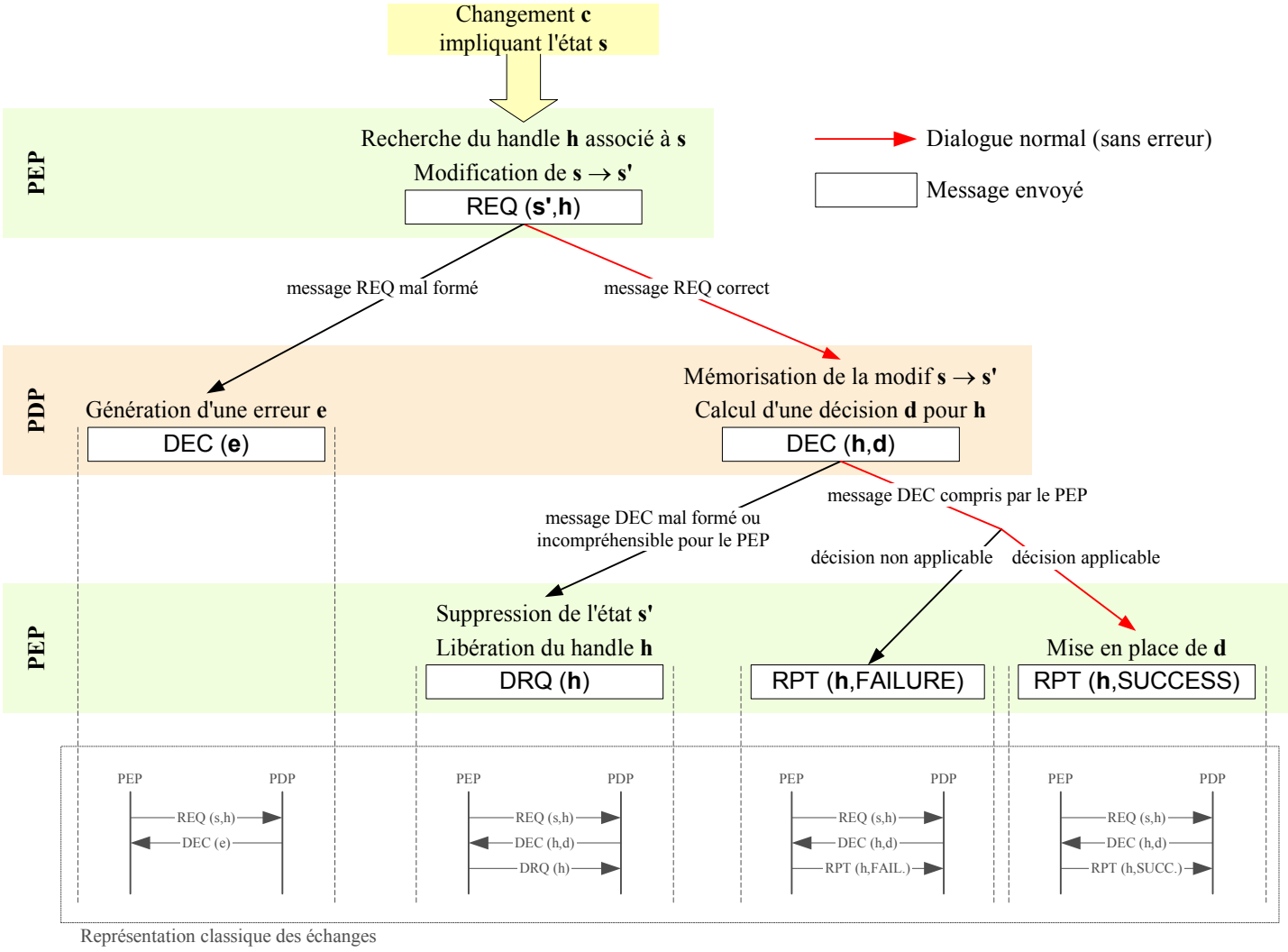


Figure 2-18 : Echanges COPS : Modification d'un état par le PEP

3.3.2.1.3 Suppression d'un état

♦ *Par le PDP*

Bien que cela ne soit pas vrai dans tous les modèles de gestion par politiques, il est possible que le PDP veuille supprimer un état.

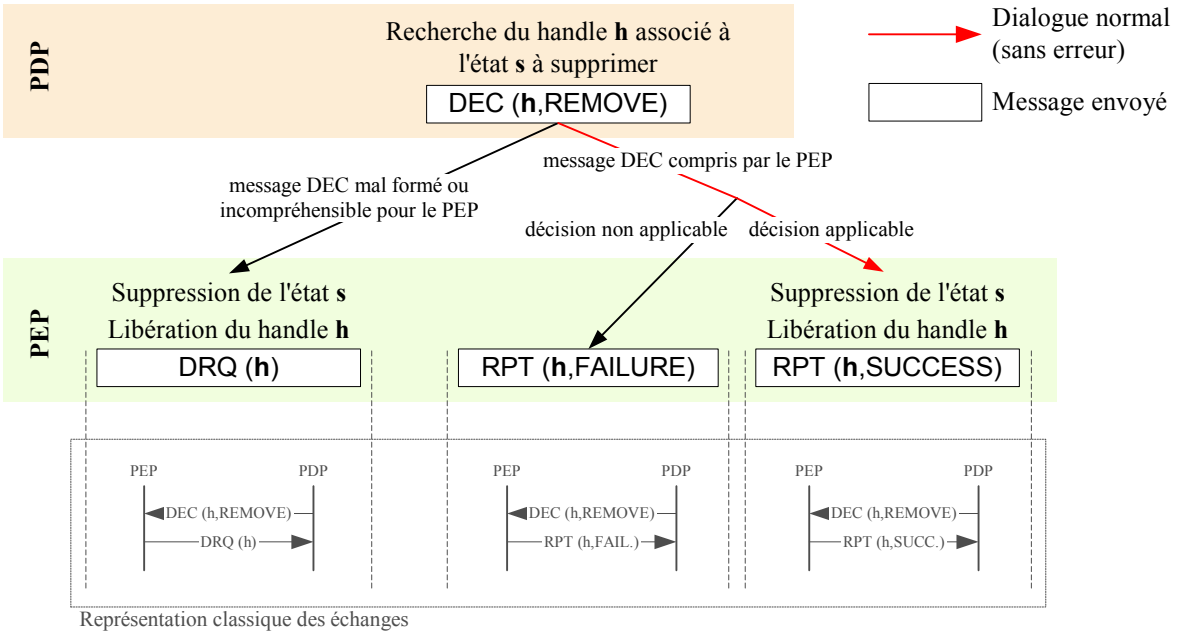


Figure 2-19 : Echanges COPS : Suppression d'un état par le PDP

Il est intéressant d'apporter quelques précisions à Figure 2-19 :

- Tout d'abord, a priori le PDP attend de recevoir un rapport de succès avant de supprimer le couple (état, Handle) de sa mémoire, même si cela n'est pas explicitement dit dans COPS.
- Ensuite, bien que cela soit peu probable, il est toujours possible que le PEP reçoive un message qu'il ne puisse pas comprendre à cause d'un bug ou d'une erreur de transmission non corrigée par TCP. Dans ce cas, comme déjà vu plus haut, le PEP supprime l'état ce qui correspond par chance à ce que le PDP lui demandait de faire.
- Enfin, il est possible que le PEP comprenne bien la demande du PDP mais qu'il ne peut la satisfaire au moment où il la reçoit. Il le signale au PDP dans un message RPT(FAILURE). COPS ne précise pas ce que doit faire le PDP dans ce cas. Doit-il supprimer l'état malgré tout ? Doit-il faire un certain nombre de tentatives de suppression avant de prendre des mesures plus radicales (comme la fermeture de la session) ? Le comportement du PDP est défini lors de son implémentation selon les besoins de l'administrateur.

♦ *Par le PEP*

Le PEP peut lui aussi supprimer un état. Il utilise pour cela le message DRQ en précisant le Handle de l'état à supprimer. Aucun message du PDP n'est attendu en retour.

3.3.2.2 Les opérations de maintenance

3.3.2.2.1 Les rapports

Il peut être intéressant pour aider le PDP dans ses prises de décisions que les PEPs lui envoient régulièrement des rapports. Le contenu des rapports dépend à la fois du modèle de gestion de politiques et des besoins du PDP. Ces rapports sont transmis au PDP dans des messages RPT de type Accounting. Les informations du rapport sont encapsulées dans des objets ClientSI.

La fréquence des rapports peut être limitée par le PDP. La fréquence maximale est alors transmise au PEP lors de l'ouverture de la session dans le message CAT (objet AcctTimer).

3.3.2.2.2 Les synchronisations

Afin de palier à des erreurs imprévisibles (bug du PEP ou du PDP, erreur de transmission non corrigée par TCP…), il n'est pas inutile d'effectuer périodiquement une synchronisation des états mémorisés par le PEP et le PDP.

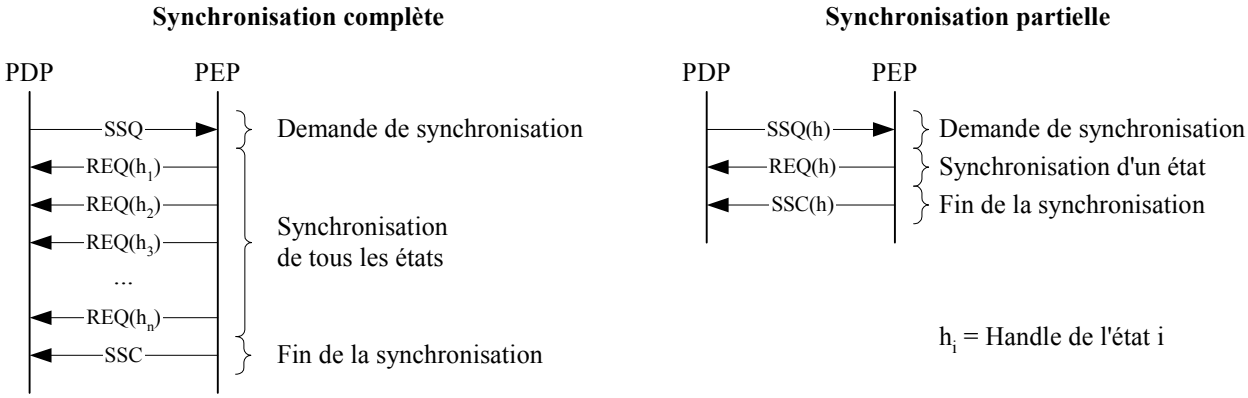


Figure 2-20 : Echanges COPS : Synchronisation

Il n'est pas précisé dans COPS comment le PDP doit traiter les informations récupérées lors d'une synchronisation :

- Comment réagir face à un message REQ mal-formé ? Faut-il envoyer immédiatement un message DEC d'erreur ou attendre la fin de la synchronisation ?
- De même, comment réagir devant un message REQ bien formé, mais contenant des informations différentes de celles dont le PDP a connaissance ? Faut-il répondre immédiatement ou attendre la fin de la synchronisation ? Faut-il modifier les états du PDP ou ceux du PEP ?

Tous ces choix sont à définir lors de l'implémentation du PDP en fonction des préférences de l'administrateur.

3.3.3 Les pannes

COPS doit être résistant aux pannes. Puisqu'on ne peut pas empêcher les pannes, plusieurs mécanismes ont été définis pour y palier.

Afin de simplifier et clarifier cette section, les notations suivantes seront utilisées :

- PDP_p est le PDP principal gérant un *Client-type* donné d'un PEP donné.

- PDP_s est un PDP de secours pouvant prendre en charge ce même *Client-type* lors d'une panne de PDP_p. PDP_s n'existe pas forcément.

3.3.3.1 Détection

Avant de pouvoir gérer une panne, il faut la détecter :

- Pendant un échange entre le PEP et PDP_p, la détection est faite par TCP.
- Pendant une période de silence, le PEP émet régulièrement des messages KA. Si le PDP_p ne lui répond pas (message KA), le PEP en déduit soit une panne de réseau, soit une panne du PDP_p.

3.3.3.2 Fonctionnement temporaire

Lorsque le PDP_p n'arrive plus à communiquer avec un PEP, il enclenche un compte à rebours. Lorsque celui-ci arrive à échéance, le PDP_p supprime de sa mémoire les états relatifs au PEP injoignable. La panne d'un ou plusieurs PEPs ne gène pas le PDP_p, il continue de fonctionner normalement en écoutant tous les messages COPS qui lui arrivent.

Au contraire, si le PEP détecte une panne du PDP_p, il ne peut pas l'ignorer puisqu'il a besoin d'un PDP pour fonctionner. Deux cas peuvent se présenter : soit le PEP se connecte à un autre PDP (PDP_s), soit il fonctionne de manière autonome.

♦ *PDP secondaire (PDP_s)*

COPS ne précise pas comment le PEP apprend l'adresse de PDP_s mais on peut envisager plusieurs scénarios :

- Le PEP utilise le même mécanisme (statique ou dynamique) pour découvrir PDP_s que celui qui lui sert à trouver l'adresse de PDP_p.
- Le PEP mémorise systématiquement l'adresse des PDP auxquels il a réussi à se connecter depuis son dernier redémarrage. Ainsi lorsque le PDP actuel n'est plus joignable, le PEP tente de se connecter à l'avant dernier PDP, puis à l'avant avant dernier…

Si le PEP arrive à se connecter à PDP_s, il doit d'abord se synchroniser avant de pouvoir fonctionner normalement. En effet, pour prendre des décisions, PDP_s doit savoir quelles sont les politiques installées dans le PEP.

Deux cas peuvent se présenter :

- Si le PEP conserve en mémoire des états installés, il doit obligatoirement signaler (dans un objet LastPDPAddr) à PDP_s, l'adresse du PDP qui a installé ces états ; dans notre cas, il s'agit donc de l'adresse de PDP_p. Deux sous cas existent :
  - PDP_s et PDP_p sont synchronisés (cette synchronisation dépasse le cadre de COPS). Alors il n'est pas utile de faire une synchronisation complète entre le PEP et PDP_s, mais le PEP est tenu de signaler tout changement intervenu pendant qu'il n'était connecté à aucun PDP.
  - Les deux PDPs ne communiquent pas. Alors PDP_s demande une synchronisation complète (message SSQ sans objet Handle) et le PEP envoie (dans des messages REQ) tous ses états installés.
- Si le PEP n'a aucun état en mémoire, il n'indique pas l'adresse de PDP_p et PDP_s demande une synchronisation complète.

Une fois que le PEP et PDP_s partagent les mêmes états, ils peuvent commencer une phase de fonctionnement normal (section 3.3.2).

♦ *Fonctionnement autonome*

S'il n'y a pas de PDP de secours (PDP_s n'existe pas) ou dans l'intervalle de temps où le PEP n'est plus connecté à PDP_p et ne l'est pas encore à PDP_s, le PEP doit fonctionner de manière autonome. COPS ne précise pas la manière de s'y prendre mais l'idée est de ré-appliquer les décisions déjà prises.

Ce mode de fonctionnement doit être temporaire. Après un certain temps, les décisions précédemment prises doivent être supprimées, notamment pour éviter des trous de sécurité. COPS n'indique pas ce que doit faire le PEP après leur suppression.

3.3.3.3 Retour à la normale

Lors d'une panne de PEP, le retour à la normale est totalement transparent du point de vue de PDP_p puisque le PEP se contente de se reconnecter comme après un redémarrage classique.

Au contraire, le retour à la normale après une panne de PDP_p demande un certain travail d'abord pour détecter la fin de la panne, puis pour redémarrer la session entre le PEP et PDP_p.

♦ *Détection de la fin de la panne*

Si le PEP est connecté à PDP_s, il n'a rien à faire pour détecter la fin de la panne de PDP_p. En effet, c'est à PDP_s de mettre en place un mécanisme pour être averti de la fin de la panne. Ce mécanisme n'est pas précisé par COPS puisqu'il se situe nécessairement à un niveau inférieur à COPS (par exemple au niveau TCP). Une fois que la fin de la panne a été détectée par PDP_s, celui-ci l'indique au PEP en fermant la connexion avec un message CC contenant un objet PDPRedirAddr indiquant l'adresse de PDP_p.

Si le PEP fonctionne en mode autonome, il doit vérifier, lui-même, régulièrement la fin de la panne. Là encore, le mécanisme de vérification se situe à un niveau inférieur à COPS.

♦ *Réouverture d'une session*

Si le PEP n'a plus d'état en mémoire, l'ouverture d'une session après une panne est identique à une ouverture classique (cf. Figure 2-14).

En revanche, si le PEP a encore en mémoire des états, PDP_p a besoin de les connaître pour prendre ses décisions. Pour cela, le PEP indique dans un objet LastPDPAddr joint à son message OPN l'adresse du dernier PDP avec lequel il s'est synchronisé qui peut être PDP_s ou même PDP_p si la panne a été très courte et que le PEP n'a pas pu ou n'a pas eu le temps de se connecter à PDP_s. Suivant que PDP_p et PDP_s peuvent communiquer ensemble, PDP_p demande ou non une synchronisation complète.

3.3.4 La fermeture

Puisque COPS supporte les pannes, il supporte aussi les fermetures de sessions brutales. Cependant, il offre des outils pour fermer proprement une session grâce aux messages CC. D'ailleurs la première opération que le PEP devrait effectuer quand il détecte une perte de connexion avec son PDP est de lui envoyer un message CC pour chaque session ouverte avec ce PDP au cas où la connexion ne serait interrompue que dans un seul sens.

L'utilisation des messages CC par le PEP permet au PDP de libérer immédiatement les ressources qui étaient réservées au PEP. L'utilisation des messages CC par le PDP permet au PEP de prendre les mesures

nécessaires le plus tôt possible et éventuellement d'être redirigé vers un autre PDP. Dans tous les cas, les messages CC permettent d'indiquer la raison de la déconnexion.

4 La représentation et le stockage des politiques

Le chapitre précédent a détaillé comment les PEPs et PDPs dialoguent ensemble pour appliquer des politiques. Il est maintenant intéressant de savoir comment celles-ci sont définies et comment elles sont stockées.

4.1 PCIM (Policy Core Information Model)

Il faut rappeler que 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.

PCIM [28] est un modèle de description de politiques indépendant du domaine d'application permettant d'atteindre cet objectif. Il existe d'autres langages pour spécifier des règles (Ponder [32], PFDL [33], XML…). Puisque aucun langage standardisé n'a été utilisé pendant ce stage, seul PCIM sera rapidement décrit dans ce chapitre parce qu'il s'agit du langage préconisé par l'IETF.

4.1.1 L'origine et les extensions de PCIM

PCIM a été conçu au sein de l'IETF (WG Policy) en s'appuyant sur CIM (Common Information Model) [30] réalisé par le DMTF (Distributed Management Task Force). CIM est un modèle générique orienté objet qui représente et organise l'information. Il est capable de gérer les ordinateurs, les périphériques (imprimantes…), les connecteurs (PCI, USB…), les fichiers, les logiciels, les utilisateurs, les organisations…

PCIM spécialise CIM pour définir des politiques. Il reste malgré tout généraliste en ce sens qu'il ne se restreint pas à un domaine de services particulier.

Il a été étendu par PCIM(e) [29] qui a pour fonction de rendre PCIM plus flexible en permettant aux différents domaines d'homogénéiser leurs concepts. Les principaux avantages de PCIM(e) sont :

- une meilleure gestion des priorités,
- l'ajout de variables et de valeurs,
- la définition de variables générales,
- l'ajout de règles simplifiées fondées sur les variables et
- la possibilité d'avoir des règles conditionnées par d'autres règles.

QPIM (QoS Policy Information Model) [31] est la première extension de PCIM conçue par le WG Policy. QPIM établit un cadre de travail standard pour spécifier et représenter les politiques dans le domaine de la QoS.

D'autres extensions de ce type devraient être définies dans l'avenir dans d'autres domaines de services (notamment pour la sécurité).

4.1.2 Un langage plutôt déclaratif

4.1.2.1 Définitions

Un langage déclaratif sert à décrire les relations entre les variables en termes de fonctions ou de règles d'inférence. L'interpréteur ou le compilateur utilise ensuite un algorithme fixe pour calculer un résultat. En d'autres termes, un langage déclaratif permet de décrire le problème que l'on veut résoudre et l'interpréteur trouve la ou les solutions du problème s'il y en a.

Un langage impératif, ou procédural, repose sur un concept fondamentalement différent. Il ne sert pas à décrire un problème, mais à décrire une solution. Cela suppose donc que la solution, ou plutôt l'algorithme pour l'atteindre, est connue. Dans ce cas, le langage permet d'indiquer à la machine toutes les actions à effectuer *impérativement* pour répondre au problème.

4.1.2.2 Le choix de PCIM

PCIM se veut plutôt déclaratif. Cependant, il reconnaît quand même que les langages procéduraux peuvent être attrayants. Ainsi PCIM fournit quelques options permettant de fixer l'ordre d'exécution des règles de politiques et de préciser si cet ordre est obligatoire ou facultatif.

En dépit de ces options, PCIM n'est pas un langage impératif car pour cela il faudrait spécifier le déroulement du test des conditions ainsi que le déroulement de l'exécution des actions. Le WG Policy a choisi de ne pas le faire pour ne pas imposer de contraintes sur l'implémentation. Pour la même raison, PCIM n'est pas non plus réellement déclaratif car il ne définit pas l'algorithme fixe qui trouve une solution au problème décrit, c'est à dire qui permet de déduire un comportement particulier du réseau à partir des règles de politiques et éventuellement d'autres informations.

Malgré tout, la conception de PCIM est plus proche du modèle déclaratif en ce sens où la partie impérative doit plutôt être définie pendant l'implémentation et que, lors de la définition des règles, on laisse de préférence à l'interpréteur le soin de choisir le meilleur ordre pour traiter aussi bien les conditions que les actions.

4.1.3 Un modèle orienté objet

Tout comme CIM, PCIM est un modèle orienté objet. Il est fondé sur deux hiérarchies de classes :

- Les classes structurelles qui forment les éléments de bases des politiques.
- Les classes associatives qui déterminent les relations entre ces éléments.

Figure 2-21 présente les principales classes de PCIM en mettant en avant leurs relations. Les deux hiérarchies complètes sont données plus bas.

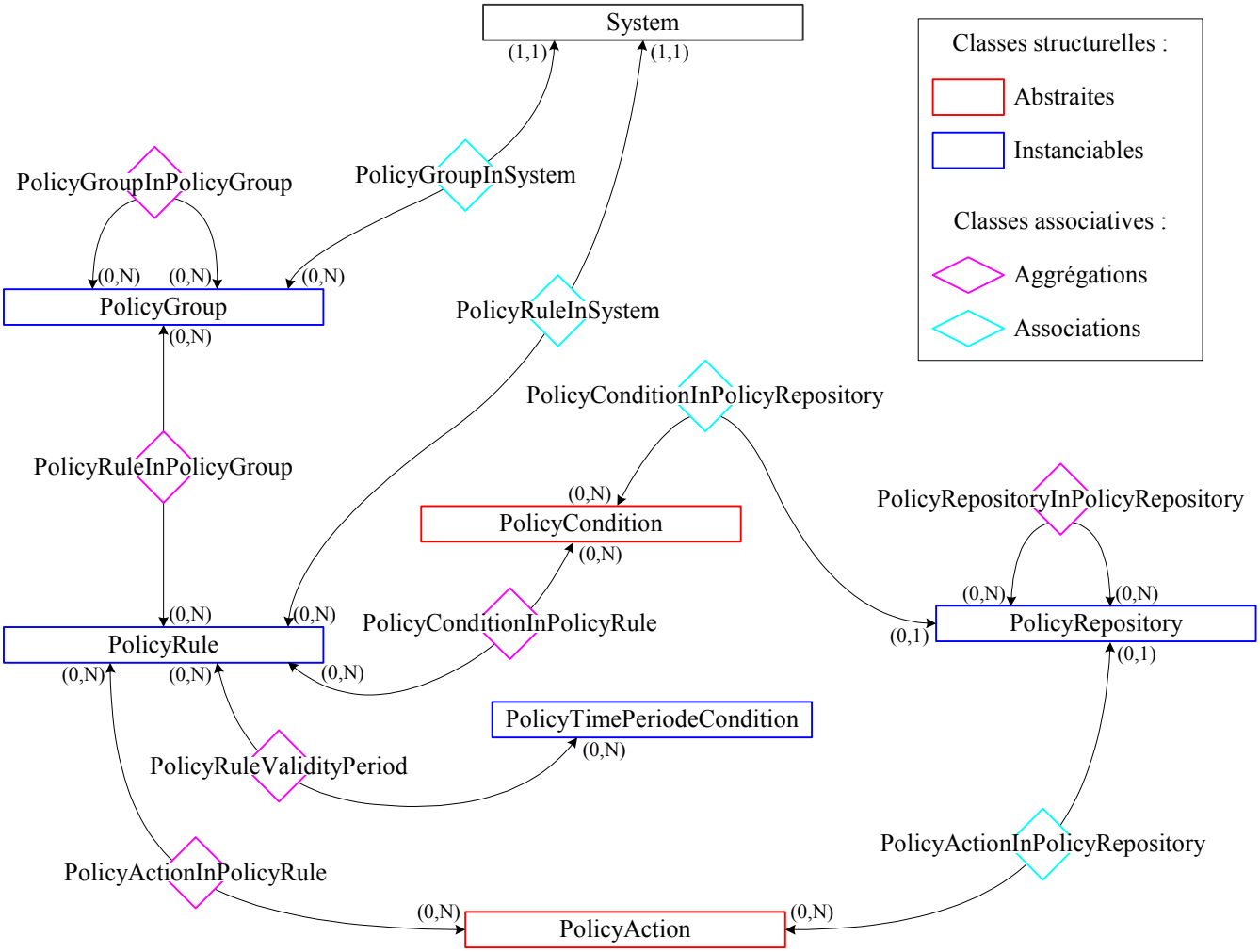


Figure 2-21 : Relations entre les classes PCIM

Remarque : Comment lire les cardinalités (*x*,*y*) dans la figure ci-dessus ? Prenons, par exemple, la relation entre les objets PolicyGroup et les objets PolicyRule qui correspondent respectivement à des groupes de règles et à des règles (voir plus bas pour plus de détails) :

- La cardinalité (0,N) proche de PolicyGroup indique qu'une règle peut être contenue dans aucun, un ou plusieurs groupes.
- La cardinalité (0,N) proche de PolicyRule indique qu'un groupe peut contenir aucune, une ou plusieurs règles.

4.2 Le Policy Repository

Le *policy repository* est un magasin destiné à stocker les politiques du réseau et toutes les informations relatives aux politiques. Ce terme est fréquemment traduit en français par « base de données de politiques », cependant il n'a pas été traduit dans ce document pour éviter toute confusion avec les bases de données *relationnelles* qui sont une implémentation possible du *policy repository*. Les annuaires (*directories*) en sont une autre implémentation possible.

Le stockage des politiques est un sujet encore mal défini. Il n'existe pas de standard pour orienter les entreprises vers une solution commune.

4.2.1 Les contraintes

Le but principal du *policy repository* est de stocker les politiques. Cependant, cette tâche 'triviale' doit aussi respecter d'autres contraintes extrêmement difficiles à satisfaire.

♦ *Intégrité des données*

Le *policy repository* doit garantir l'intégrité des données selon plusieurs points de vue.

L'intégrité, d'un point de vue transactionnel, est une propriété qui garantie que toutes les modifications de données doivent être complètes et vérifiées. Une modification, même constituée de plusieurs étapes, doit être considérée comme une commande atomique. Donc elle doit être effectuée complètement ou bien, en cas d'impossibilité, toutes les modifications partielles doivent être annulées.

L'intégrité, d'un point de vue système réparti, consiste à s'assurer que les données redondantes sont toujours cohérentes dans tous les éléments du système. Cette cohérence n'a pas besoin d'être vraie à n'importe quelle instant, mais il est impératif que la fréquence de synchronisation des éléments du système soit supérieure à la fréquence à laquelle le Policy Repoitory est consulté afin que les PDPs, s'il y en a plusieurs, reçoivent tous les mêmes politiques.

L'intégrité, d'un point de vue multi-utilisateurs, empêche qu'une donnée soit modifiée simultanément et différemment par plusieurs utilisateurs.

L'intégrité référentielle permet d'imposer certaines règles pour créer, modifier ou supprimer une information. Par exemple, il peut être interdit de supprimer une entreprise du *policy repository* si celui-ci contient encore des personnes travaillant dedans ; ou au contraire, la suppression d'une entreprise peut entraîner automatiquement la suppression de tout son personnel.

♦ *Passage à l'échelle*

A l'heure actuelle, le passage à l'échelle n'est pas une priorité. En effet, en pratique les solutions de gestion par politiques déployées ont des besoins de stockage assez restreints car le nombre de politiques et de nœuds actifs n'est pas très important. Cependant, il est très probable que cette situation évolue.

La principale solution pour étendre les capacités du *policy repository* est de distribuer les données sur plusieurs serveurs. Malheureusement, cette solution n'est pas sans poser d'autres problèmes. En effet, l'intégrité des données est plus difficile à garantir dans ce cas. Par ailleurs, il peut devenir nécessaire d'établir une topologie des serveurs difficile à optimiser. Il est en effet assez évident que la localisation des données peut avoir un impact significatif sur les performances du système.

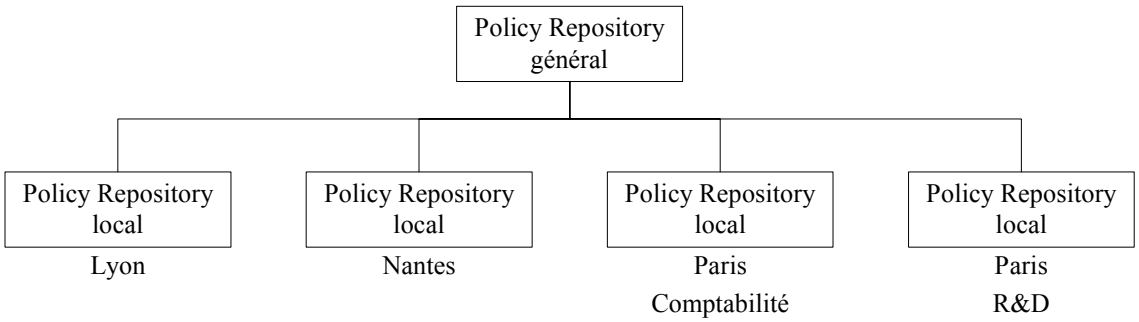


Figure 2-22 : Exemple de topologie d'un *policy repository* distribué

♦ *Sécurité*

Le *policy repository*, de part sa fonction, est un élément très sensible. La sécurité doit donc être une contrainte forte. Cette sécurité recouvre plusieurs aspects :

- Authentification des utilisateurs
- Intégrité des données échangées. Un utilisateur malveillant ne doit pas pouvoir en modifier le contenu.
- Eventuellement confidentialité des données échangées.

♦ *Accès à d'autres sources de données*

Le *policy repository* contient les politiques du réseau. Cependant, il peut être nécessaire de disposer d'autres sources de données, par exemple des informations d'authentification ou de facturation (voir Figure 2-2). Dans certains cas, il est possible de mettre ces informations directement dans le *policy repository* à coté des politiques. Cette approche n'est pas forcément souhaitable. Elle n'est pas non plus toujours réalisable.

L'idéal serait de mettre en place un méta-index permettant d'accéder à toutes les données. Cela implique que le *policy repository*, ainsi que les autres sources de données, soit compatible avec ce méta-index.

4.2.2 Les solutions

Il existe principalement deux solutions techniques pour réaliser le *policy repository* : les bases de données relationnelles et les annuaires (*directories*). Chacune d'elles a ses avantages et ses défauts.

4.2.2.1 Les bases de données relationnelles

Une base de données relationnelle est un ensemble d'objets organisés par des relations. Concrètement, pour chaque objet, il existe une table dont les colonnes représentent les attributs de l'objet et les lignes correspondent aux instances de l'objet stockées dans la base de données. Il existe aussi une table pour chaque relation. Dans une base de données relationnelle, il est possible d'accéder aux données et de les ré-assembler de différentes manières sans modifier les tables.

L'interface pour accéder à une base de données relationnelle est généralement le SQL (Structured Query Langage).

Les avantages des bases de données relationnelles sont :

- L'intégrité transactionnelle est garantie.
- L'intégrité référentielle est parfaitement maîtrisée.
- Il existe des mécanismes de contrôle d'accès concurrents (intégrité multi-utilisateurs).

- Une fréquence élevée, aussi bien de consultation que de modifications des données, ne pose pas de problème particulier.
- Le passage à l'échelle est maîtrisé tant qu'il n'est pas nécessaire de distribuer les données sur plusieurs serveurs.

Les inconvénients sont :

- Il est possible de répartir l'information, ou juste la structure de la base, en plusieurs endroits. Cependant cette opération a un coût. D'abord, il s'agit de mécanismes propriétaires. Ensuite, cela supprime ou modifie souvent les garanties d'intégrité.
- Il n'existe pas de mécanisme pour rendre la connexion à une base de données répliquée indépendante de sa localisation.
- Les données stockées peuvent provenir d'un grand nombre de types d'applications. Cependant, les bases de données sont mal adaptées pour stocker des données structurées par des modèles orientés objet avec des informations hiérarchiques et des processus d'héritage.
- Les mécanismes de sécurité, s'ils existent, sont propriétaires ou en tout cas ils ne sont pas directement intégrés à SQL.

4.2.2.2 Les annuaires

Un annuaire est un type de base de données spécifique basé sur une architecture hiérarchique. Cette architecture permet de retrouver n'importe quelle information très facilement. Il est bien sûr possible d'insérer ou de modifier une information dans un annuaire, cependant celui-ci est prévu pour être plus sollicité en lecture qu'en écriture.

Comme pour toutes les bases de données, son fonctionnement et son organisation interne dépendent fortement de son constructeur. Pour cette raison, dans un souci d'interopérabilité, tous les annuaires utilisent un protocole d'accès standardisé : LDAP.

4.2.2.2.1 LDAP (Lightweight Directory Access Protocol)

Le service d'annuaire X.500 est un standard conçu en 1988 par les opérateurs télécoms pour interconnecter tout type d'annuaire. Malheureusement cette norme ISO était très lourde à mettre en place. LDAP est apparu en 1993 pour alléger le protocole DAP de X.500 (d'où son nom) et l'adapter au protocole TCP/IP.

♦ *Les fonctionnalités*

Le protocole LDAP est uniquement prévu pour gérer l'interfaçage avec les annuaires (il ne s'occupe pas de leur fonctionnement). Il s'agit d'une norme définissant comment les informations sont échangées entre le client et le serveur LDAP et comment les données sont représentées.

Ainsi ce protocole définit :

- Un modèle d'information définissant le type d'information stockée dans l'annuaire.
- Un modèle de nommage (parfois appelé modèle de désignation) définissant comment l'information est organisée et référencée.
- Un modèle fonctionnel (parfois appelé modèle de services) définissant la manière d'accéder aux informations et éventuellement de les modifier, c'est-à-dire les services offerts par l'annuaire.
- Un modèle de sécurité définissant les mécanismes d'authentification et de droits d'accès des utilisateurs à l'annuaire.

♦ *La structure des données*

Dans un annuaire, une information stockée est appelée *entrée* ou encore DES (Directory Service Entry). Une entrée peut représenter une organisation (i.e. une entreprise), une personne, un groupe, un serveur, une imprimante…

Les entrées sont structurées dans une arborescence hiérarchique appelée DIT (Directory Information Tree). Chaque nœud de l'arbre correspond à une entrée.

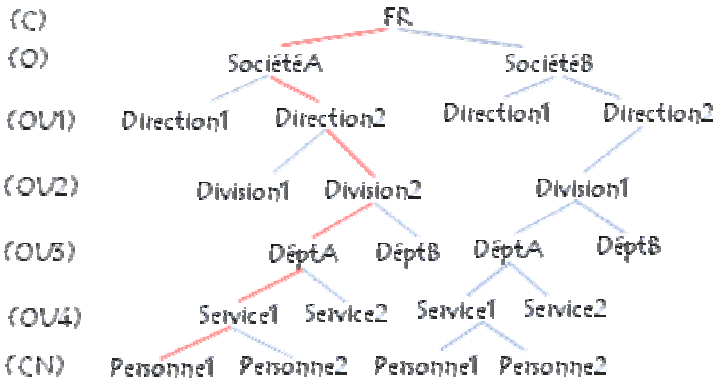


Figure 2-23 : Exemple de DIT

Une entrée possède un type et des attributs. Ces attributs peuvent être obligatoires, facultatifs ou dédiés à la maintenance de l'annuaire (par exemple, la date de dernière modification). Une entrée dans un DIT est décrite par ses propres attributs (par exemple, pour une personne, son nom, son surnom, son adresse, etc.) et par les attributs de tous les objets dont elle descend.

♦ *Les services d'annuaires répartis*

Un annuaire réparti est un annuaire hébergé sur plusieurs serveurs. Cela peut être imposé par le volume d'entrées à gérer, leur gestion répartie sur plusieurs sites, les types d'accès au réseau physique ou le mode d'organisation de la société. LDAP fournit deux services pour gérer un annuaire réparti : le *replication service* et le *referral service*.

Le *replication service* (service de duplication) permet aux serveurs de s'échanger leur contenu et de se synchroniser. Figure 2-24 donne un exemple d'utilisation de la duplication de données. Dans cet exemple, la duplication permet la mise en place d'un annuaire secondaire qui prendrait le relais en cas de panne de l'annuaire principal. Elle permet également d'avoir deux annuaires identiques à des endroits géographiquement éloignés pour des questions de performances.

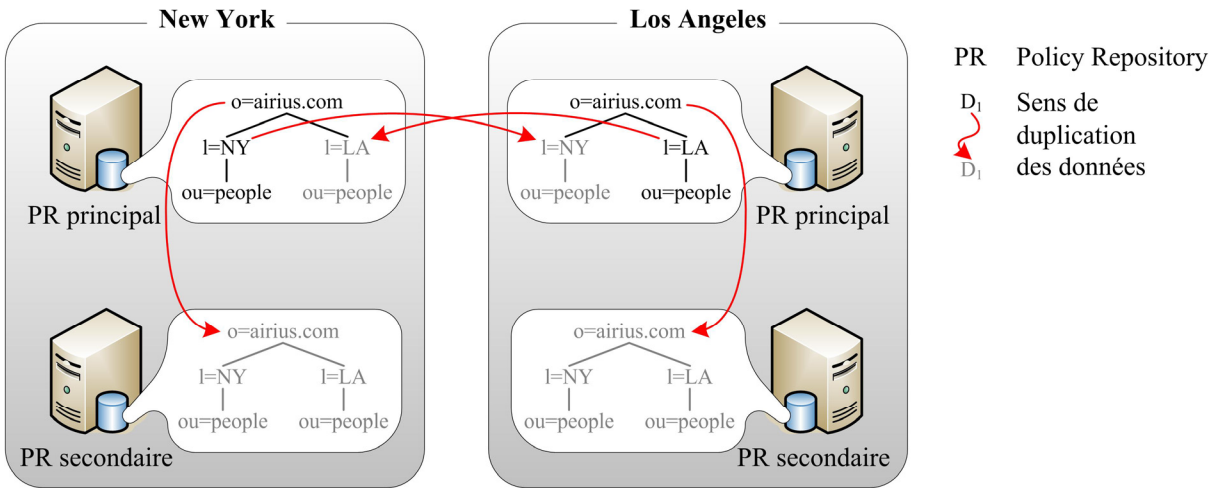


Figure 2-24 : Exemple d'utilisation du *replication service* de LDAP

Le *referral service* (service de renvoi) permet de partitionner un annuaire sur plusieurs serveurs, chacun de ces serveurs hébergeant seulement une partie de l'annuaire.

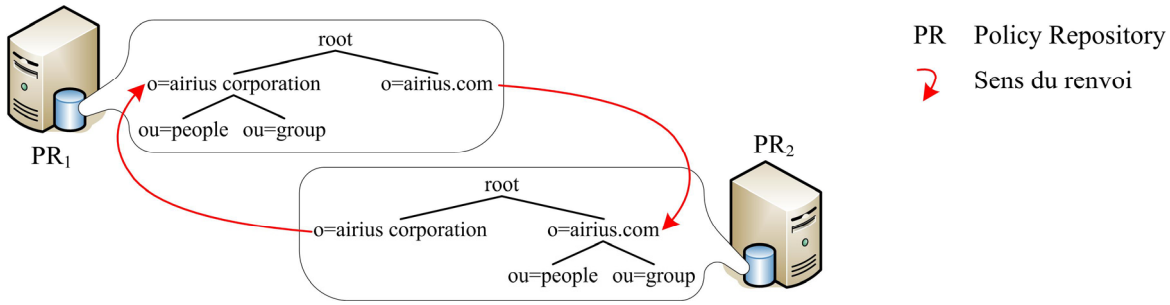


Figure 2-25 : Exemple d'utilisation du *referral service* de LDAP

Le renvoi peut être géré soit par le client (par exemple, après avoir interrogé PR_1, le client interroge PR_2 et obtient son information), soit par le serveur (par exemple, PR_1 contacte PR_2 et retourne le résultat au client). Dans ce dernier cas, on parle de chaînage (*chaining*) plutôt que de renvoi.

4.2.2.2.2 Avantages et inconvénients

Les avantages des annuaires sont :

- Il est possible de répartir (avec ou sans duplication) les informations sur plusieurs serveurs ce qui est une garantie de passage à l'échelle.
- La connexion au *policy repository* est indépendante de la localisation des serveurs.
- La structure des annuaires est parfaitement adaptée aux données organisées hiérarchiquement et notamment basées sur un modèle orienté objet comme PCIM. [36] normalise d'ailleurs l'utilisation de LDAP pour transporter des objets PCIM.
- Les opérations de consultation sont très rapides.
- La sécurité a été un objectif dès le début. Ainsi, LDAP intègre des mécanismes d'authentification, de signatures électroniques, de cryptographie, de filtrage réseau, de règles d'accès (ACLs) aux données et d'audit des journaux.

Les inconvénients des annuaires sont :

- Leur structure est mal adaptée pour certaines applications.

- Les opérations de modification sont assez lentes.
- Il existe peu ou pas de mécanismes pour garantir l'intégrité aussi bien référentielle, que multi-utilisateurs.
- L'intégrité transactionnelle n'est pas maîtrisée ce qui peut poser de graves problèmes d'interopérabilité. En effet, chaque constructeur exécute les commandes dans des ordres différents. Ainsi une transaction qui n'est pas menée à son terme peut corrompre le *policy repository*.
- L'intégrité d'un point de vue système réparti peut être difficile à obtenir si les modifications sont trop fréquentes.

4.2.2.3 Conclusion

Après avoir rapidement décrit les deux principales solutions pour mettre en place un *policy repository*, deux constats :

- Chaque solution a des avantages et des inconvénients et aucune ne répond à toutes les contraintes définies à la section 4.2.1.
- Certaines de ces contraintes ne sont résolues par aucune des solutions.

Par ailleurs, bien que les annuaires semblent plus propices à s'incorporer dans un PBN, à l'heure actuelle, les constructeurs préfèrent souvent utiliser des bases de données relationnelles car elles sont plus simples à mettre en œuvre et qu'il est encore rarement nécessaire de répartir le *policy repository* sur plusieurs serveurs.

Chapitre 3 : Le stage

1 Le projet RHODOS

1.1 Description du projet

Le projet RNRT RHODOS (Réseau Hybride Ouvert pour le DéplOiement de Services mobiles) a débuté en 2002 avec pour partenaire Eurecom, Siradel, France Telecom et Philips France. Thales Communications (aujourd'hui, Thales Land & Joint Systems) ainsi que l'Ecole des Mines d'Alès sont entrés au sein de RHODOS suite au désistement de Philips France.

1.1.1 Objectifs

Le projet RHODOS utilise comme point de départ les résultats du projet RNRT PLATON qui a montré la faisabilité d'une plate-forme ayant pour principales caractéristiques techniques :

- une communication mobile dans la bande UMTS/TDD (1900-1920 MHz) en temps réel
- un mode de duplexage temporel (TDD)
- les couches de protocoles nécessaires pour permettre une connectivité IP entre un mobile et une station de base.

Le projet RHODOS peut être vu comme une amélioration substantielle de la plate-forme existante. En effet, les développements logiciels de PLATON sont essentiellement axés sur les couches 1 et 2 (couche physique et couche de liaison de données), et sur les couches applicatives. Les aspects suivants en particulier ne sont que très peu abordés :

- gestion des ressources radio
- mobilité
- gestion de la qualité de service.

Le projet RHODOS vise à compléter et améliorer la plate-forme PLATON, par le développement des fonctionnalités évoquées ci-dessus.

Le principal objectif du projet est la réalisation et le déploiement d'un réseau hybride UMTS/TDD et WLAN (Réseau Local Sans Fil). Ce réseau sera déployé sur le site de Sophia-Antipolis, en utilisant pour ce faire une planification conjointe UMTS et WLAN. Par ailleurs, le réseau permettra de valider et d'intégrer des études sur la Qualité de Service, la gestion des ressources radio et la mobilité. Enfin, le réseau intégrera plusieurs services temps réel innovants, un service de visiophonie et un service de video-streaming, ces deux services ayant des besoins en terme de qualité de service.

1.1.2 Contexte scientifique

1.1.2.1 Historique de la recherche

Depuis le début des années 90, l'Europe a lancé un vaste programme de recherche autour des techniques de transmission large bande basées sur l'accès multiple à répartition de codes (AMRC large bande ou W-CDMA en anglais). Les projets supportés par la communauté européenne tels que CODIT et FRAMES ont livré des propositions pour la standardisation des systèmes 3 G UMTS/IMT2000. En janvier 1998, l'ETSI a décidé de baser l'interface radio de l'UMTS (UTRA : UMTS Terrestrial Radio Access) sur la technologie W-CDMA selon deux versions. Une version en mode FDD (Frequency Division Duplexing) basée sur du CDMA pur et une version en mode TDD (Time Division Duplexing) qui permet de combiner la notion d'accès multiple par répartition de temps (AMRT ou TDMA en anglais). D'une manière plus globale, les comités de standardisation ARIB (Japon), TIA (USA) et TTA (Corée) ont créé avec l' ETSI le groupe de travail 3GPP (3rd Generation Partnership Project) dont le but est de fournir un standard harmonisé mondialement pour permettre l'interopérabilité des systèmes mobiles de troisième génération.

Les principales caractéristiques des standards proposés sont les suivantes :

- le support de nouveaux services multimédias : liens asymétriques, débits variables, transmission par paquets
- amélioration des performances en termes de capacité et de couverture : options pour la détection multi utilisateurs, réjection d'interférences, traitements spatio-temporels, contrôle de puissance rapide, diversité multi trajets, etc.
- possibilité de services à débits élevés jusqu'à 2Mb/s

Plusieurs prototypes industriels ont vu le jour, mais étant donné leur caractère propriétaire, les résultats d'expérimentation sont peu disponibles dans la littérature scientifique et l'accès lui même à ces plates-formes reste problématique.

De leurs côtés plusieurs projets RNRT et IST développent des prototypes pour démontrer quelques composants technologiques liés aux systèmes mobiles de nouvelle génération. A ce titre, on peut citer des projets tels que PAESTUM et PETRUS (RNRT), CAST et TRUST (IST). On constate que les plates-formes expérimentales existantes pour les systèmes de troisième génération sont

- soit développées dans des environnements fermés
- soit non accessibles à cause de propriété industrielle
- soit non conçues pour fonctionner en temps réel
- soit non conçues pour des transmissions bi directionnelles.

1.1.2.2 Positionnement du projet RHODOS

A partir des études précédentes, il semble très évident que le développement d'un environnement ouvert et flexible pour la démonstration des technologies et des services innovants pour la prochaine génération de communications sans fil est un besoin réel. Il pourrait bénéficier à toute la communauté nationale et scientifique. RHODOS est destiné à l'implantation d'un tel environnement. Dans une perspective future, ces caractéristiques font de RHODOS un environnement idéal pour le support d'intégration et de développement de projets RNRT existants aussi bien que de futurs projets. D'un point de vue technique, RHODOS est différent des autres démonstrateurs dans les aspects décrits ci-dessous :

- Architecture radio-logicielle ouverte pour le traitement bande de base, qui permet l'implantation de tous les algorithmes avancés sans limitation de complexité pour les fonctionnement non temps réels
- Support des antennes multiples aussi bien en transmission qu'en réception et transmission full-duplex et fonctionnement temps réel en mode TDD
- Couches de protocoles ouvertes basées sur IP pour l'interconnexion de réseau avec la possibilité d'accès et de modification des procédures principales pour pouvoir expérimenter les nouvelles solutions et pour l'implantation des nouvelles applications et des nouveaux services (En particulier, la transmission multimédia, le browsing sur l'Internet, la videotéléphonie ...)
- Environnement complet d'expérimentation, permettant à des partenaires futurs des validations et des expérimentations très réalistes, grâce notamment à la convergence des deux standards UMTS et WLAN au sein d'un même réseau expérimental.

1.2 Organisation

1.2.1 Répartition du projet

Le projet RHODOS est divisé en 7 sous projets afin d'avoir une meilleure maîtrise des développements et des expérimentations à effectuer sur le démonstrateur.

♦ *Sous-projet 1 : Définition des tests et scénarios de configurations réseaux*

Ce sous-projet vise à définir de manière concise le cadre de l'étude en termes expérimentaux, d'objectifs et de critères d'évaluation. Pour ce faire, des études seront menées sur les scénarios d'utilisation du réseau hybride en prenant en compte des paramètres à la fois techniques, économiques et d'usage.

♦ *Sous-projet 2 : Planification conjointe UMTS-TDD et WLAN*

L'objectif de ce sous-projet est de développer des modèles de prédiction de la propagation des ondes électromagnétiques dans les multiples environnements couverts par la plate-forme proposée : réseau extérieur (UMTS) et réseau local sans fil (WLAN).

♦ *Sous-projet 3 : Etude de la gestion des ressources radio*

Ce sous-projet consiste à développer des couches permettant la gestion des ressources radio et de la mobilité. Les développements de ce sous-projet sont de deux ordres :

- tout d'abord, il s'agit d'apporter au démonstrateur « PLATON » des fonctionnalités nouvelles en terme de synchronisation des stations de base et de gestion des ressources radio
- ensuite, des études plus prospectives sont réalisées dans le domaine de l'allocation de canaux et la prise en compte des antennes multiples.

♦ *Sous-projet 4 : Etude de la gestion de la mobilité et de la QoS de bout en bout*

Le premier objectif de ce sous-projet est de définir de nouveaux protocoles de gestion de la qualité de service (QoS) de bout en bout de transmission des paquets dans un réseau hétérogène. Le second objectif de ce sous-projet est d'étudier, de spécifier et d'implémenter les mécanismes impliqués dans le handover entre plusieurs technologies d'accès, dans notre cas l'UMTS/TDD et le WLAN.

♦ *Sous-projet 5 : Service de visiophonie dans l'embarqué contraint avec QoS dans le lien radio*

Ce sous-projet consiste à réaliser une application de visiophonie sur un réseau composé de liens Internet, de liens radio formés par des terminaux UMTS-TDD et/ou 802.11b.

♦ *Sous-projet 6 : Mise en place d'une plate-forme de streaming vidéo avec gestion dynamique de la qualité de service*

L'objectif de ce sous-projet est de développer un service de vidéo streaming sur un réseau composé d'un lien radio (UMTS, WLAN, …) et d'une pile protocolaire IPv6 avec gestion de la Qualité de Service (QoS) dynamique, au sein du réseau. Un mécanisme de négociation et de maintien de QoS doit être assuré entre les couches applicatives chargées du filtrage vidéo dans les nœuds du réseau et les couches situées au-dessous, à savoir la couche IPv6 ainsi que les couches basses de la plate-forme UMTS développée par EURECOM.

♦ *Sous-projet 7 : Déploiement d'un réseau hybride, intégration et expérimentations*

Le principal objectif de ce sous-projet est le déploiement d'un réseau hybride UMTS/TDD et WLAN, sur le site de Sophia-Antipolis. Ce réseau expérimental intégrera les couches de protocoles et les applications développées dans le projet, pour former un démonstrateur très ambitieux cumulant des résultats de PLATON et du nouveau projet.

1.2.2 Le rôle de Thales dans RHODOS

Compte tenu de son savoir faire dans le développement d'applications de vidéo streaming sur les réseaux Internet, en particulier dans le cadre du projet ITEA BRIC, et plus généralement dans le domaine des technologies multimédia comme JPEG-2000 (projet IST 2Kan), Thales participe au projet RHODOS en prenant en charge le sous-projet 6 décrit précédemment.

Ce sous-projet est divisé en trois tâches :

♦ *Tâche 1 : Définition de scénarios de démonstration et des politiques de QoS*

Cette tâche consiste à mettre en place des scénarios pour la démonstration finale de qualité de service sur une plate-forme de vidéo streaming. La plate-forme sera composée d'un serveur applicatif, d'un client, et d'un réseau hétérogène constitué de divers nœuds, routeurs et passerelles.

De plus, le démonstrateur visé par le projet RHODOS offrira, grâce à la mise en place d'un réseau de deux ou trois stations de base, la possibilité d'étudier des scénarios de vidéo streaming en situation de mobilité

intercellulaire (*handover* en anglais). Un autre scénario portant sur l'interopérabilité entre systèmes sera étudié, lors du passage d'une cellule UMTS à une cellule 802.11.

♦ *Tâche 2 : Développement d'une application de streaming vidéo et d'une couche middleware de filtrage de contenu vidéo.*

Cette tâche se divise en deux points étroitement lies:

- Développement d'une application client/serveur de streaming vidéo. Le choix du format de contenu vidéo se portera sur une technologie adaptable en termes de débit. L'un des candidats les plus sérieux à l'heure actuelle est sans doute le Motion JPEG-2000 dont l'avantage est de pouvoir modifier son débit sans aucun travail de codage/décodage intensif.
- Mise en place, au niveau applicatif, d'une structure de filtrage de contenu/contrôle de débit afin d'adapter automatiquement le débit du contenu transféré en fonction des variations de l'état du canal. Les modules de filtrage vidéo qui prendront place sur les nœuds stratégiques du réseau (à définir) devront collaborer avec les couches inférieurs pour prendre en compte les caractéristiques particulière des liens sans fils.

Enfin, cette architecture sera TCP-courtoise (*TCP-friendly*).

♦ *Tâche 3 : Gestion et négociation dynamique de la Qualité de Service.*

Cette tâche consiste à établir des liens étroits entre les modules de filtrage de niveau applicatif et les couches basses pour gérer dynamiquement la qualité de service.

Cette tâche se décompose en :

- Expression des paramètres de QoS pour les modules de filtrage vidéo.
- Développement d'un algorithme permettant de garantir les paramètres de QoS ciblés par l'application en fonction des capacités du réseau à l'aide des modules de filtrage vidéo précédemment développés. Cette partie est liée à l'utilisation simultanée et optimale des mécanismes conjoints de protection présents à la fois au niveau des couches basses et au niveau de l'application.
- Gestion et négociation, d'abord statique, puis dynamique, des paramètres de QoS en fonction des scénarios envisagés.

1.2.3 Mon rôle

Mon stage s'inscrit dans le sous-projet 6 de RHODOS. Il a consisté à mettre en place une architecture pour piloter les modules de filtrage de contenu multimédia décrits dans la tâche 2 ci-dessus.

Cette architecture devait être à la fois puissante du fait des objectifs fixés à Thales (gestion dynamique de la QoS et interaction avec les couches basses) et souple à cause de l'imprécision du travail de certains partenaires dans le projet. De ce fait, il a été choisi d'utiliser la gestion par politiques présentée au chapitre 2 pour piloter les modules de filtrage.

Ce choix a reçu un accueil très favorable de la part des partenaires. De plus, il présentait l'avantage de pouvoir réutiliser l'architecture PDP/PEP déjà implémentée au sein de TAI dans le cadre d'autres projets.

Mon travail a donc consisté à ajouter un nouveau service, appelé contentAdaptation, à cette architecture.

2 L'architecture PDP/PEP existante

Dans le cadre du projet RTIPA, TAI a développé une architecture PDP/PEP ayant pour objectif de simplifier la gestion d'un réseau en permettant la configuration de ses équipements à partir d'un unique organe central.

Dans cet optique, le modèle de gestion choisi est celui du *provisioning*. Ce choix se justifie d'une part parce qu'il est plus simple à mettre en œuvre et d'autre part parce qu'il est tout à fait adapté aux services développés pour ce projet (QoS par DiffServ et sécurité par IPSec).

Par la suite, d'autres projets (ARCADE et POLLENS) ont réutilisé cette architecture en la faisant évoluer plus ou moins sensiblement. Les modifications apportées par le projet POLLENS ont même donné lieu à une architecture plus robuste basée sur une nouvelle souche COPS (la souche COPS Intel® ayant remplacée celle de Vovida®).

L'architecture réutilisée dans le projet RHODOS est celle du projet RTIPA, principalement parce qu'il est plus simple de développer de nouveaux services dans cette architecture. Par ailleurs les services de cette architecture sont facilement intégrables, sous forme de librairies, dans l'architecture du projet POLLENS.

2.1 Une architecture calquée sur le modèle trois-tiers

2.1.1 Présentation

L'architecture PDP/PEP du projet RTIPA a été développée afin de démontrer le principal intérêt de la gestion par politiques : la simplification de l'administration d'un réseau. Son passage à l'échelle ne faisait absolument pas parti des objectifs à atteindre. De même, elle n'a pas non plus été conçue pour être particulièrement robuste aux pannes.

Cette architecture est donc basée sur une structure trois-tiers simplifiée bien qu'il ait été dit au chapitre 2 que la structure trois-tiers ne répondait pas à toutes les contraintes de la gestion par politiques.

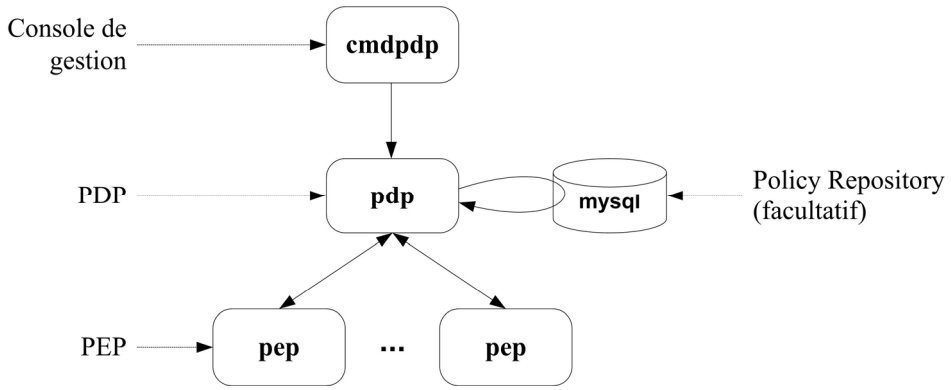


Figure 3-1 : Les entité de l'architecture PDP/PEP de RTIPA

L'organisation des entités de Figure 3-1 correspond bien à la structure trois-tiers simplifiée schématisée dans Figure 3-2.

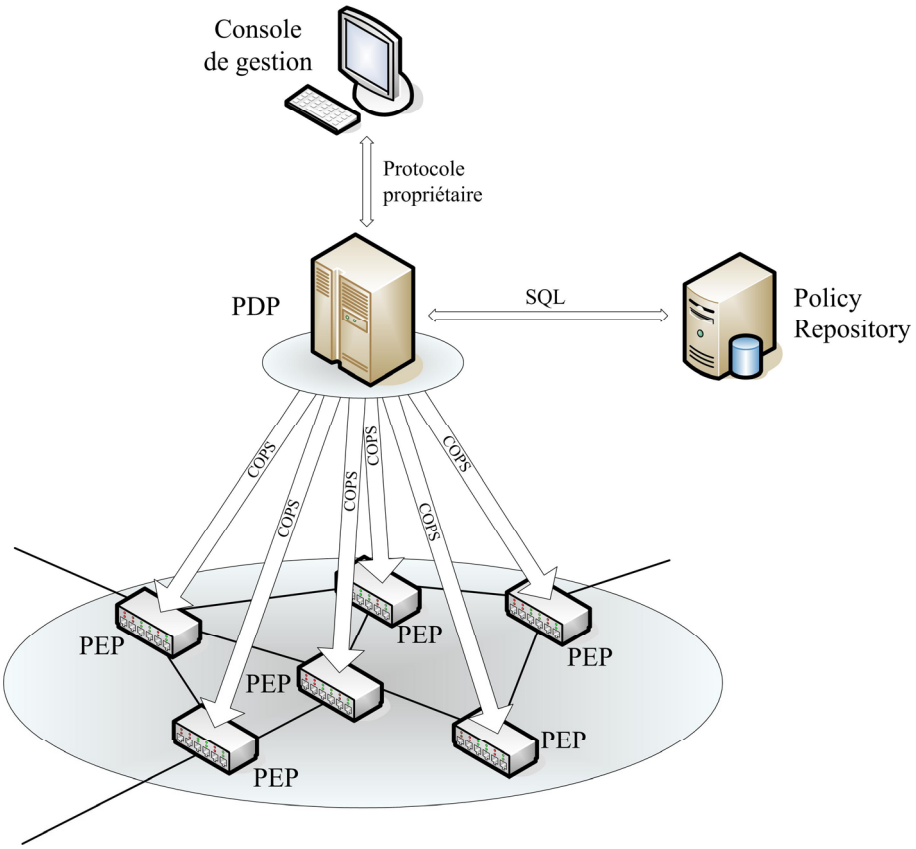


Figure 3-2 : PBN basé sur une structure trois-tiers simplifiée

2.1.2 Les entités

Il est intéressant de voir plus en détail le rôle de chacune des entités de Figure 3-1 en insistant sur les différences avec ce qui a été présenté au chapitre 2 notamment dans la section 2.1 page 23.

2.1.2.1 cmdpdp

L'entité « cmdpdp » a un double rôle :

- Celui de console de gestion : installer, supprimer et modifier des politiques.
- Celui de contrôle du PDP à distance : fermer/éteindre/redémarrer le PDP, activer/désactiver les PEP…

Il existe plusieurs versions de cmdpdp. Ces différentes versions varient sur deux aspects :

- Convivialité : interface graphique ou simple ligne de commande.
- Interactivité : retour d'information du PDP ou non.

Pour plus de simplicité, je n'ai utilisé pendant mon stage que la version la plus basique sous forme de ligne de commande sans retour d'information.

Aucune des versions ne gère les conflits entre les politiques.

2.1.2.2 mysql

Le *policy repository* est implémenté par une base de données relationnelle (mysql) car celle-ci est bien plus simple à mettre en œuvre qu'un annuaire.

L'entité mysql ne joue cependant pas tout à fait le rôle du *policy repository*. Son principal objectif est d'aider le PDP à gérer les pannes des PEPs.

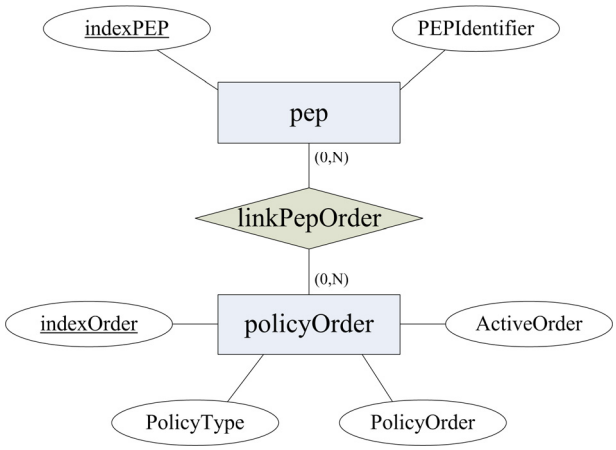


Figure 3-3 : Structure de la base de données du *policy repository*

Cette structure permet de savoir à tout moment quelle(s) politique(s) a(ont) été installée(s) sur quel(s) PEP(s). Donc si un PEP se reconnecte au PDP après avoir subit une panne, le PDP est capable lui réinstaller toutes les politiques qui le concernent.

2.1.2.3 pdp

L'entité « pdp » correspond bien sûr au PDP. Elle n'implémente cependant pas toutes les fonctions du PDP.

L' 'intelligence' de cette entité est en effet limitée. Elle dépend beaucoup du service mis en place, mais de manière générale, cette entité se contente de faire appliquer les commandes reçues de la console de gestion et n'est pas capable de réagir face à un changement du réseau non prévu par cette commande.

Malgré cette 'limitation', il s'agit tout de même d'une entité très complexe.

2.1.2.4 pep

L'entité « pep » représente un PEP.

A l'exception de quelques commandes COPS non implémentées, telle que la synchronisation, elle remplit très bien son rôle, c'est à dire appliquer les politiques transmises par l'entité pdp.

2.2 Une conception modulaire

Les deux éléments centraux de l'architecture du projet RTIPA (le PEP et le PDP) ont été développés de façon modulaire. Le principal intérêt de cette approche est de simplifier le travail de développement et de débuggage grâce à une conception rigoureuse, surtout pour une application de cette taille. Cela facilite également la lecture des sources lorsqu'une personne n'ayant pas participé au développement doit rajouter de nouvelles fonctionnalités.

2.2.1 pdp

L'entité pdp est très complexe puisqu'elle est constituée d'un nombre conséquent de modules dont une partie non négligeable fonctionne en parallèle.

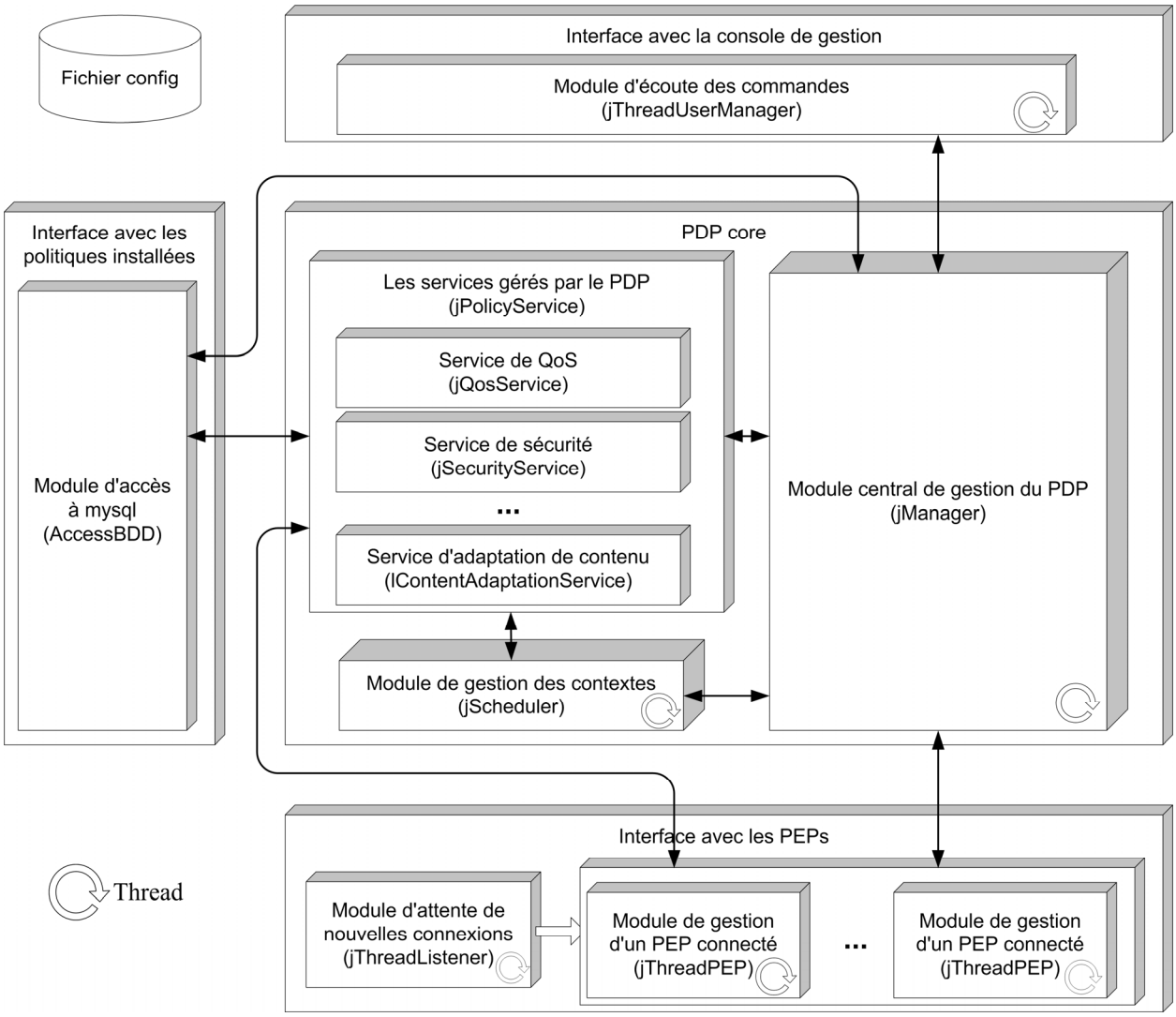


Figure 3-4 : Décomposition modulaire du pdp

Il est possible de répartir ces modules selon qu'ils s'agissent d'interfaces ou qu'ils appartiennent au cœur du PDP.

2.2.1.1 Les interfaces

Comme cela se voit sur Figure 3-1, le PDP possède trois interfaces pour dialoguer avec :

- la console de gestion,
- le *policy repository* et
- les PEPs.

Les deux premières interfaces sont assez simples et ne nécessitent chacune qu'un seul module.

La dernière interface qui communique avec les PEPs est plus complexe. Elle est constituée de :

- Un module qui attend de nouvelles connexions de PEP.
- Plusieurs modules (autant qu'il y a de PEPs connectés au PDP) communiquant avec un seul PEP.

Tous ces modules s'exécutent en parallèle afin de garantir une disponibilité maximale du PDP pour tous les PEPs de son réseau.

2.2.1.2 Le cœur

Le cœur du PDP est la partie la plus intéressante.

On y trouve d'abord un module central qui gère tout le PDP. C'est lui qui s'occupe de rediriger tous les messages entrants, provenant aussi bien de la console de gestion que des PEPs, vers les modules adéquats.

Il est épaulé par le module de gestion des contextes pour le traitement des commandes de la console de gestion. La section 2.3.2 page 72 définira le concept de contexte.

Enfin, chaque service offert par le PDP correspond à un module. Cette modularité des services est très puissante car elle permet d'ajouter très facilement un nouveau service. Il suffit de créer un nouveau module possédant un certain nombre de points d'accès bien définis (concrètement il suffit d'instancier la classe abstraite C++ jPolicyService) et de l'enregistrer dans le module central. Tout le développement spécifique à ce service est réalisé dans ce module et aucune autre modification du PDP n'est a priori nécessaire.

2.2.2 pep

La conception de l'entité pep est elle aussi modulaire.

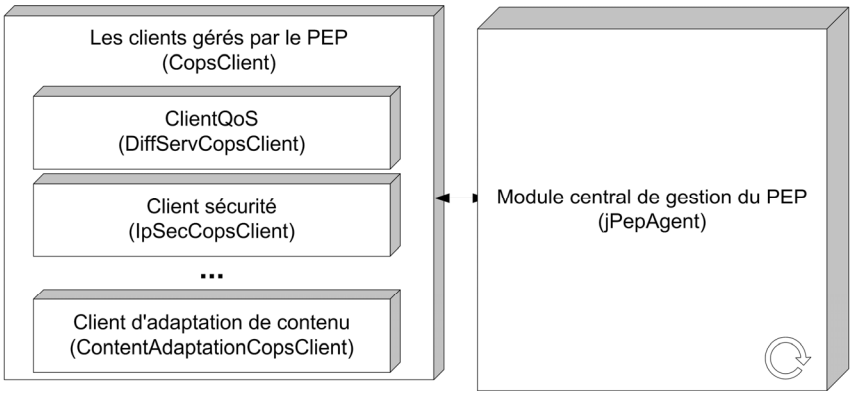


Figure 3-5 : Décomposition modulaire du pep

De manière évidente, l'entité pep est beaucoup plus simple que l'entité pdp. Cela correspond bien à l'esprit de l'architecture présentée au chapitre 2 où le PEP nécessite beaucoup moins de ressources que le PDP.

Le module central a un double rôle :

- Il qui dialogue avec le PDP (par l'intermédiaire de messages COPS).
- Il fait appel aux services adéquats pour traiter les ordres du PDP.

Les clients, qui appliquent les politiques, sont gérés de la même manière que les services dans le PDP. Ils sont modulaires. Pour en ajouter un, il suffit de créer un nouveau module possédant les points d'entrée requis. Concrètement il faut instancier la classe abstraite CopsClient. Puis il faut le déclarer dans le module central du PEP.

2.3 Principes de fonctionnement

2.3.1 Interaction entre les modules

Après avoir décomposé les entités pep et pdp en différents modules, il est intéressant de voir comment ceux-ci interagissent entre eux.

2.3.1.1 Traitement des commandes de la console de gestion

♦ *Installation d'une politique*

Lorsque la console de gestion envoie au PDP une commande d'installation de politique, cela donne lieu aux échanges suivants (Figure 3-6) :

1. Réception de la commande par le module d'interfaçage.
2. Transfert de la commande au module central pour l'interpréter. Ce module reconnaît qu'il s'agit d'une commande d'installation et récupère les informations suivantes :
   - Le service auquel la commande est adressée.
   - Le contenu de la commande (i.e. la politique à installer) sans l'analyser.
   - La liste des PEPs où installer la politique.
3. Transfert du contenu de la commande et de la liste des PEPs concernés au service adéquat.
4. Ce service fait appel au module d'interfaçage mysql pour mémoriser la commande et les PEPs concernés dans le *policy repository*.
5. Stockage dans le *policy repository*.
6. Création d'un contexte de type *addReq* pour cette commande (le concept de contexte sera abordé dans la section 2.3.2 page 72).
7. Après interprétation du contenu de la commande et éventuellement traitement spécifique à chaque service, PEP_1 est choisi dans la liste des PEPs concernés. L'ordre que PEP_1 doit appliquer est alors envoyé au module dialoguant avec lui.
8. Envoi de l'ordre à appliquer dans un message COPS DEC de type *install*.
9. Réception et analyse du message COPS, par le module central de PEP_1, pour transmettre au client adéquat le contenu non interprété.
10. Analyse du contenu du message COPS et application de l'ordre qu'il contient.
11. Appel au module central de PEP_1 pour transmettre un rapport au PDP.
12. Envoi du rapport du client dans un message COPS RPT.
13. Réception et transfert du message COPS entier vers le module central du PDP après avoir retrouvé à quelle commande il se rapporte.
14. Extraction du contenu du message COPS et transfert de celui-ci vers le gestionnaire de contextes.
15. Stockage de la réponse de PEP_1 dans le contexte de la commande et appel au service adéquat pour traiter cette réponse.
16. Le service analyse du contenu du message COPS et en fonction de celui-ci fait évoluer le contexte de la commande et continue ou non son traitement par exemple en répétant les étapes 7 à

16 pour installer la politique dans les autres PEPs concernés…

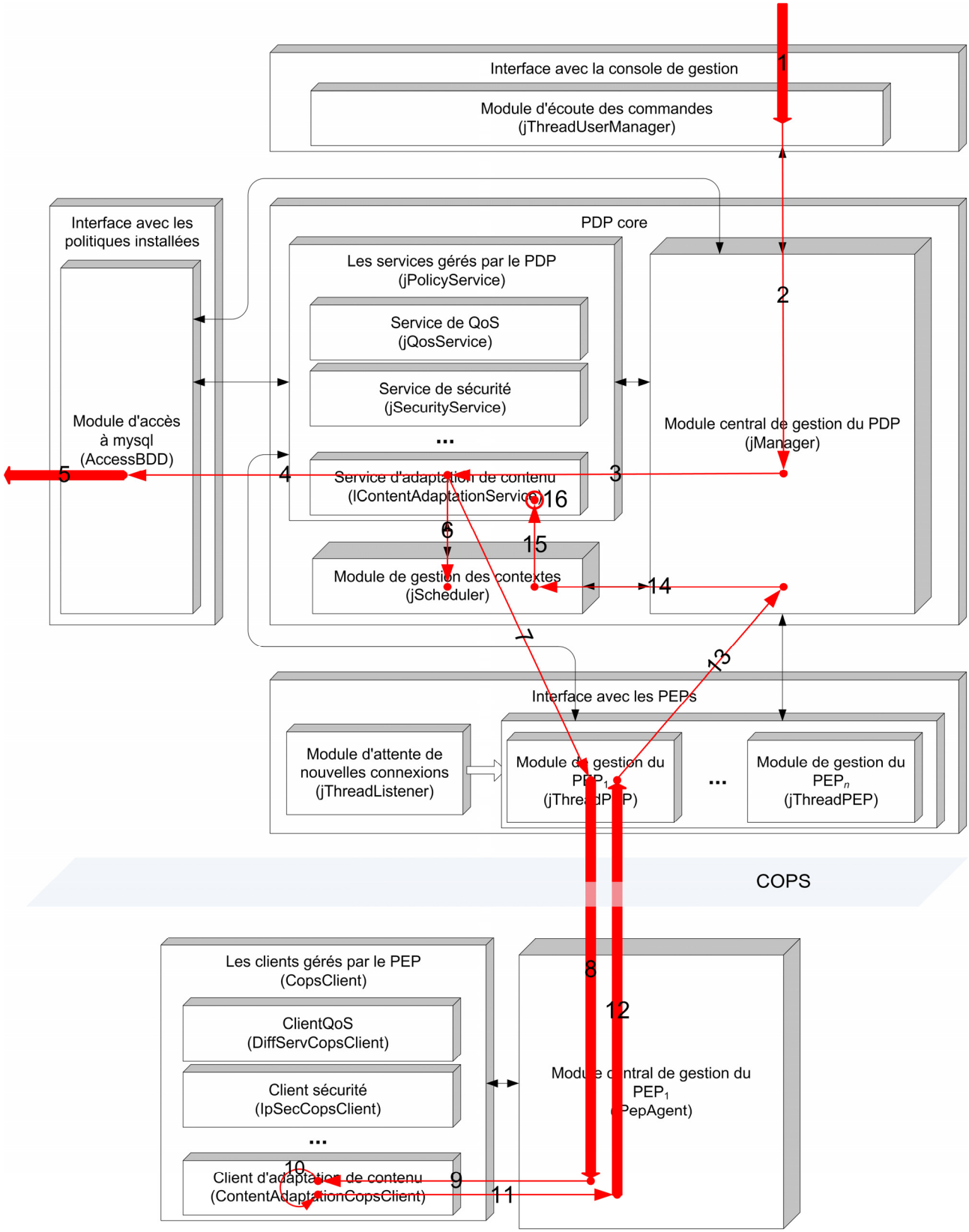


Figure 3-6 : Interaction entre les modules : installation d'une politique

♦ *Suppression d'une politique*

Les interactions entre les modules engendrées par une commande de suppression de politique sont très similaires à celles vues précédemment dans le cadre d'une commande d'installation de politique.

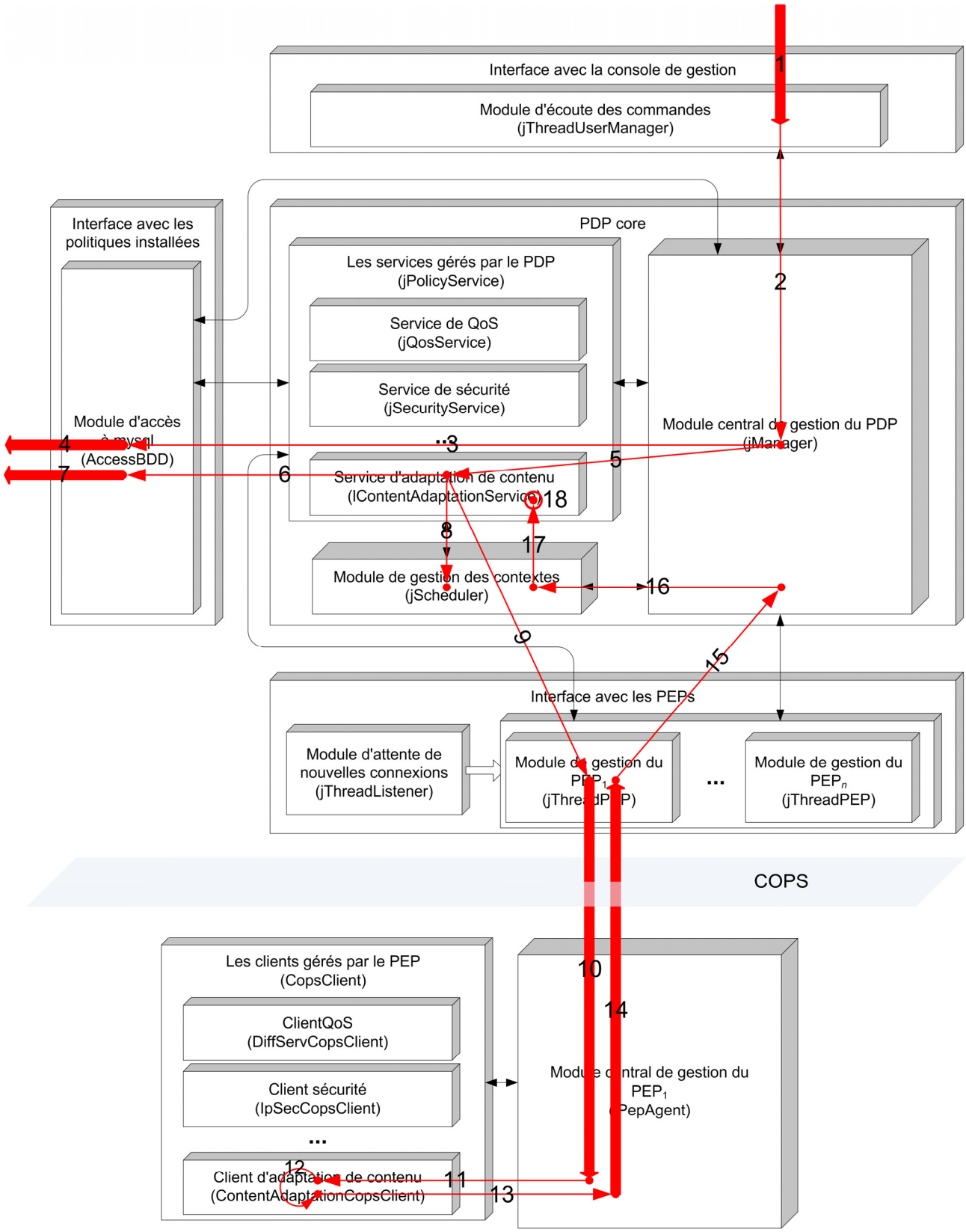


Figure 3-7 : Interaction entre les modules : suppression d'une politique

Seuls les huit premiers échanges différent de ceux décrits page 67 :

1. Réception de la commande par le module d'interfaçage.
2. Transfert de la commande au module central pour l'interpréter. Ce module reconnaît qu'il s'agit d'une commande de suppression et récupère l'identificateur de la politique à supprimer.
3. Appel au module d'interfaçage mysql pour retrouver le service concerné par cette politique.
4. Interrogation du *policy repository*.
5. Appel par le module central du service adéquat pour supprimer la politique.
6. Appel au module d'interfaçage mysql pour retrouver le contenu de la commande ayant installé cette politique et la liste des PEPs où elle a été appliquée.
7. Interrogation du *policy repository*.
8. Création d'un contexte de type *addReq* pour la suppression.
9. Suite identique excepté que le message COPS DEC envoyé est de type *remove*…

♦ *Commandes de contrôle à distance*

Les commandes de contrôle à distance servent à :

- Autoriser/interdire un PEP donné à se connecter au PDP.
- Autoriser/interdire toute nouvelle connexion au PDP.
- Déconnecter tous les PEPs (à partir du PDP) sans leur interdire de se reconnecter par la suite.
- Terminer le PDP.
- Lister tous les PEPs connectés au PDP.

Toutes ces commandes sont directement gérées par le module central du PDP et ne nécessitent pas d'interaction avec d'autres modules sauf pour la première commande où un message COPS CC doit être envoyé si le PEP non autorisé est déjà connecté.

2.3.1.2 Traitement des messages non sollicités du PEP

La plupart des messages envoyés au PDP par un PEP répondent à un précédent message émis par le PDP. Il existe tout de même deux situations où le PEP envoie des messages dits « non sollicités » :

- Lorsqu'il vient de (re)démarrer, le PEP demande au PDP, via un message COPS REQ, de le configurer, c'est à dire de lui envoyer toutes les politiques qu'il doit appliquer pour chacun de ses clients.
- Le PEP peut aussi envoyer des rapports de type *accounting* par exemple pour tenir le PDP au courant de l'évolution de sa charge.

Ce deuxième cas est possible mais n'est pas traité de manière très propre dans cette architecture. Le premier cas, lui, est plus intéressant et donne lieu aux échanges de Figure 3-8 :

1. Au démarrage du client, celui-ci appelle le module central du PEP d'envoyer une demande de configuration au PDP.
2. Envoi d'un message COPS REQ contenant éventuellement des informations spécifiques au client (dans un objet COPS ClientSI).
3. Transfert du message COPS au module central du PDP.
4. Analyse du message pour déterminer qu'il s'agit d'une demande de configuration à adresser au service adéquat.

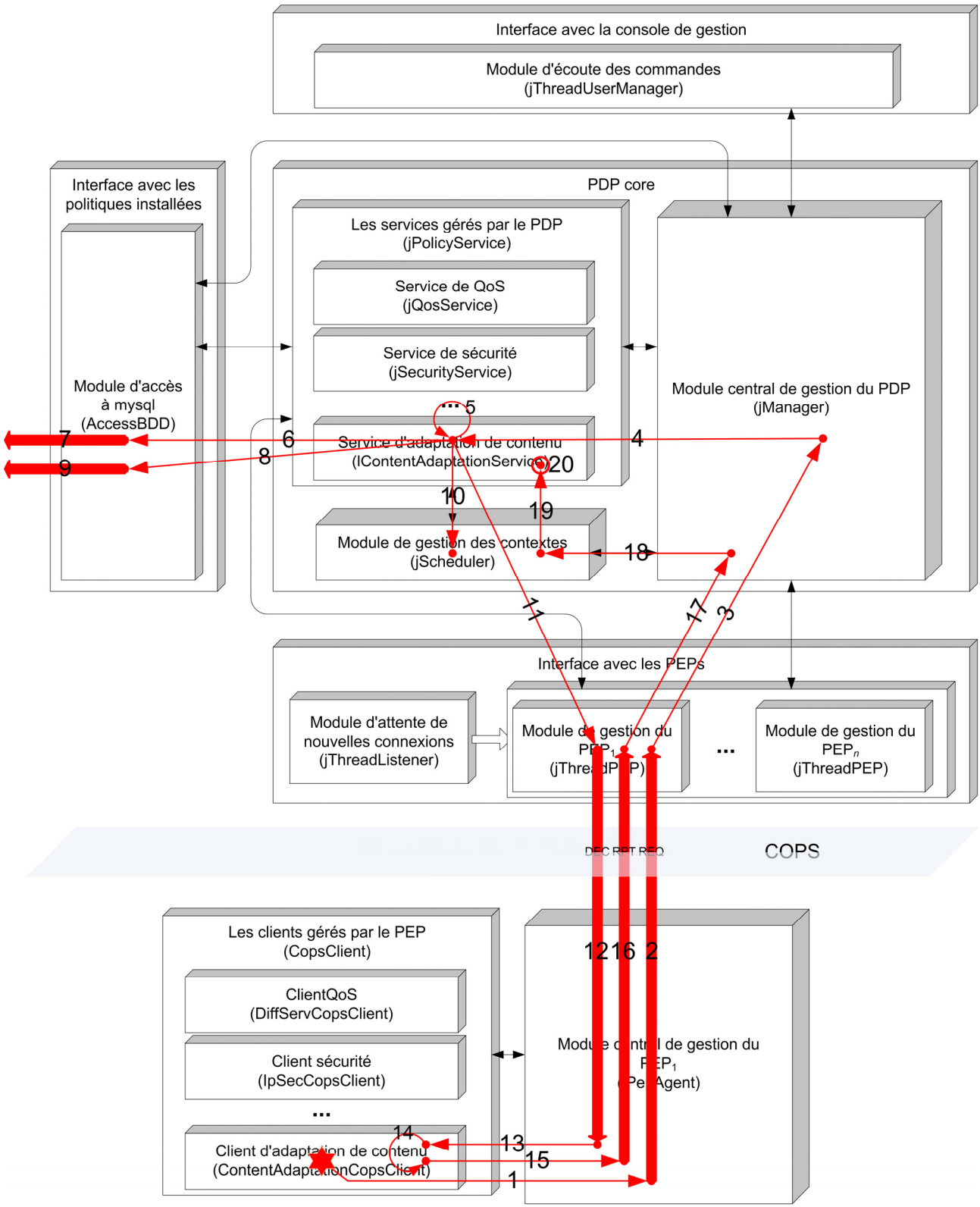


Figure 3-8 : Interaction entre les modules : demande de configuration

5. (Optionnel) Analyse du message à la recherche d'informations spécifiques au service.
6. Appel au module d'interfaçage mysql pour connaître la liste $\mathcal{L}_1$ des politiques à installer.
7. Interrogation du *policy repository*.

8. Choix d'une politique $\mathcal{P}_1$ dans la liste $\mathcal{L}_1$ et appel au module d'interfaçage mysql pour obtenir la description de cette politique et la liste des PEPs qu'elle concerne. Les étapes suivantes (8 à 20) seront répétées pour les autres politiques de $\mathcal{L}_1$.
9. Interrogation du *policy repository*.
10. Création d'un contexte de type *addReq* pour installer $\mathcal{P}_1$.
11. …

Les étapes 11 à 20 sont identiques aux étapes 7 à 16 de Figure 3-6 pour installer une politique depuis la console de gestion. Une fois la politique $\mathcal{P}_1$ installée, on revient à l'étape 8 pour installer la politique $\mathcal{P}_2$, etc…

Il faut aussi ajouter que normalement au démarrage d'un PEP, la demande de configuration décrite ci-dessus est réalisée pour chacun de ses clients.

2.3.2 Le concept de contexte

Dans la section précédente, le concept de contexte a été évoqué à plusieurs reprises. Il s'agit en effet d'un élément important dans le fonctionnement de cette architecture.

2.3.2.1 Définition

De manière générale, un contexte permet de savoir dans quel état se trouve une commande en cours de traitement. Un contexte est donc créé pour toute nouvelle commande émise par la console de gestion. Ce contexte évolue pendant le traitement de la commande en fonction de son 'bon' déroulement. Enfin, lorsque la commande a été complètement traitée, le contexte est détruit.

Un contexte contient principalement deux types d'informations :

- Des informations identifiant précisément la commande associée :
  - Identificateur unique de la commande. Cet identificateur correspond en fait au champ indexOrder de la table policyOrder de la base de données mysql (voir Figure 3-3 page 64).
  - Type de la commande : insertion, suppression ou modification.
  - Contenu de la commande, autrement dit la politique.
  - Service capable de traiter cette commande (QoS, sécurité…).
- Des informations sur le traitement de la commande :
  - Les réponses complètes des PEPs qui ont déjà été traités.
  - L'état courant dans l'automate fini décrivant tous les cas possibles dans le traitement de la commande.

La section suivante illustre comment les informations sur le traitement de la commande évoluent.

2.3.2.2 Fonctionnement

2.3.2.2.1 Principe

En fonction de son type et du service qui l'a créé, un contexte est associé à un automate fini déterministe avec un unique état initial. La transition d'un état à un autre déclenche l'appel d'une fonction spécifique au service. Les évènements engendrant les transitions correspondent aux réponses des PEPs (messages COPS RPT).

Lors de la création d'un contexte, son automate fini est initialisé et reçoit immédiatement l'événement « start » pour commencer le traitement de la commande.

2.3.2.2.2 Exemple

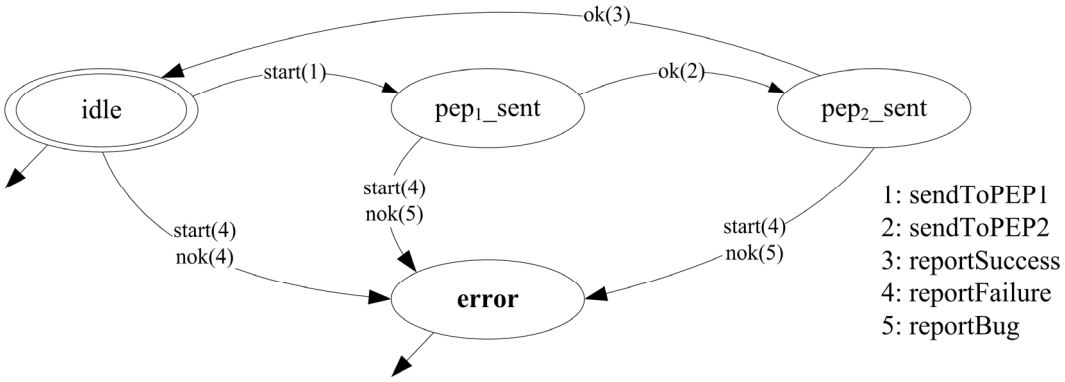


Figure 3-9 : Exemple d'automate fini associé à un contexte

L'automate fini de Figure 3-9 peut servir à installer une politique sur deux PEPs comme décrit Figure 3-10.

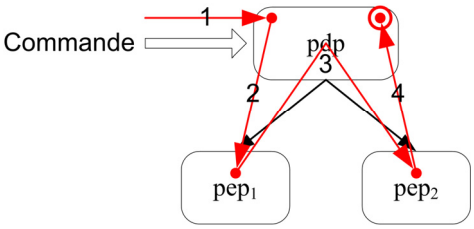


Figure 3-10 : Exemple de fonctionnement d'un automate fini

Si aucune erreur ne survient, le traitement de la commande d'installation se déroule comme suit :

1. Création d'un contexte et initialisation de l'automate représenté Figure 3-9.
2. Emission automatique de l'événement « start » déclenchant l'appel à la fonction « sendToPEP1 » qui installe la politique dans PEP_1.
3. Réponse de PEP_1. Cette réponse est stockée dans le contexte pour être éventuellement utilisée ultérieurement. La réponse étant de type *success*, elle engendre l'événement « ok » déclenchant l'appel à la fonction « sendToPEP2 » qui installe la politique dans PEP_2.
4. Réponse de PEP_2. Cette réponse est stockée dans le contexte. La réponse étant de type *success*, elle engendre l'événement « ok » déclenchant l'appel à la fonction « reportSuccess » qui fait éventuellement un dernier traitement avant de détruire le contexte.

3 Le service contentAdaptation

Pendant mon stage, j'ai mis en place un service réseau d'adaptation de contenu basé sur COPS. Ce service, appelé contentAdaptation, a été développé sur l'architecture PDP/PEP du projet RTIPA décrite dans la section précédente.

3.1 Caractéristiques

Compte tenu des contraintes du projet RHODOS, le service contentAdaptation doit être à la fois puissant et souple.

3.1.1 Puissance

Plusieurs aspects permettent de dire que ce service est puissant.

3.1.1.1 Gestion de politiques complexes

Jusqu'ici, les services développés sur l'architecture du projet RTIPA ne géraient que des politiques relativement simples adressées à un ou deux PEP(s). Le service contentAdaptation utilise, lui, des politiques qui s'adressent à un nombre quelconque de PEPs et qui peuvent comporter plusieurs ordres élémentaires.

La figure ci-dessous illustre comment une politique complexe pourrait être utilisée dans le projet RHODOS.

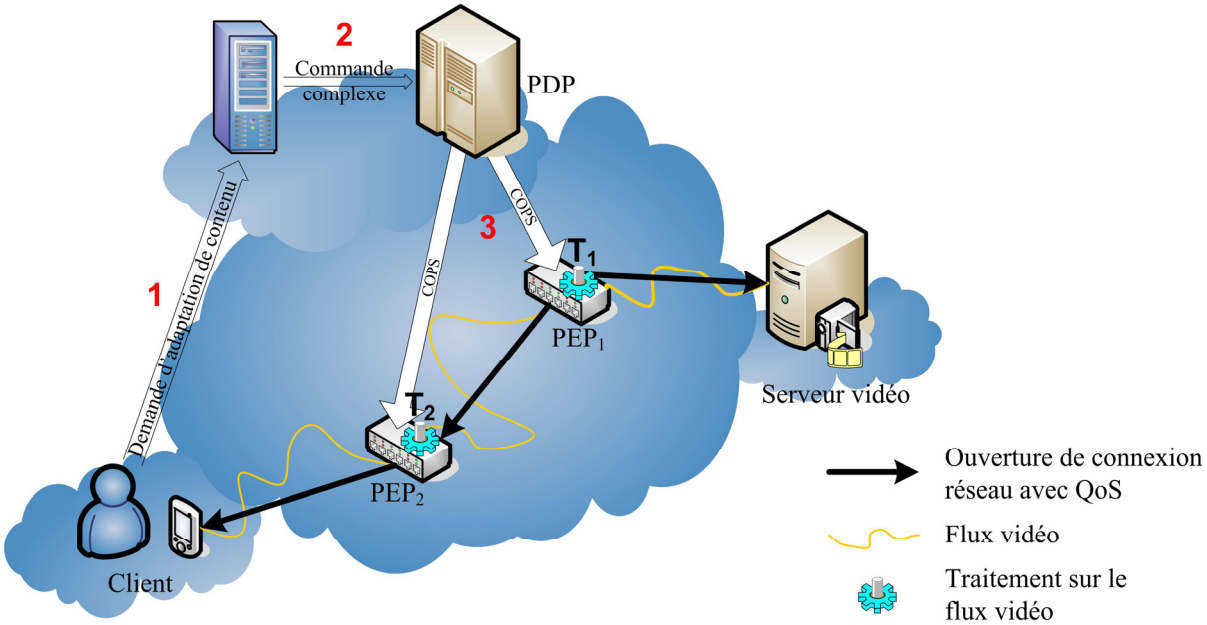


Figure 3-11 : Exemple de politique complexe

Voici les détails du scénario (réaliste) représenté ci-dessus :

1. Le client s'adresse à une entité du réseau (à définir) pour recevoir un flux vidéo en lui précisant le traitement à effectuer dessus et la QoS désirée.
2. Cette entité calcule où le flux doit être traité. Par exemple ici, il doit être traité en deux fois : dans PEP_1 puis dans PEP_2. Il formule donc une politique complexe indiquant que :
   - PEP_1 doit :
     - ouvrir une connexion avec QoS vers le serveur vidéo
     - ouvrir une connexion avec QoS vers PEP_2
     - effectuer le traitement T_1 sur le flux transitant par ces connexions.
   - PEP_2 doit :
     - ouvrir une connexion avec QoS vers le client
     - effectuer le traitement T_2 sur le flux provenant de PEP_1.
3. Le PDP analyse cette politique, la segmente en ordres élémentaires et envoie ceux-ci aux PEPs concernés.

Il faut signaler que les politiques complexes ont un « défaut » : elles ne permettent pas de spécifier dans quel ordre les ordres élémentaires doivent être appliqués. En réalité, cela n'a généralement pas d'importance car, si on

reprend l'exemple précédent, le client ne reçoit la vidéo qu'il a demandé que lorsque tous les ordres élémentaires sont installés quelque soit la manière dont est traitée la politique complexe.

3.1.1.2 Mécanisme de rollback

Evidemment en soi une politique complexe n'a que très peu d'intérêt puisqu'il est tout à fait possible d'envoyer à la place une série d'ordres élémentaires, d'autant plus que cela évite le temps de calcul nécessaire pour construire la politique complexe.

Cependant, le service contentAdaptation possède un mécanisme de *rollback* pour les commandes d'installation et de modification. Cela garantit qu'en cas d'erreur lors du traitement de la commande, toutes les modifications partielles du réseau sont supprimées. Autrement dit, les ordres élémentaires déjà appliqués sont annulés.

De ce fait, le concept de politique complexe se justifie pleinement. On comprend notamment pourquoi le scénario de la section précédente utilise une seule politique complexe plutôt qu'un ensemble d'ordres élémentaires. En effet, l'impossibilité d'appliquer un seul de ces ordres rend inutile l'installation de tous les autres.

3.1.1.3 Modification des politiques sans réinstallation

Le service contentAdaptation offre la possibilité de modifier une politique installée. Cette caractéristique, bien qu'elle puisse sembler naturelle, est en réalité souvent implémentée dans les architectures basées sur COPS par une désinstallation suivie d'une réinstallation. En effet, le protocole COPS ne définit que des messages DEC de type *install*, *remove* et *null*. Il n'existe pas de message DEC de type *modify*.

Malgré cette limitation de COPS, il est important que le service contentAdaptation possède un vrai mécanisme de modification des politiques installées. Pour en comprendre les raisons, il faut savoir que, comme le montre l'exemple de scénario Figure 3-11, ce service s'occupe à la fois du traitement des flux vidéo et de la gestion des connexions avec QoS. La qualité de service est réalisée par un module extérieur (développé dans le sous-projet 4 de RHODOS) qui est piloté par le service contentAdaptation. Or ce module extérieur offre la possibilité de modifier dynamiquement la classe de services des connexions qu'il a mis en place. Cette fonction est très intéressante puisqu'on se trouve sur un réseau sans fils combinant des technologies hétérogènes où les clients sont mobiles. Donc pour pouvoir en bénéficier, il est indispensable de pouvoir modifier une politique installée sans avoir à la supprimer préalablement ce qui engendrerait une coupure dans le flux vidéo fort peu agréable pour le client.

3.1.2 Souplesse

Bien que puissant, le service contentAdaptation se doit aussi d'être souple. La principale raison à cela est qu'il a été développé sans connaître les caractéristiques des modules qu'il devra piloter.

3.1.2.1 Langage des politiques libre

Afin de pouvoir dialoguer avec n'importe quel type de module, il a été mis en place une sorte de langage 'haut niveau' basé sur un système de fichiers :

- Ces fichiers sont appelés « fichiers de politique » ou *policy files*.

- Le nom de ces fichiers (*file name*) désigne leur fonction. Ex : « connection.net » désigne une politique relative aux connexions avec QoS. Il est possible d'associer des options avec ce nom pour préciser la politique, comme par exemple le niveau de la QoS désirée.
- Le contenu de ces fichiers (*data file*) décrit la politique dans un langage 'bas niveau' interprétable par le module qui devra appliquer la politique. Il n'y a, a priori, aucune restriction sur le choix de ce langage bas niveau. Un fichier de politique peut contenir du XML, une ligne de commande, un script, du code binaire…

Ainsi, la politique est exprimée au niveau de la console de gestion dans ce langage 'haut niveau' utilisant le nom des fichiers de politique. Elle est transmise sous une forme similaire au PDP puis au PEP qui utilise alors le contenu de ces fichiers pour appliquer la politique.

3.1.2.2 Interfaçage facile avec le PEP

D'autre part, afin de pouvoir intégrer facilement n'importe quel module extérieur, le client contentAdaptation au niveau du PEP utilise une interface très générique basé sur des *sockets*.

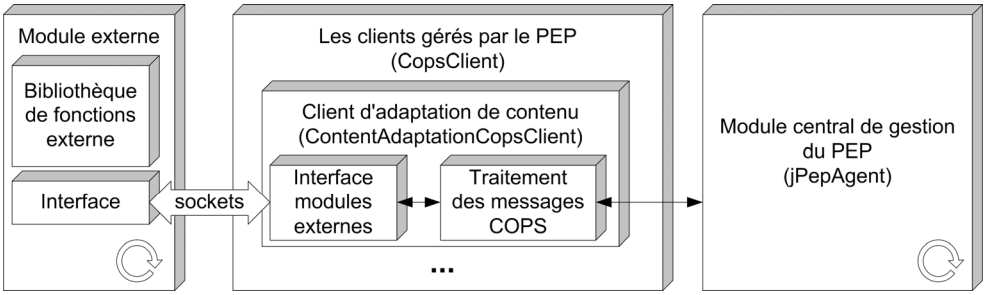


Figure 3-12 : Utilisation d'une bibliothèque de fonction extérieur au service contentAdaptation

Ainsi quelque soit la forme sous laquelle le module extérieur est fourni (application, bibliothèque en C, C++, Java…), il est simple de l'utiliser à partir du service.

3.2 Implémentation

La mise en place du service contentAdaptation a évidemment donné lieu au développement de deux nouveaux modules : un service pour le PDP et un client pour les PEPs. Cependant, compte tenu des caractéristiques de ce service, il a également fallu modifier légèrement le module central du PDP pour qu'il prenne en compte de nouvelles commandes et étendre le protocole COPS avec trois nouveaux objets.

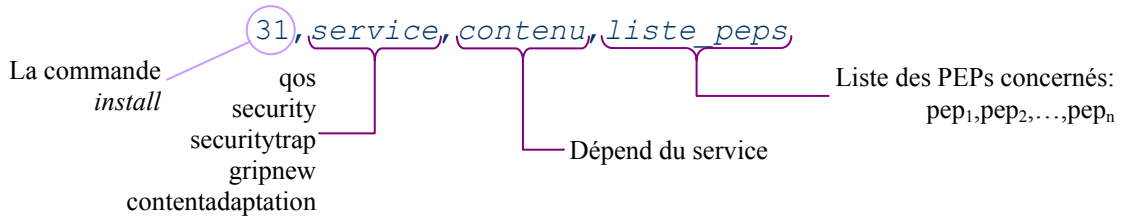
3.2.1 De nouvelles commandes

3.2.1.1 Commande d'installation

Les anciens services ne gérant pas plus de deux PEPs par politique, il a été nécessaire de rajouter une deuxième commande d'installation plus générale prenant en compte un nombre non défini à l'avance de PEPs.

♦ *Structure de la commande*

La principale différence entre la structure de l'ancienne et de la nouvelle commande est que la liste des PEPs concernés par la politique se trouve à la fin :



♦ *Utilisation dans le service contentAdaptation*

Dans le service contentAdaptation, le contenu de la commande d'installation doit être formaté de la façon suivante :

$contenu := tâche_1 | tâche_2 | … | tâche_p$

$tâche_i := file\_name=xxx; opt_{i.1}=val_{i.1}; …; opt_{i.q}=val_{i.q}; pep_{i.1}=true; …; pep_{i.m}=true$

Nom d'un fichier de politique — Liste d'options propres à chaque fichier de politique — Liste des PEPs concernés

Le contenu de la commande est constitué de une ou plusieurs tâche(s). Une tâche correspond à un ordre élémentaire destiné à un ou plusieurs PEP(s). La liste des PEPs d'une tâche est un sous-ensemble de la liste des PEPs de la commande.

Il est donc assez évident qu'avec une telle structure, il est possible de définir des politiques très complexes comme l'illustre l'exemple suivant.

♦ *Exemple*

```
31,                                                                    ← Commande d'installation
contentadaptation,                                                     ← Service demandé
file_name=monitor.scr;type=CPU;venus.tai=true;saturne.tai=true;gamma=true|
file_name=monitor.scr;type=BP;saturne.tai=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1234;dst_2=gamma:1234;venus.tai=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1235;dst_2=saturne:1234;redim=0.66;gamma=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1236;redim=0.66;saturne.tai=true,
venus.tai,saturne.tai,gamma                                            ← Liste des PEPs concernés par la commande
```

Liste des tâches

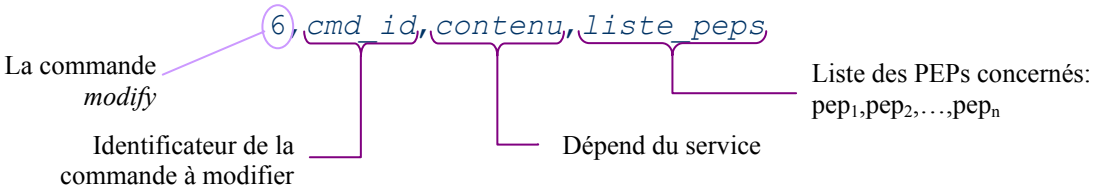
L'exemple précédent permet d'installer :

- Sur le PEP gamma :
  - Une console de *monitoring* du CPU.
  - Un processus VLC qui écoute un flux vidéo sur le port 1234, redimensionne sa taille par 2/3 et le redirige vers les machines saturne et tanith.
- Sur le PEP saturne :
  - Une console de *monitoring* du CPU.
  - Une console de *monitoring* de la bande passante.
  - Un processus VLC qui écoute un flux vidéo sur le port 1234, redimensionne sa taille par 2/3 et le redirige vers tanith.
- Sur le PEP venus :
  - Une console de *monitoring* du CPU.
  - Un processus VLC qui écoute un flux vidéo sur le port 1234 et le redirige vers tanith et gamma.

3.2.1.2 Commande de modification

Puisqu'il n'existait pas de commande de modification, il a fallu en créer une qui, comme la commande d'installation, puisse prendre en compte un nombre non défini à l'avance de PEPs.

♦ *Structure de la commande*



L'identificateur d'une commande correspond au champ indexOrder du *policy repository* (Figure 3-3). Il l'identifie de manière unique.

♦ *Utilisation dans le service contentAdaptation*

Dans le service contentAdaptation, le contenu de la commande de modification doit être formaté de la façon suivante (les retour à la ligne « ↵ » ont été rajoutés pour une meilleure lisibilité) :

`contenu:=tâche_add`$_1$`|…|tâche_add`$_p$`% ↵`
`tâche_del`$_1$`|…|tâche_del`$_q$`% ↵`
`tâche_chg`$_{1\_old}$`|tâche_chg`$_{1\_new}$`|…|tâche_chg`$_{r\_old}$`|tâche_chg`$_{r\_new}$

On retrouve le concept de tâche déjà vu avec la commande d'installation. Cette fois, les tâches sont regroupées en trois blocs (éventuellement vides) séparés par « % » :

- Le premier bloc contient toutes les tâches à ajouter par rapport à l'ancienne commande.
- Le deuxième bloc contient des tâches de l'ancienne commande à supprimer.
- Le dernier bloc contient des paires de tâches afin de spécifier sans ambiguïté quelles tâches modifier et comment.

♦ *Exemple*

```
6,
xxx,
file_name=vlc.scr;rcv_port=@:1235;dst_1=tanith:1235;dst_2=saturne:1234;redim=0.66;saturne.tai=true%
file_name=monitor.scr;type=CPU;gamma=true;saturne.tai=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1235;dst_2=saturne:1234;redim=0.66;gamma=true%
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1234;dst_2=gamma:1234;venus.tai=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1234;dst_2=saturne:1235;venus.tai=true,
venus.tai,saturne.tai,gamma
```

← Commande de modification
← Commande à supprimer
← Liste des PEPs concernés par la commande
Liste des tâches

La commande de cet exemple correspond, dans cet ordre, aux ordres élémentaires suivants :

- Installation :
  - Sur le PEP saturne :
    - D'un module de traitement VLC
- Suppression :
  - Sur le PEP gamma :
    - D'une console de *monitoring* du CPU
    - D'un module de traitement VLC

- Sur le PEP saturne :
  - D'une console de *monitoring* du CPU
- Modification :
  - Sur le PEP venus :
    - D'un module de traitement VLC pour rediriger son flux de sortie de gamma vers saturne.

3.2.2 Un service PDP complexe

Les mécanismes fournissant au service contentAdaptation la puissance qui a été évoquée dans la section 3.1.1 sont principalement implémentés au niveau du PDP dans le nouveau module de service. Ce module est donc relativement complexe.

Sans rentrer dans les détails de l'implémentation, les sections suivantes décrivent comment sont traitées les commandes de suppression, d'installation et de modification.

3.2.2.1 Fonctionnement de la suppression

La commande de suppression est la plus simple des commandes car elle ne possède pas de mécanisme de *rollback*. En fait, s'il n'est pas possible de supprimer un ordre élémentaire d'une politique, l'information est remontée pour être éventuellement traitée, mais le PDP continue à désinstaller les autres ordres élémentaires. C'est un comportement naturel qui est préférable dans la majorité des situations.

Le traitement des commandes de suppression doit donc :

- Gérer un nombre non défini à l'avance de PEPs.
- Signaler toutes les erreurs survenues en cas de problème.

L'automate fini implémenté, représenté Figure 3-13, répond à ces deux besoins.

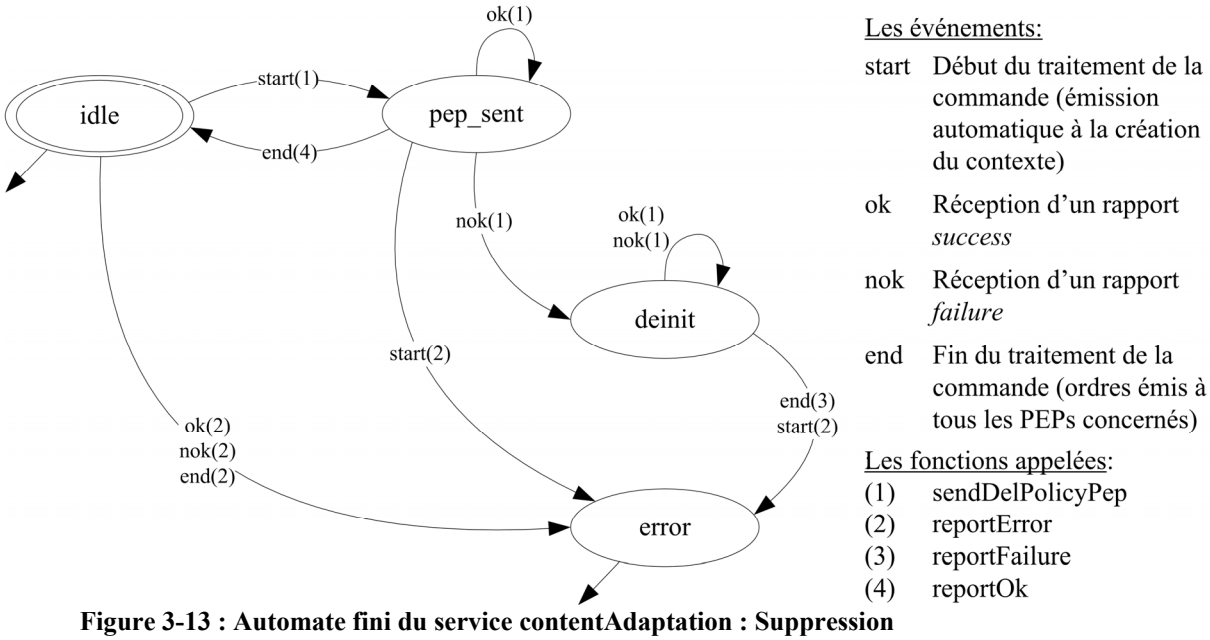


Figure 3-13 : Automate fini du service contentAdaptation : Suppression

♦ *Fonctionnement nominal*

Afin d'illustrer comment cet automate fini gère un nombre non défini à l'avance de PEPs, examinons la suppression d'une politique installée sur quatre PEPs dans le cas nominal.

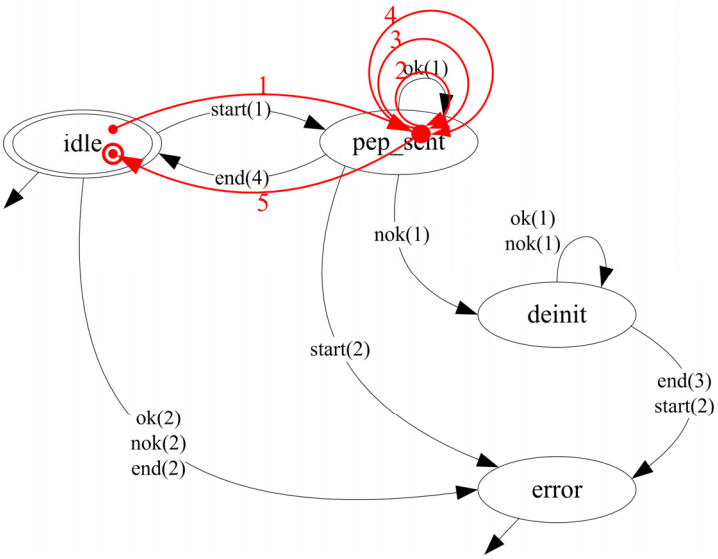


Figure 3-14 : Exemple de suppression d'une politique sans erreur

Si aucune erreur ne survient, le traitement d'une commande de suppression se déroule entièrement dans l'état « pep_sent » (étapes 1 à 4). L'automate ne sort de cet état qu'en recevant l'événement « end » qui signifie que la commande a été complètement traitée (étape 5).

Au cours des étapes 1 à 4, c'est toujours la même fonction « sendDelPolicyPep » qui est appelée. Cette fonction commence par rechercher (grâce aux réponses des PEPs stockées dans le contexte) un PEP à qui l'ordre de suppression n'a pas encore été envoyé puis il effectue un traitement identique pour tous les PEPs.

Ce principe de fonctionnement qui est utilisé dans tous les autres automates finis du service contentAdaptation garantit de pouvoir gérer un nombre quelconque de PEPs.

♦ *En cas d'erreur*

Figure 3-15 est aussi un exemple de suppression d'une politique installée sur quatre PEPs (PEP_1 à PEP_4). Cependant, cette fois, une erreur survient au niveau de PEP_2 (en supposant que les PEPs sont traités dans l'ordre $PEP_1 \rightarrow PEP_2 \rightarrow PEP_3 \rightarrow PEP_4$) qui n'arrive pas à désinstaller au moins un ordre élémentaire.

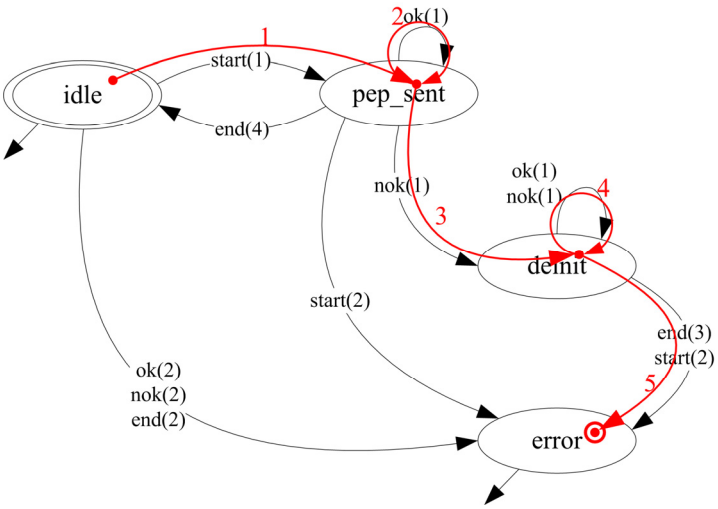


Figure 3-15 : Exemple de suppression d'une politique avec erreur

Cette erreur, comme celles qui pourraient se produire par la suite dans PEP_3 et PEP_4, est mémorisée dans le contexte au niveau des réponses des PEPs. De plus, l'automate fini passe dans l'état « deinit ». Dans cet état, on continue à désinstaller la politique dans les PEPs qui n'ont pas encore reçu l'ordre de suppression (uniquement PEP_4 ici). L'automate fini se comporte donc comme s'il se trouvait encore dans l'état « pep_sent », cependant le fait d'être dans l'état « deinit » indique qu'il s'est produit au moins une erreur. Ainsi, une fois que tous les PEPs ont été traités et que l'événement « end » ait été émis (étape 5), c'est la fonction « reportFailure » qui est appelée à la place de « reportOk ». Dans cette fonction, il est possible d'effectuer un traitement sur les erreurs survenues.

3.2.2.2 Fonctionnement de l'installation

La commande d'installation est plus complexe que celle de suppression. Elle possède les mêmes caractéristiques :

- Gestion d'un nombre non défini à l'avance de PEPs.
- Remontée du maximum d'information sur les erreurs survenues.

Plus d'autres :

- Mécanisme de *rollback* permettant d'annuler l'installation dès qu'un seul ordre élémentaire ne peut être appliqué. Le *rollback* se comporte exactement de la même façon que la commande de suppression : si certains ordres élémentaires ne peuvent être désinstallés, l'information est mémorisée mais le processus continue sans tenir compte des erreurs.
- Traitement sélectif des erreurs. Pendant la phase d'installation, les erreurs de type « il manque un ou plusieurs fichier(s) de politique » engendre l'émission l'événement « file ». Cette erreur est immédiatement traitée en renvoyant au PEP adéquat les fichiers de politique manquants. Elle n'est donc pas considérée comme une 'erreur' et fait partie du fonctionnement nominal de la commande.

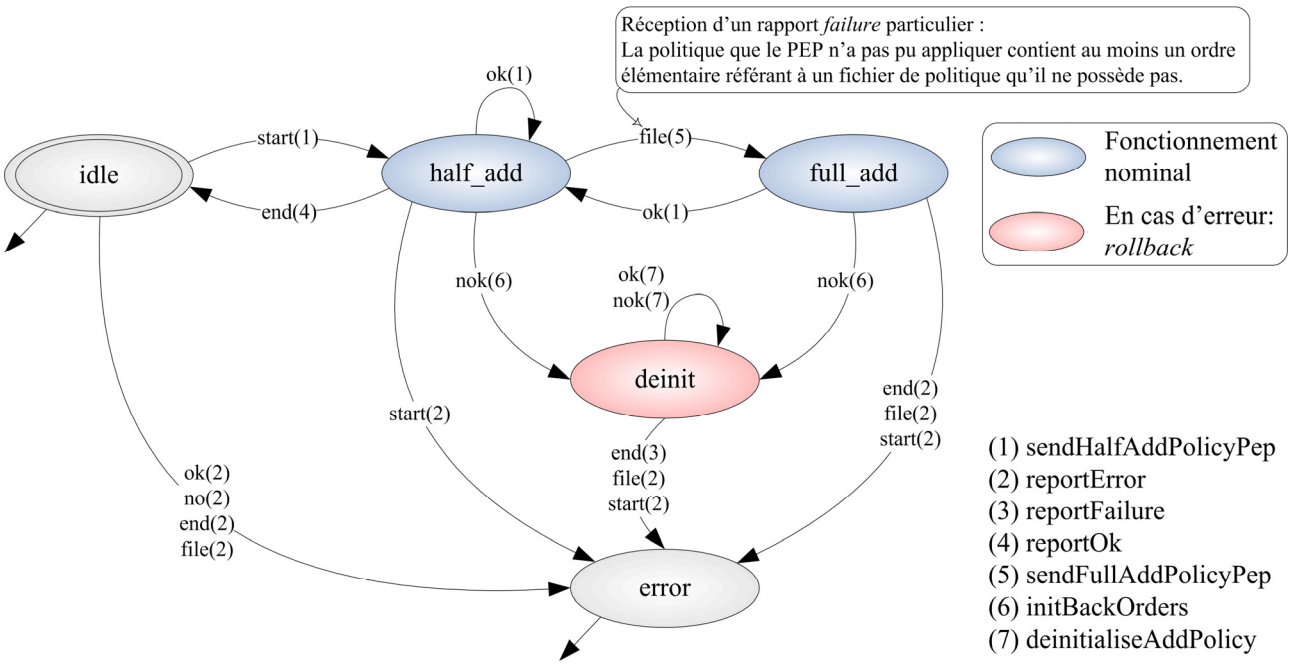


Figure 3-16 : Automate fini du service contentAdaptation : Installation

Sur cet automate fini, on peut faire plusieurs remarques :

- La gestion d'un nombre quelconque de PEPs est réalisée de la même manière que dans l'automate fini de suppression (voir section précédente).
- Le traitement éventuel des erreurs à la fin de la commande est aussi fait par la fonction « reportFailure » en s'appuyant sur les réponses des PEPs stockées dans le contexte (voir section précédente).
- L'état « full_add » sert à savoir, pour un PEP donné, qu'il a déjà envoyé un rapport *failure* de type « fichier(s) de politique manquant(s) ». Ce rapport doit contenir la liste exhaustive des fichiers de politique manquants. Donc si l'automate reçoit de nouveau un événement « file », cela signifie qu'il y a un bug dans le service (appel de la fonction « reportError »).
- Le mécanisme de *rollback* est assez simple puisqu'il ne nécessite qu'un seul état (« deinit »).

3.2.2.3 Fonctionnement de la modification

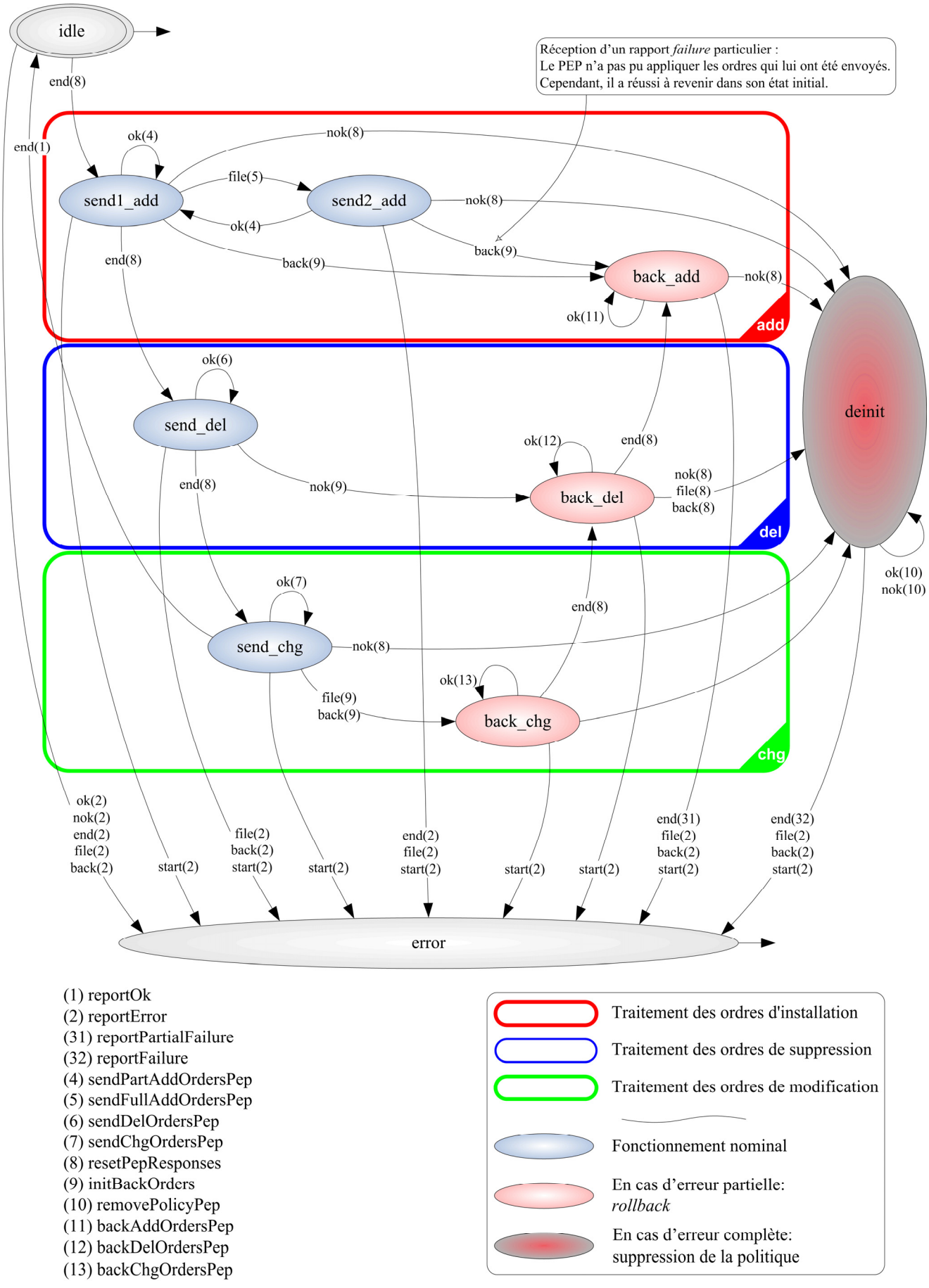
La commande de modification est de loin la plus complexe. A titre indicatif, elle représente, dans le service contentAdaptation du PDP, environ deux fois plus de lignes de code que les commandes d'installation et de suppression réunies.

Une telle complexité se comprend assez facilement étant donné le nombre de possibilités offertes par cette commande qui permet de changer une politique :

- en ajoutant de nouveaux ordres élémentaires éventuellement destinés à des PEPs qui ne faisaient pas partie de la commande initiale ;
- en supprimant et modifiant d'anciens ordres élémentaires.

♦ *Automate fini*

Cette complexité se retrouve dans l'automate fini Figure 3-17.



(1) reportOk
(2) reportError
(31) reportPartialFailure
(32) reportFailure
(4) sendPartAddOrdersPep
(5) sendFullAddOrdersPep
(6) sendDelOrdersPep
(7) sendChgOrdersPep
(8) resetPepResponses
(9) initBackOrders
(10) removePolicyPep
(11) backAddOrdersPep
(12) backDelOrdersPep
(13) backChgOrdersPep

Traitement des ordres d'installation
Traitement des ordres de suppression
Traitement des ordres de modification

Fonctionnement nominal
En cas d'erreur partielle: *rollback*
En cas d'erreur complète: suppression de la politique

Figure 3-17 : Automate fini du service contentAdaptation : Modification

La première remarque à faire sur cet automate fini est que sa structure reflète assez fidèlement celle du contenu de la commande de modification (voir section 3.2.1.2 page 78) où les tâches sont réparties en trois blocs : installation, puis suppression, puis modification. Dans l'automate fini, ces trois blocs de tâches correspondent à trois blocs d'états qui se suivent dans le même ordre.

La principale conséquence de cette structure est qu'en cas d'erreur, il n'est pas possible de remonter autant d'informations que dans les automates finis vus précédemment. En effet, la transition d'un des blocs d'états à un autre, déclenche toujours un appel à la fonction « resetPepResponses » qui, comme son nom l'indique, supprime toutes les réponses des PEPs stockées dans le contexte de la commande.

Cette structure rend aussi le mécanisme de *rollback* plus complexe à mettre en œuvre. Il est ainsi réparti sur trois états : un état dans chaque bloc. A la détection de la première erreur (autre que « fichiers de politique manquants »), l'automate fini bascule de l'état « send_xxx » vers « back_xxx » où « xxx » correspond à un bloc d'états (« add », « del » ou « chg »). Si tout se déroule correctement, le *rollback* continue en 'remontant' vers l'état « back_add » qui finit par recevoir l'événement « end » ce qui signifie que tous les effets de la commande de modification ont été annulés. Le PDP se retrouve donc à nouveau dans un état parfaitement stable.

Cependant, il est possible que le *rollback* rencontre des problèmes, par exemple parce que les ressources qui viennent d'être libérées ont déjà été réallouées à un autre client. Comme il est impossible d'envisager toutes les erreurs pouvant survenir pendant le *rollback*, si celui-ci rencontre la moindre difficulté, il cède sa place à un mécanisme de désinstallation brutale de la politique. Concrètement, cela signifie que l'automate fini passe d'un état « back_xxx » à l'état « deinit ». Dans cet état, pour chaque PEP, la fonction « removePolicyPep » liste tous les ordres installés par la politique initiale ou la commande de modification et demande au PEP de les supprimer.

♦ *Interaction entre les modules*

La structure de l'automate fini influe également beaucoup sur les interactions entre les modules comme on peut le voir Figure 3-18 où les étapes 7 à 16 sont exécutées trois fois : d'abord pour installer des ordres élémentaires, puis en supprimer et enfin en modifier. Il faut d'ailleurs remarquer que les messages COPS DEC transportant les modifications sont du type *null* qui sera donc appelé, dans le service contentAdaptation, *modify*.

Les étapes 1 à 4 servent à analyser la commande pour déterminer

- qu'il s'agit d'une commande de modification
- qu'elle s'adresse au service contentAdaptation.

Enfin les étapes 17 et 18 ne sont utilisées qu'en cas de succès. Elles permettent de modifier l'ancienne commande dans le *policy repository*.

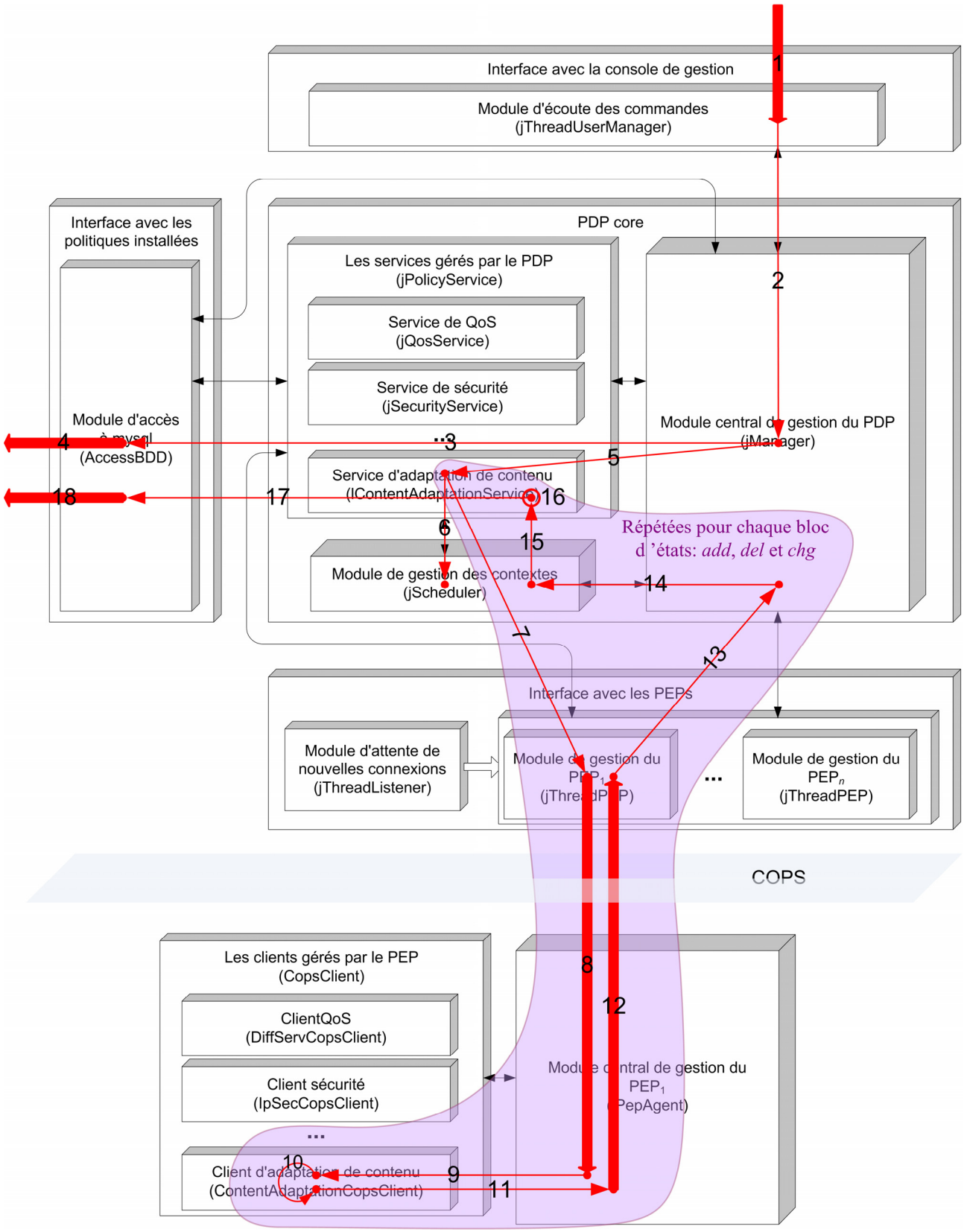


Figure 3-18 : Interaction entre les modules : modification d'une politique

3.2.3 Un client PEP souple

La section précédente a présenté tout le travail effectué au niveau du service contentAdaptation du PDP. Cela a permis d'y inclure :

- Une gestion des politiques complexes
- Un mécanisme de *rollback*
- Une commande de modification sans réinstallation

En réalité, ces caractéristiques ont aussi demandé du travail dans le client contentAdaptation du PEP, mais dans des proportions moindres. Ce client est surtout remarquable pour sa souplesse.

3.2.3.1 Conception modulaire

Cette souplesse est due à sa conception modulaire.

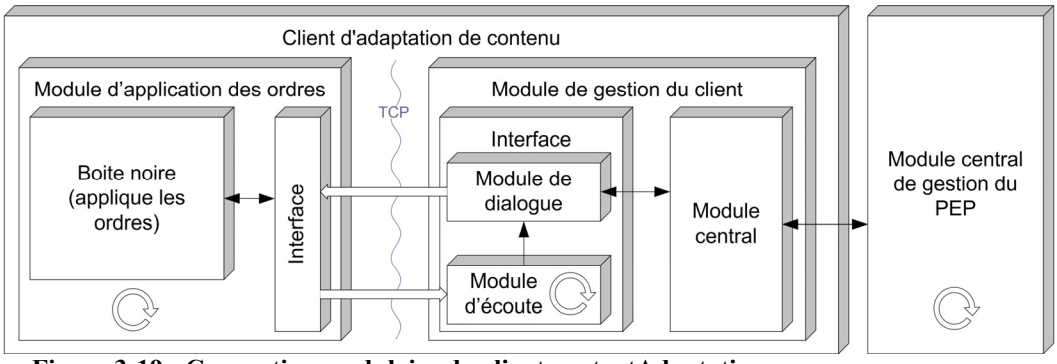


Figure 3-19 : Conception modulaire du client contentAdaptation

Sur Figure 3-19, on peut voir que le client contentAdaptation est constitué de deux principaux modules :

- Un module de gestion du client qui est implémenté directement dans l'entité « pep ». Son rôle est d'interpréter le contenu des objets COPS Decision venant du PDP. Il met ensuite tout en œuvre pour faire appliquer chaque ordre élémentaire interprété. Il possède également un mécanisme de *rollback* local.
- Un module d'application des ordres élémentaires appelé *actioner*. Il gère chaque ordre indépendamment des autres (pas de vision globale de la politique). Il possède une interface pour être piloté par le module de gestion du client via un protocole propriétaire basé sur TCP.

Cette séparation des fonctions décisionnelles et exécutives permet de ne pas implémenter l'*actioner* directement dans le PEP. Le principal avantage de ce choix est qu'il devient très facile d'utiliser des *actioners* développés indépendamment. L'utilisation de TCP supprime notamment les difficultés d'interaction entre différents langages de programmation.

L'inconvénient de faire dialoguer les deux modules par TCP est de ralentir le client. Cependant aucune baisse sensible de performances n'a été ressentie pendant les différents tests. Cette solution est donc tout à fait satisfaisante pour la plateforme du projet RHODOS.

3.2.3.2 Fonctionnement

Le fonctionnement du client contentAdaptation est assez simple et identique pour les commandes d'installation, suppression et modification.

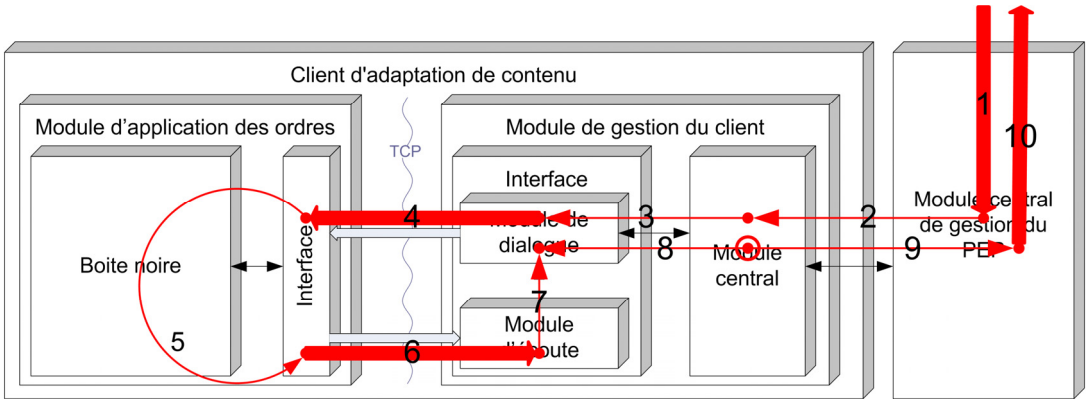


Figure 3-20 : Interaction entre les modules du client contentAdaptation

Voici les détails des interactions entre les modules du client :

1. Réception d'un message COPS DEC adressé au client contentAdaptation.
2. Transmission du contenu du message au module central de gestion.
3. Après analyse de ce contenu qui contient *n* ordres élémentaires, le module central de gestion appelle le module de dialogue pour faire appliquer un de ces ordres par l'*actioner*. Les étapes 3 à 8 sont exécutées pour chacun des *n* ordres.
4. Envoi de l'ordre à l'*actioner*.
5. Application (installation, suppression ou modification) de l'ordre.
6. Envoi d'un rapport au module d'écoute. Les rapports d'échec peuvent contenir une description de l'erreur survenue.
7. Transmission de la réponse de l'*actioner* au module de dialogue.
8. Consultation de la réponse de l'*actioner*. Si celle-ci correspond à une erreur, le module central du client annule les ordres déjà traités et stoppe son traitement. Dans le cas contraire, il continue à appliquer les autres ordres jusqu'à ce qu'il n'y en ait plus.
9. Transmission d'un rapport au module central du PEP.
10. Envoi d'un message COPS RPT contenant le rapport du client.

3.2.4 Une extension de COPS : COPS-CA

Un des objectifs du service contentAdaptation est de gérer des politiques complexes. Les sections précédentes ont montré qu'il en est capable aussi bien au niveau du PDP que du PEP. Maintenant, il est intéressant de décrire les extensions qu'il a fallu ajouter au protocole COPS pour transporter ces politiques complexes.

3.2.4.1 Nouveaux objets

Tout d'abord, il faut rappeler qu'une décision du PDP envoyée au PEP dans un objet COPS Decision.

Dans le service contentAdaptation, cet objet COPS Decision doit nécessairement contenir les ordres élémentaires à installer. Par exemple, `file_name=monitor.scr;type=CPU;context_id=xxx`. Cependant cela n'est pas suffisant. Il faut aussi pouvoir y mettre le contenu du fichier de politique (voir section 3.1.2.1 page 75). De plus, comme un fichier de politique peut être volumineux, il serait intéressant de remplacer son contenu par une courte signature dès que cela est possible.

Trois objets COPS ont donc été créés pour répondre à ces besoins :

- Un objet Digest (`CA_DIGEST`) transporte la signature d'un fichier de politique.
- Un objet FileData (`CA_FDATA`) transporte le contenu d'un fichier de politique.
- Un objet Order (`CA_ORDER`) transporte un ordre élémentaire.

Ces trois objets sont structurés comme indiqué Figure 3-21.

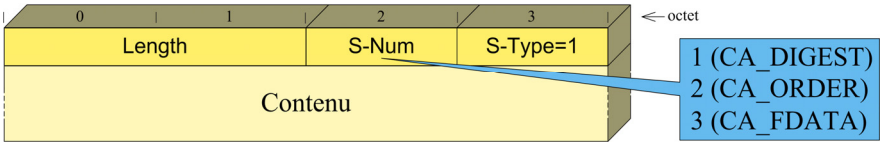


Figure 3-21 : Structure d'un objet contentAdaptation

Il s'agit en fait de la structure classique des objets COPS qui a déjà été décrite dans le chapitre 2 page 32.

3.2.4.2 Organisation des objets COPS-CA

L'organisation de ces trois objets COPS-CA (COPS contentAdaptation) dépend du type de message COPS DEC qui les transporte. L'Annexe A donne un exemple de message COPS complet dans chacun de ces cas.

3.2.4.2.1 Message install

Dans un message de décision d'installation, les objets COPS-CA sont regroupés par fichier de politique.

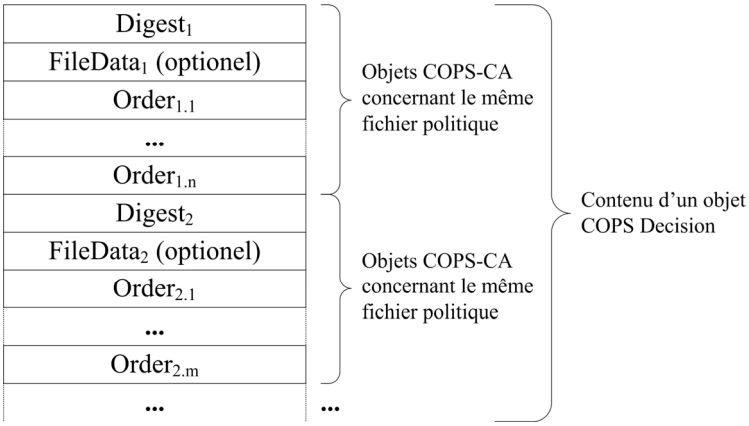


Figure 3-22 : Organisation des objets COPS-CA dans un message COPS DEC *install*

Dans un groupe d'objets COPS-CA, le premier objet est toujours la signature du fichier de politique utilisé par les objets Order qui suivent. La liste des objets Order peut contenir autant d'éléments que nécessaire. Si le PEP ne connaît pas encore ce fichier de politique, il est possible de préciser son contenu avec un objet FileData placé juste après l'objet Digest.

Si l'objet FileData est absent, l'objet Digest permet au PEP de s'assurer que le fichier politique qu'il connaît est identique à celui qui se trouve sur le PDP. Dans le cas contraire, il sert au PEP à éviter le calcul de la signature.

3.2.4.2.2 Message remove

Dans un message de décision de suppression, il n'y a que des objets Order. Ceux-ci ne sont pas organisés. Ils doivent tous être désinstallés.

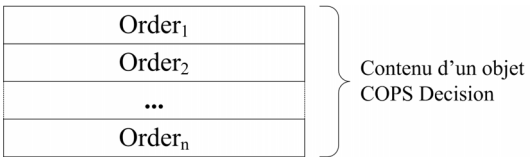


Figure 3-23 : Organisation des objets COPS-CA dans un message COPS DEC *remove*

3.2.4.2.3 Message modify

Dans un message de décision de modification, les objets COPS-CA sont organisés en groupe d'objets concernant le même fichier de politique. Ces groupes commencent toujours par un objet Digest. Ensuite, les objets Order viennent par paire pour indiquer quel ordre élémentaire modifier et comment le modifier.

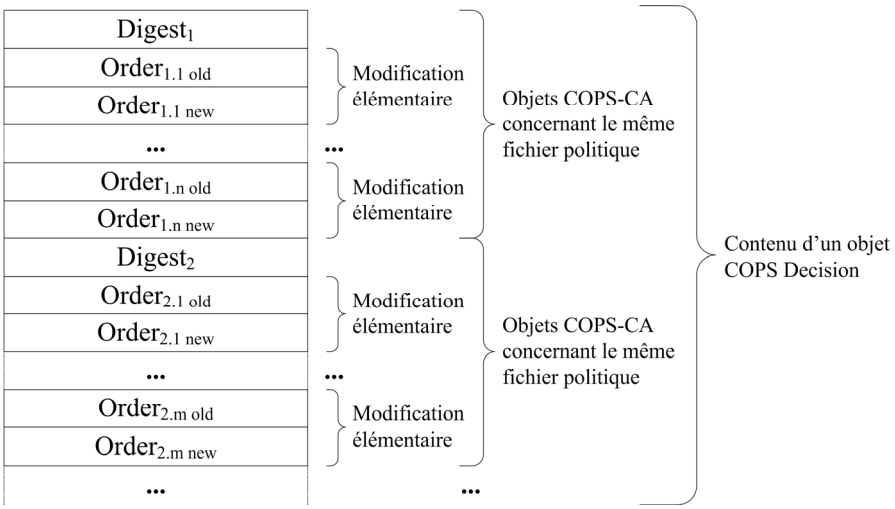


Figure 3-24 : Organisation des objets COPS-CA dans un message COPS DEC *modify*

Il n'y a pas d'objet FileData dans les messages de modification pour des raisons de stabilité. En effet, il n'est nécessaire d'envoyer le contenu du fichier de politique que si celui-ci a été modifié dans le PDP depuis son installation dans le PEP. Or il n'est pas certain que ces deux versions soient compatibles. Dans ce cas, il faut d'abord procéder à une suppression puis à une réinstallation de l'ordre élémentaire.

3.3 Démonstration

Le travail réalisé pendant mon stage dans le cadre du projet RHODOS pouvant servir au sein d'autres projets, il a été décidé de mettre en place une démonstration montrant les capacités du service contentAdaptation.

3.3.1 Plateforme

Cette démonstration s'est effectuée sur la plateforme du service TAI. Cette plateforme est utilisée à la fois pour le développement et les démonstrations.

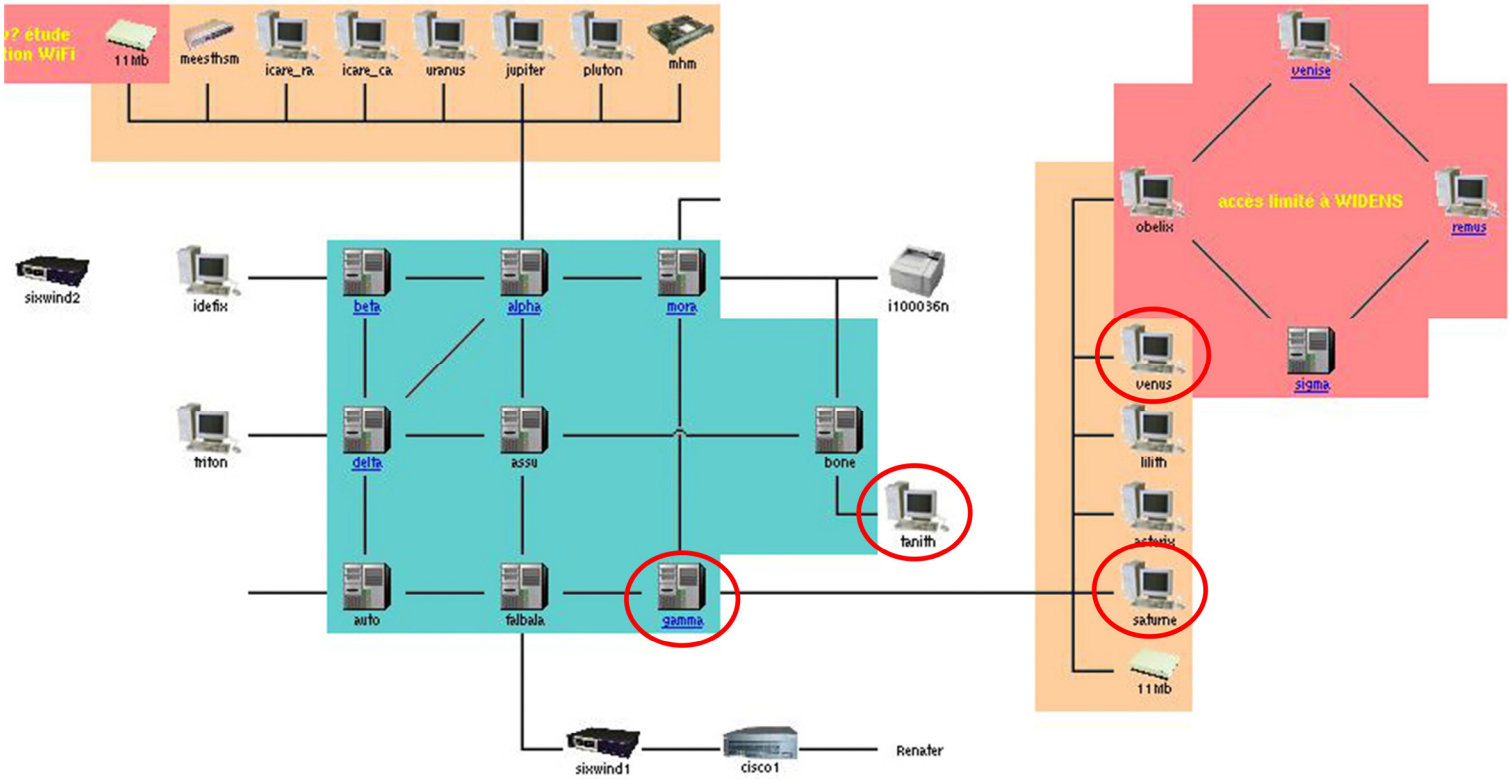


Figure 3-25 : Plateforme de développement et de démonstration du service TAI

♦ *Les machines*

Quatre machines sont utilisées pour la démonstration :

- venus (sous Linux) qui cumule les rôles de serveur vidéo, PDP et PEP.
- saturne (sous Linux) qui sert de PEP.
- gamma (sous Linux) qui sert de PEP.
- tanith (sous Windows) qui abrite tous les clients.

♦ *Le flux vidéo*

Le serveur de vidéo MJPEG 2000 à développer pour le projet RHODOS, ainsi que les clients et les modules de traitement de la vidéo, n'étant pas encore prêts, la démonstration utilise le client VLC du projet VidéoLAN à leur place.

♦ *La mobilité*

Les scénarios de la section 3.3.2 utilisent la mobilité des clients et considèrent les PEPs comme des antennes. La plateforme TAI étant entièrement filaire, cette mobilité est simulée par l'utilisation de plages de ports sur tanith.

Ainsi, les clients écoutant :

- sur les ports 1240 à 1249 se trouvent dans la zone couverte par venus
- sur les ports 1250 à 1259 se trouvent dans la zone couverte par gamma
- sur les ports 1260 à 1269 se trouvent dans la zone couverte par saturne

♦ *La plateforme finale*

Pour résumé, la démonstration se déroule sur la plateforme page suivante.

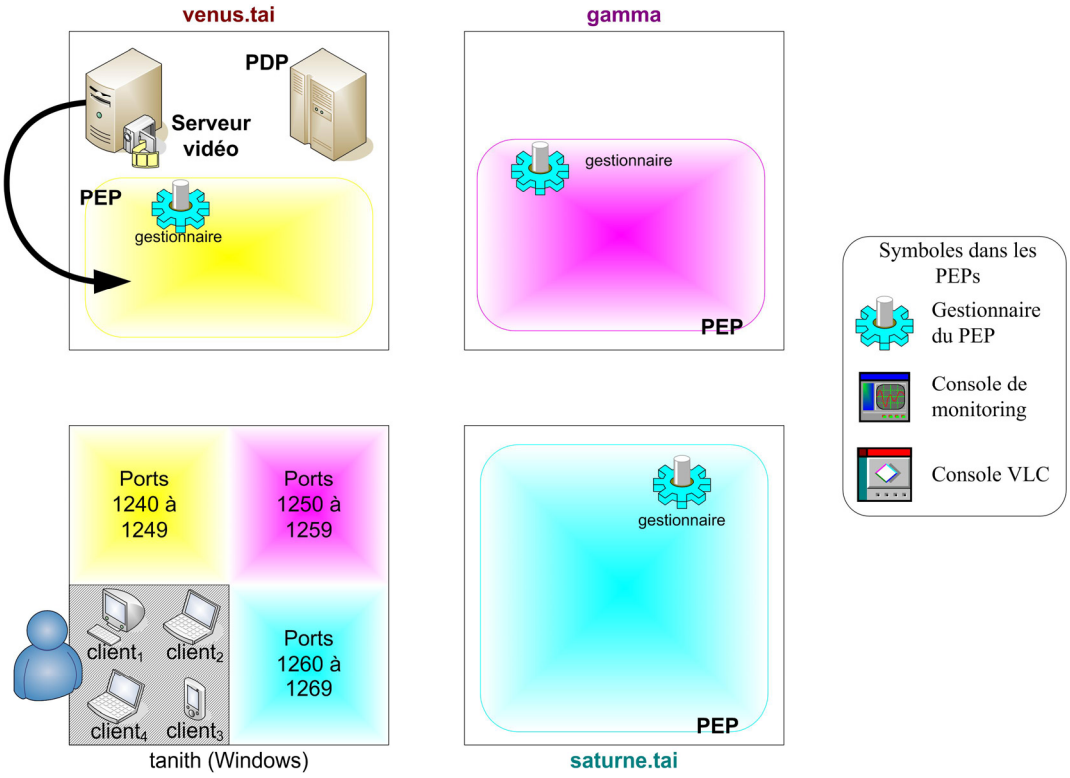


Figure 3-26 : Plateforme de démonstration

Cet ensemble de quatre machines représente le réseau virtuel suivant :

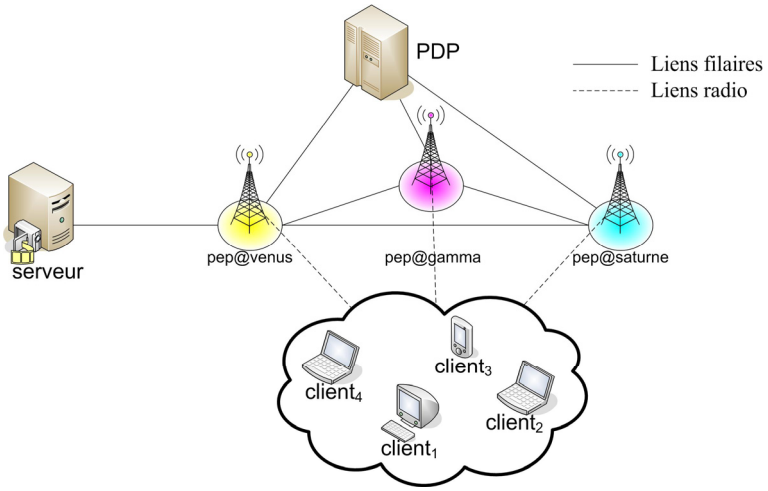


Figure 3-27 : Réseau virtuel de la démonstration

3.3.2 Scénarios

Afin d'illustrer les capacités du service contentAdaptation et son comportement face à des erreurs, trois scénarios ont été mis en place.

3.3.2.1 Fonctionnement nominal

Le premier scénario a pour objectif de mettre en œuvre les trois commandes du service. Il se décompose en quatre étapes :

- Installation d'une politique complexe.
- Première modification de la politique.
- Deuxième modification de la politique.
- Suppression de la politique.

3.3.2.1.1 Installation d'une politique complexe.

Dans ce scénario, trois clients souhaitent recevoir une vidéo sur des terminaux ayant des capacités d'affichage différentes :

- $Client_1$, qui se trouve dans la zone couverte par venus, peut afficher la vidéo dans sa taille d'origine.
- $Client_2$, qui se trouve à proximité de gamma, demande au service de réduire la taille de la vidéo par 2/3.
- $Client_3$ a le plus petit terminal. La taille de la vidéo doit être divisée par 2. Il est couvert par saturne.

Le réseau virtuel doit donc être configuré comme indiqué ci-dessous.

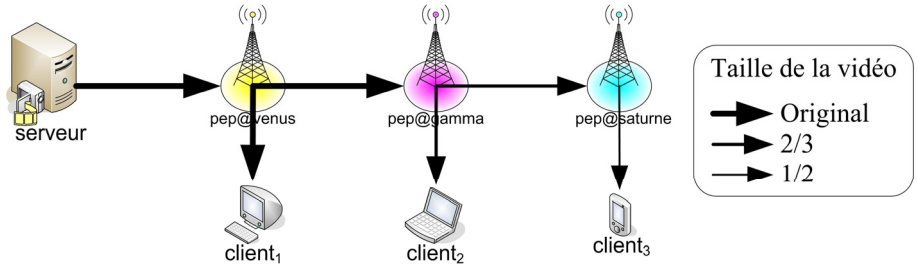


Figure 3-28 : Scénario 1, étape 1 : Configuration du réseau virtuel

Pour les besoins de la démonstration, la configuration du réseau est réalisée avec une seule politique complexe. Cette politique installe également une console de *monitoring* du CPU dans chaque PEP, ainsi qu'une autre console de *monitoring* de la bande passante dans saturne. Ces consoles ne servent pas en réalité. Elles n'ont d'autre objectif que de complexifier un peu plus la politique.

Finalement la commande d'installation envoyée au PDP est la suivante :

```
31,
contentadaptation,
file_name=monitor.scr;type=CPU;venus=true;saturne=true;gamma=true|
file_name=monitor.scr;type=BP;saturne=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1240;dst_2=gamma:1234;venus=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1250;dst_2=saturne:1234;redim=0.66;gamma=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1260;redim=0.66;saturne=true,
venus,saturne,gamma
```

Cette commande a déjà été analysée page 77. Elle engendre l'échange des messages COSP illustré Figure 3-29.

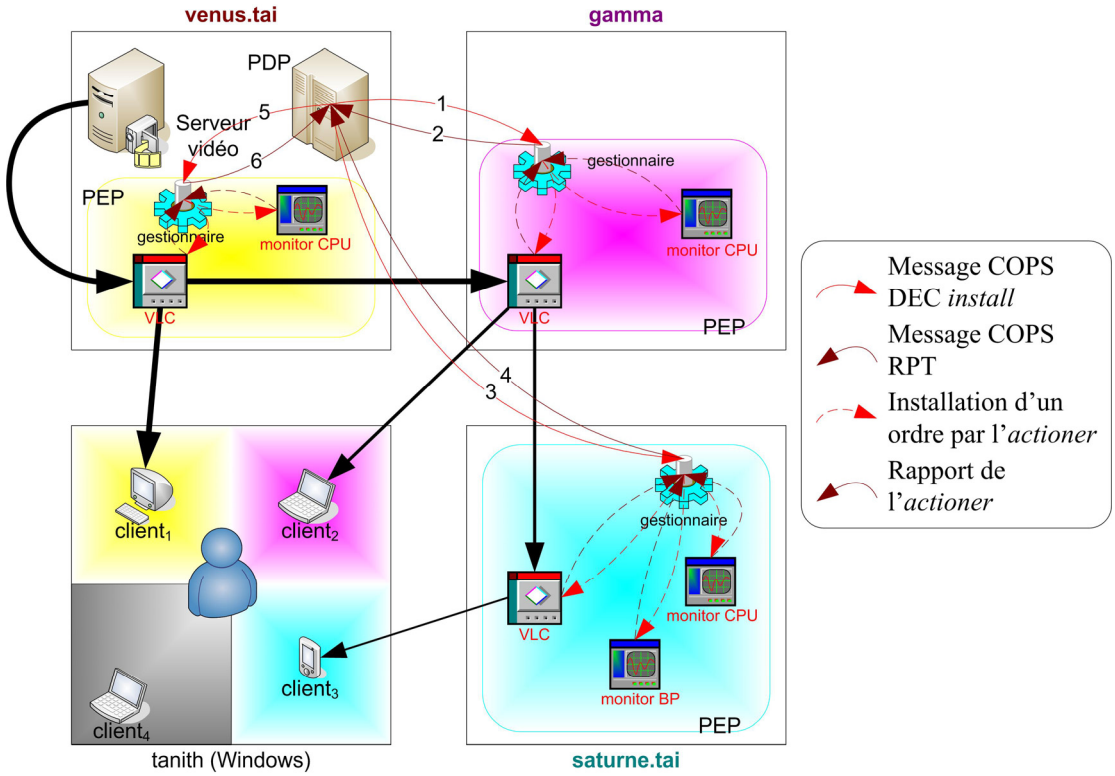


Figure 3-29 : Scénario 1, étape 1 : Messages COPS envoyés

Tous les clients peuvent ainsi visualiser la même vidéo avec des terminaux différents.

3.3.2.1.2 Première modification de la politique.

Dans cette étape, le client 2 se déplace dans la zone de couverture de saturne. Il demande toujours la même taille de vidéo. Il faut donc reconfigurer le réseau comme indiqué Figure 3-30.

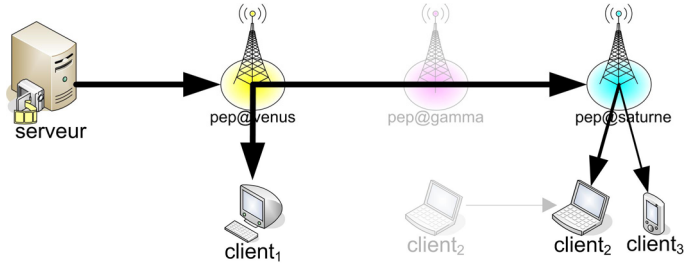


Figure 3-30 : Scénario 1, étape 2 : Configuration du réseau virtuel

Globalement, cette reconfiguration consiste à :

- Installer un nouveau module de filtrage (i.e. une console VLC) dans saturne
- Supprimer les ordres de gamma
- Rediriger le flux sortant de venus vers saturne

La commande de modification qui permet de réaliser cela est :

```
6,xxx,
file_name=vlc.scr;rcv_port=@:1235;dst_1=tanith:1261;dst_2=saturne:1234;redim=0.66;saturne=true%
file_name=monitor.scr;type=CPU;gamma=true;saturne=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1250;dst_2=saturne:1234;redim=0.66;gamma=true%
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1240;dst_2=gamma:1234;venus=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1240;dst_2=saturne:1235;venus=true,
venus,saturne,gamma
```

Voici les messages COPS envoyés pour la faire appliquer (les rapports ne sont pas indiqués) :

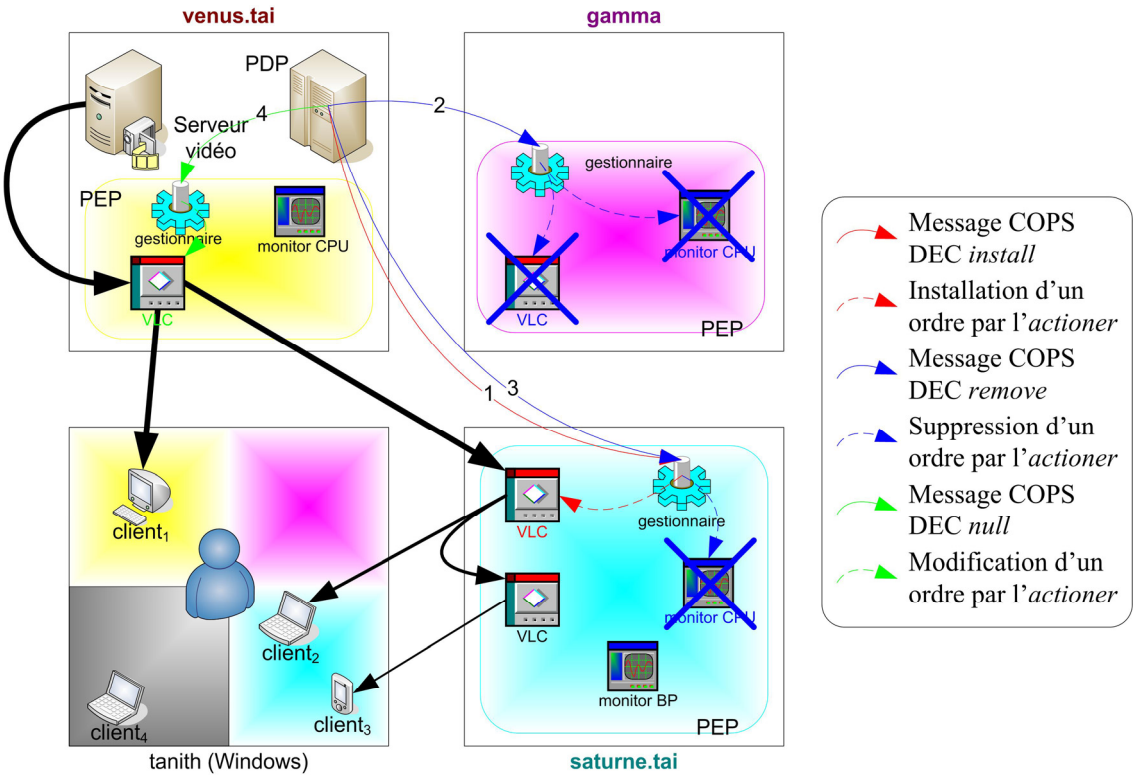


Figure 3-31 : Scénario 1, étape 2 : Messages COPS envoyés

3.3.2.1.3 Deuxième modification de la politique.

Le but de cette troisième étape est de prouver qu'il est possible de faire plusieurs modifications sur une même politique. On considère ici qu'un quatrième client arrive dans la zone de couverture de venus. Il demande de recevoir la vidéo redimensionnée avec un rapport 2/3.

Pour traiter cette demande, il faut :

- Installer un nouveau module d'adaptation sur venus.
- Rediriger l'ancien module d'adaptation de venus vers le nouveau.
- Modifier le module d'adaptation de saturne qui recevait le flux de venus pour qu'il ne redimensionne plus la taille de la vidéo.

La commande de modification pour cela est :

```
6,xxx,
file_name=vlc.scr;rcv_port=@:1235;dst_1=tanith:1241;dst_2=saturne:1235;redim=0.66;venus=true%
%
file_name=vlc.scr;rcv_port=@:1235;dst_1=tanith:1261;dst_2=saturne:1234;redim=0.66;saturne=true|
file_name=vlc.scr;rcv_port=@:1235;dst_1=tanith:1261;dst_2=saturne:1234;saturne=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1240;dst_2=saturne:1235;venus=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1240;dst_2=venus:1235;venus=true,
saturne,venus
```

Figure 3-32 représente les messages COPS envoyés pour appliquer cette commande.

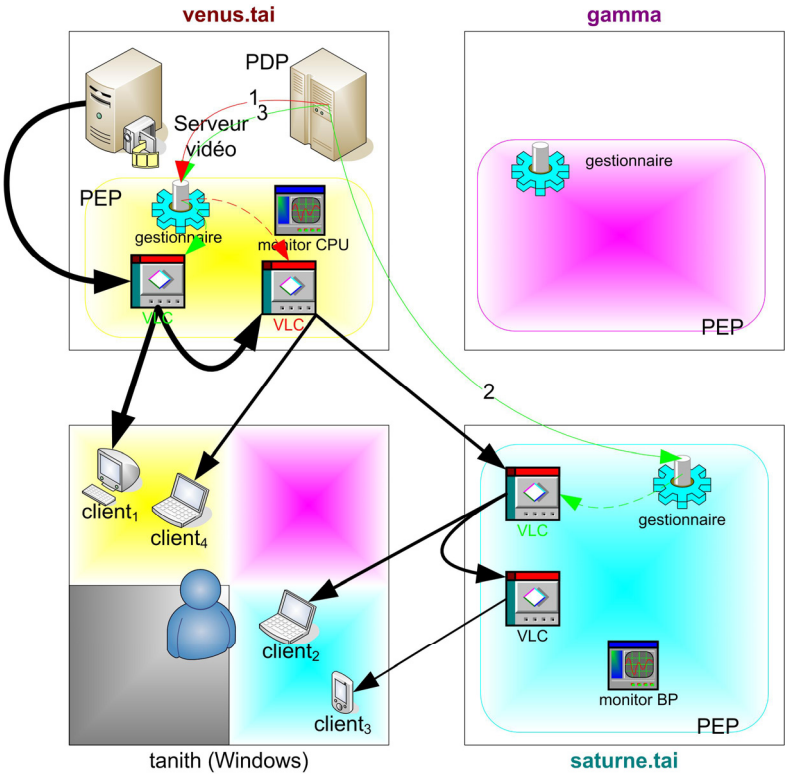


Figure 3-32 : Scénario 1, étape 3 : Messages COPS envoyés

3.3.2.1.4 Suppression de la politique.

Finalement les clients se retirent, par exemple parce que la vidéo est terminée. Il faut donc désinstaller la politique.

Cela est fait par la commande `4,xxx` qui déclenche les message COPS suivants :

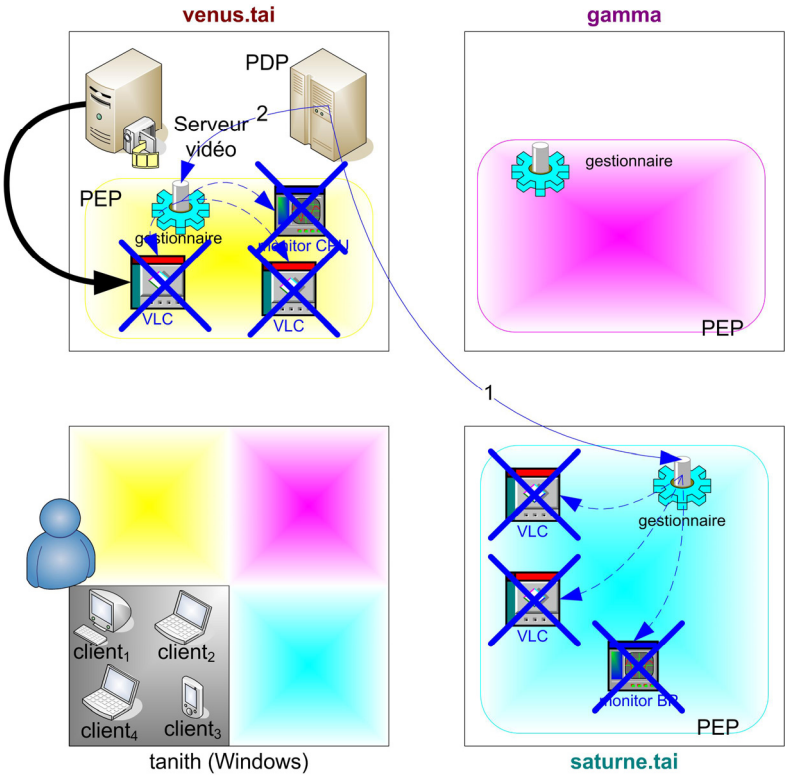


Figure 3-33 : Scénario 1, étape 4 : Messages COPS envoyés

Malgré plusieurs modifications, le PDP est resté dans un état parfaitement stable. Il a dons été capable de supprimer tous les ordres élémentaires encore installés.

3.3.2.2 Erreur pendant une modification

L'objectif de ce deuxième scénario est de voir comment le service contentAdaptation réagit face une erreur pendant l'application d'une modification.

Ce scénario est très fortement inspiré du premier. Trois clients 1, 2 et 3, ayant des terminaux de capacités différentes, se connectent respectivement à venus, gamma et saturne pour visualiser une vidéo. Il faut donc utiliser la même commande que dans le scénario précédent. Cependant, un nouvel ordre élémentaire a été rajouté afin de déclencher une erreur pendant la modification.

```
31,
contentadaptation,
file_name=monitor.scr;type=CPU;venus=true;saturne=true;gamma=true|
file_name=monitor.scr;type=BP;saturne=true|
file_name=x_chg_error.err_m;venus=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1240;dst_2=gamma:1234;venus=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1250;dst_2=saturne:1234;redim=0.66;gamma=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1260;redim=0.66;saturne=true,
venus,saturne,gamma
```

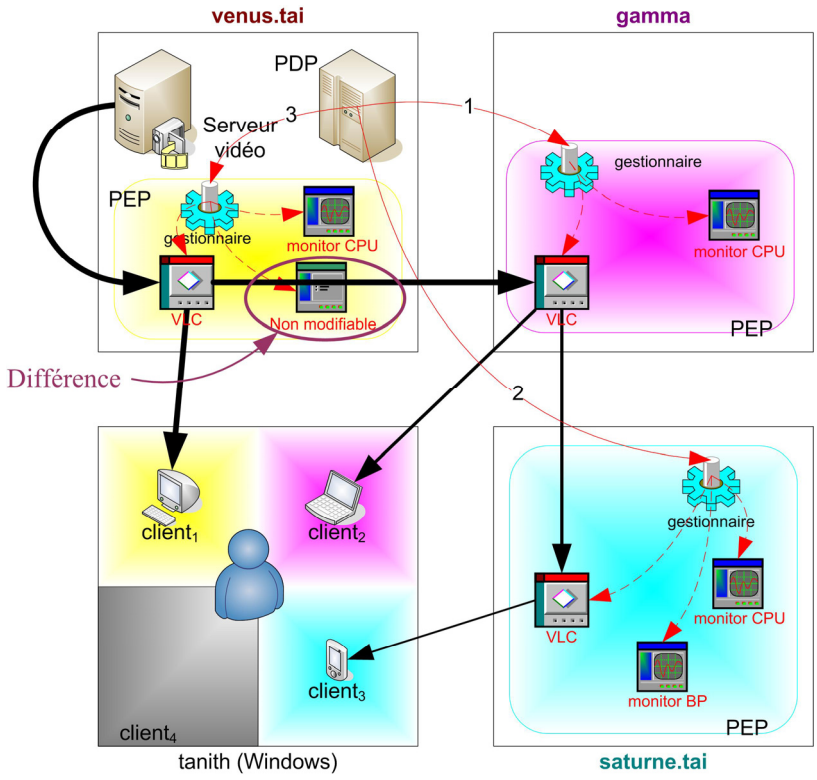


Figure 3-34 : Scénario 2, installation

Comme dans le premier scénario, le client 2 se déplace dans la zone couverte par saturne. Il faut donc reconfigurer le réseau avec la même commande. Cependant, pour déclencher l'erreur, nous allons modifier l'ordre élémentaire rajouté (qui n'est pas modifiable).

```
6,xxx,
file_name=vlc.scr;rcv_port=@:1235;dst_1=tanith:1261;dst_2=saturne:1234;redim=0.66;saturne=true%
file_name=monitor.scr;type=CPU;gamma=true;saturne=true|
```

```
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1250;dst_2=saturne:1234;redim=0.66;gamma=true%
file_name=x_chg_error.err_m;venus=true|
file_name=x_chg_error.err_m;p1=v1;venus=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1240;dst_2=gamma:1234;venus=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1240;dst_2=saturne:1235;venus=true,
venus,saturne,gamma
```

Tant que l'erreur ne s'est pas déclenchée, les messages COPS envoyés sont identiques à ceux de Figure 3-31 du scénario précédent.

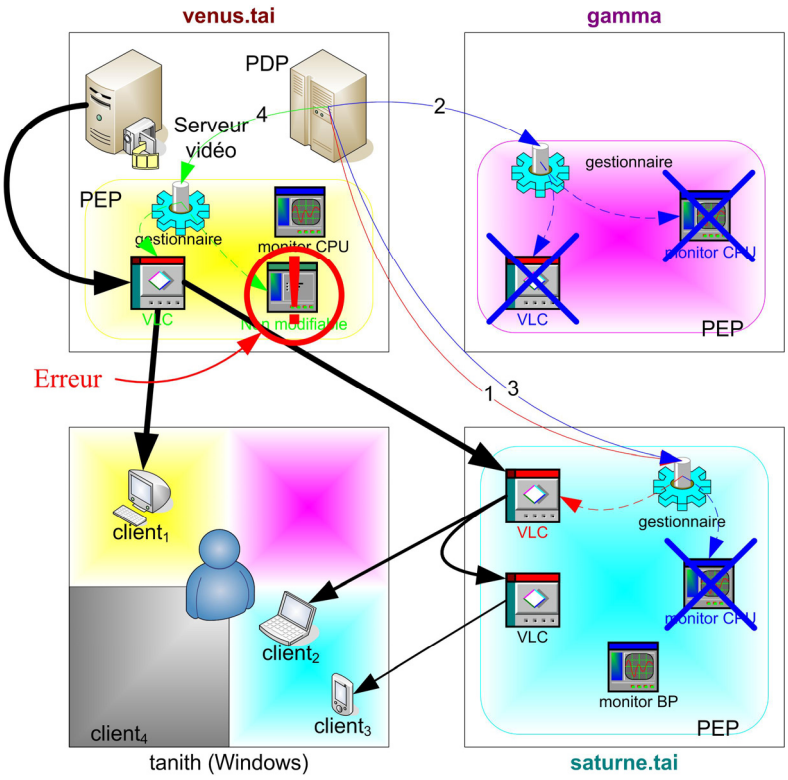


Figure 3-35 : Scénario 2, modification : fonctionnement nominal jusqu'à erreur

A la détection de l'erreur, le mécanisme de *rollback* annule toutes les modifications.

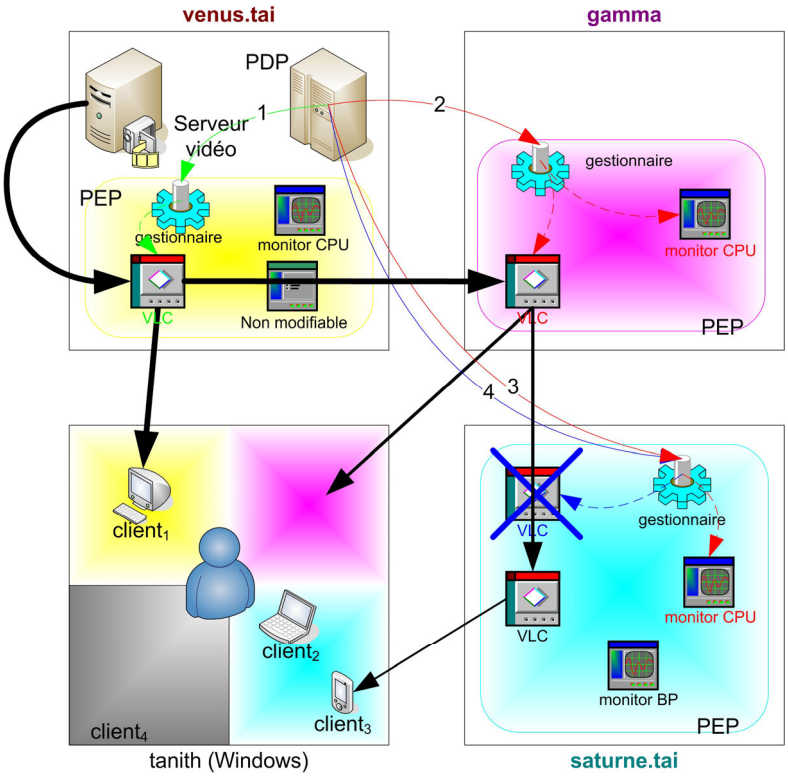


Figure 3-36 : Scénario 2, modification : *rollback*

Certes le client 2 ne reçoit plus le flux vidéo mais au moins le PDP est revenu à un état stable pour cette politique. Il est donc possible de réessayer la modification ultérieurement ou d'en tenter une autre.

3.3.2.3 Erreur pendant un rollback

Ce dernier scénario modifie très légèrement le scénario 2 afin d'introduire une erreur pendant le *rollback*.

La politique installée est quasi-identique au scénario précédent (voir Figure 3-37). Par contre la commande de modification installe sur venus un nouvel ordre élémentaire impossible à supprimer.

```
6,xxx,
file_name=x_del_error.err_d;venus=true|
file_name=vlc.scr;rcv_port=@:1235;dst_1=tanith:1261;dst_2=saturne:1234;redim=0.66;saturne=true%
file_name=monitor.scr;type=CPU;gamma=true;saturne=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1250;dst_2=saturne:1234;redim=0.66;gamma=true%
file_name=x_chg_error.err_m;venus=true|
file_name=x_chg_error.err_m;p1=v1;venus=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1240;dst_2=gamma:1234;venus=true|
file_name=vlc.scr;rcv_port=@:1234;dst_1=tanith:1240;dst_2=saturne:1235;venus=true,
venus,saturne,gamma
```

Les messages COPS engendrés par cette commande peuvent se diviser en trois parties :

- Application de la modification jusqu'à l'apparition d'une erreur (Figure 3-38).
- Annulation des modifications jusqu'à l'arrivée d'une erreur (Figure 3-39).
- Suppression complète de la politique (Figure 3-40).

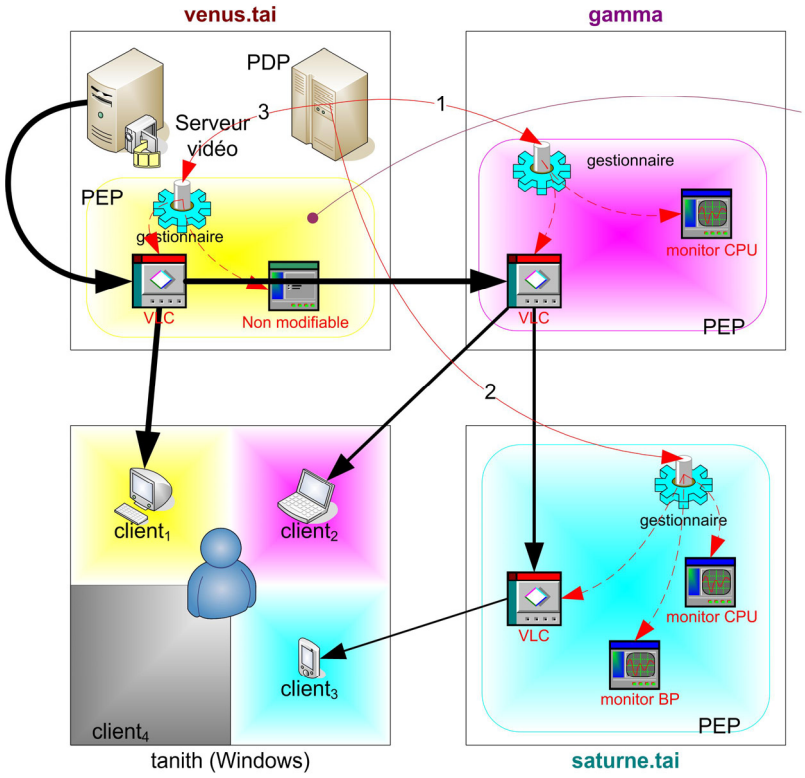


Figure 3-37 : Scénario 3, installation

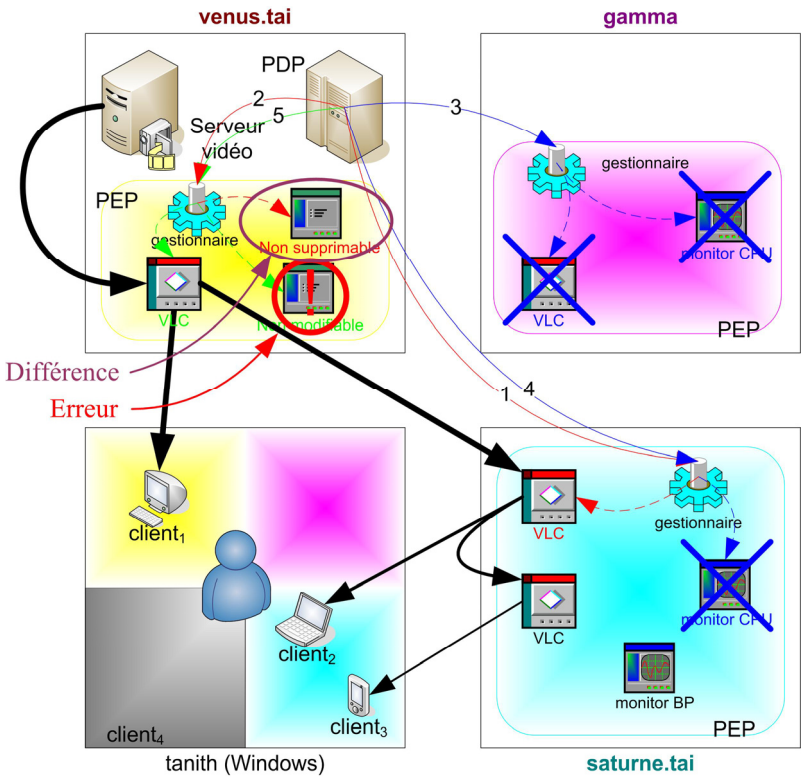


Figure 3-38 : Scénario 3, modification : fonctionnement nominal jusqu'à erreur

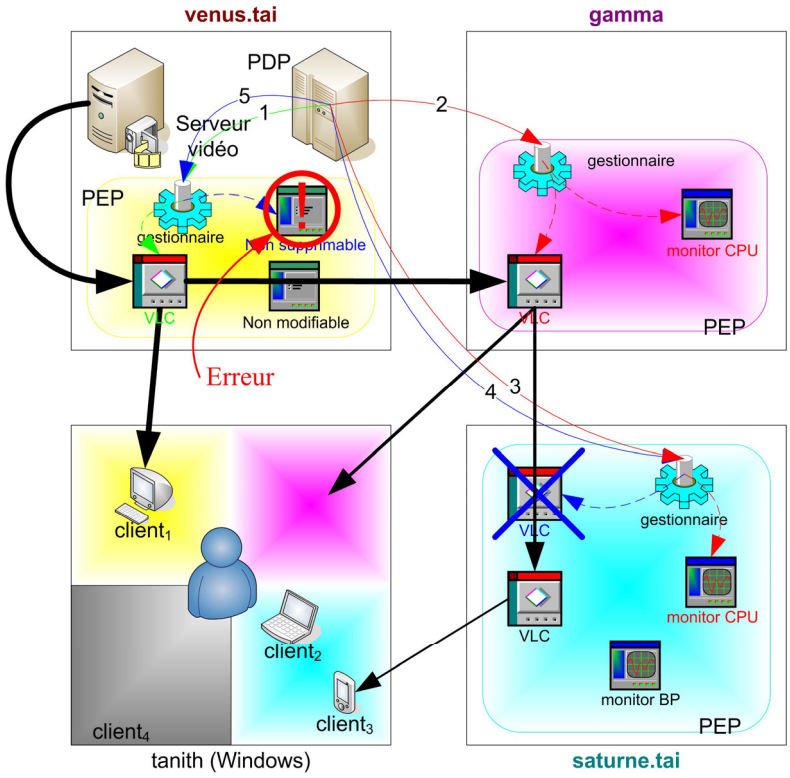


Figure 3-39 : Scénario 3, modification : *rollback* jusqu'à erreur

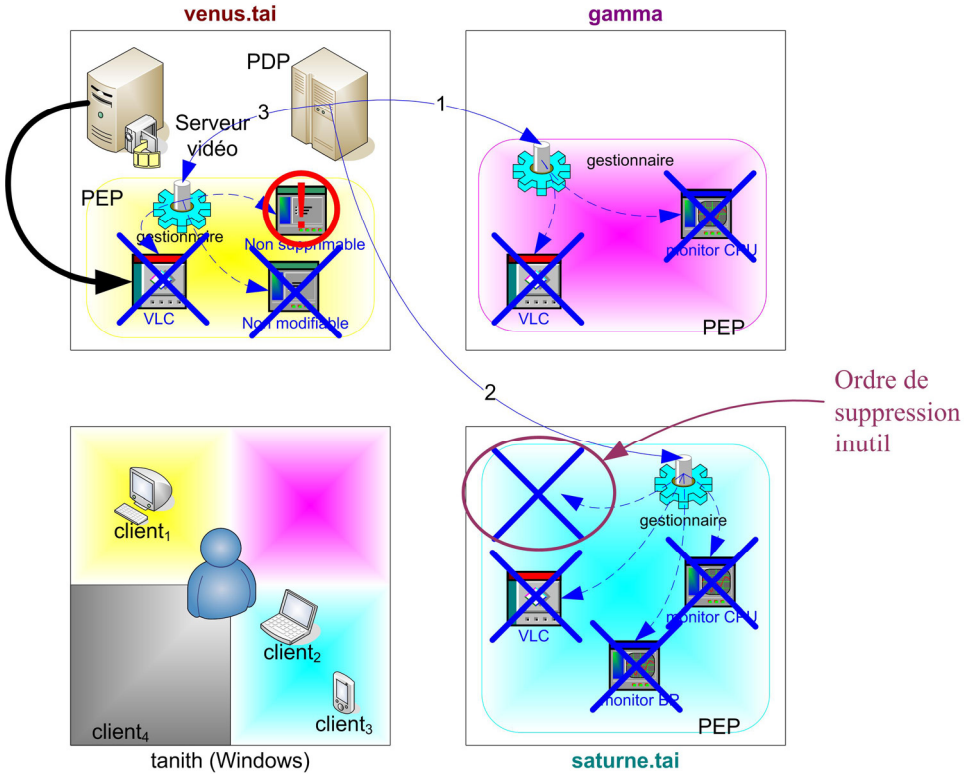


Figure 3-40 : Scénario 3, modification : suppression brutale de la politique

Figure 3-40 montre bien que la suppression due à une erreur pendant le *rollback* n'est pas identique à une simple suppression comme celle illustrée par Figure 3-33 page 95. En effet, dans ce cas, le PDP n'est plus dans un état stable. Il ne sait notamment plus quels ordres sont encore installés. Afin d'être sûr de désinstaller

complètement la politique, il envoie donc des ordres de suppression pour tous les ordres élémentaires potentiellement installés. Il est donc possible que certaines de ces suppressions soient inutiles.

3.3.3 Performances

Il est difficile d'effectuer de réelles mesures de performances sur ce service contentAdaptation. Cependant, la démonstration a permis de mettre en évidence plusieurs points.

Tout d'abord, l'utilisation de VLC pour traiter les flux vidéo n'est pas une solution envisageable. En effet, un décalage de quelques secondes a pu être constaté entre le flux entrant dans VLC et le flux sortant. Il faut préciser, à la décharge de VLC, que ce décalage était peut être en partie dû à la vidéo de mauvaise qualité utilisée. Cela souligne la nécessité d'une part, de développer des modules de traitement performants et d'autre part, d'utiliser le format Motion JPEG 2000 modifiable *frame* par *frame*.

La démonstration a également permis de voir les limites du traitement séquentiel des PEPs dans le cas de politiques complexes. Le fait d'attendre la réponse d'un PEP avant d'appliquer la politique à un autre PEP facilite grandement la fiabilisation du service. En contrepartie, cela peut engendrer, notamment dans le cas des commandes de modification, un délai de mise en place sensible. Ce délai devrait être acceptable dans le projet RHODOS, mais est clairement inadapté pour une utilisation à plus grande échelle.

CONCLUSION

Tout d'abord, pendant ce stage, j'ai pu acquérir de nouvelles connaissances dans le domaine des réseaux qui me serviront très probablement plus tard. Mes recherches bibliographiques m'ont, en effet, fait découvrir la gestion par politiques et le protocole COPS en détail, ainsi que d'autres sujets annexes plus superficiellement.

Mon travail bibliographique m'a également été bénéfique, de façon plus générale, en m'apprenant d'une part à lire des documents de recherche et des normes, notamment des RFCs et d'autre part à rédiger un état de l'art.

Ce stage m'a aussi fait découvrir ce qu'est la recherche dans un laboratoire industriel d'un grand groupe comme Thales. Le rôle qu'un tel service joue dans l'entreprise et comment il est organisé.

Ce stage m'a, par ailleurs, permis de participer à un projet RNRT. J'ai ainsi pu voir l'avantage d'une telle structure : le regroupement de compétences diverses autour d'un sujet commun ; et ses défauts potentiels : difficulté de communication entre les partenaires et avancement des travaux à des vitesses différentes.

Finalement, j'ai énormément apprécié ce stage au sein du service TAI qui m'a tant apporté d'un point de vue culturel que personnel. Si bien que je souhaiterais poursuivre mes études par une thèse également en partenariat avec un industriel.

REFERENCES

1. Généralités sur le PBNM

1.1. Présentations généralistes

[1] **Understanding Policy-Based Networking**
D. Kosiur
Wiley (Edition)
2001

[2] **A Distributed Policy-based Network Management (PBNM) system for Enriched Experience Networks™ (EENs)**
Sheridan-Smith Nigel
http://www.eng.uts.edu.au/~nigelss/NSS_Final_DoctoralAssessment-v1.00.pdf
Novembre 2003

[3] **A Policy Framework for Integrated and Differentiated Services in the Internet**
R. Rajan, D. Verma, S. Kamat, E. Felstaine, S. Herzog
IEEE Network, Septembre/Octobre 1999, pp. 36-41

[4] **Policy-Based Networks**
JC. Martin
Sun BluePrints Online
http://www.sun.com/blueprints
Octobre 1999

[5] **Qualité de service sur IP**
JL. Mélin
Eyrolles
2001

1.2. Définitions plus formelles

[6] **A Framework for Policy-based Admission Control**
RFC 2753
R. Yavatkar, D. Pendarakis, R. Guerin
Janvier 2000

[7] **Terminology for Policy-Based Management**
RFC 3198
A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog,A. Huynh, M. Carlson, J. Perry, S. Waldbusser
Novembre 2001

2. COPS

[8] **The COPS (Common Open Policy Service) Protocol**
RFC 2748
D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, A. Sastry
Janvier 2000
[9] **HMAC: Keyed-Hashing for Message Authentification**
RFC 2104
H. Krawczyk, M. Bellare, R. Canetti
Février 1997
[10] **The MD5 Message-Digest Algorithm**
RFC 1321
R. Rivest
Avril 1992
[11] **Service Location Protocol , Version 2**
RFC 2608
E. Guttman, C. Perkins, J. Veizades, M. Day
Juin 1999

3. COPS-RSVP

3.1. Les bases

[12] **Resource ReSerVation Protocol (RSVP) - Functional Specification**
RFC 2205
R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin
Septembre 1997
[13] **RSVP Extensions for Policy Control**
RFC 2750
S. Herzog
Janvier 2000
[14] **COPS usage for RSVP**
RFC 2749
S . Herzog, J. Boyle, R. Cohen, D. Durham, R. Rajan, A. Sastry
Janvier 2000

3.2. Pour aller plus loin

[15] **Signaled Preemption Priority Policy Element**
RFC 3181
S. Herzog
Octobre 2001
[16] **Identity Representation for RSVP**
RFC 3182
S. Yadav, R. Yavatkar, R. Pabbati, P. Ford, T. Moore, S. Herzog, R. Hess
Octobre 2001

4. COPS-PR

4.1. Les bases

[17] **COPS Usage for Policy Provisioning (COPS-PR)**
RFC 3084
K. Chan, J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer,R. Yavatkar, A. Smith
Mars 2001

[18] **Structure of Policy Provisioning Information (SPPI)**
RFC 3159
K. McCloghrie, M. Fine, J. Seligson, K. Chan, S. Hahn, R. Sahita, A. Smith,F. Reichmeyer
Août 2001
[19] **Framework Policy Information Base**
RFC 3318
R. Sahita, S. Hahn, , K. ChanK. McCloghrie
Mars 2003

4.2. Pour aller plus loin

[20] **RSVP Policy Control Criteria PIB**
draft-ietf-rap-rsvppcc-pib-01.txt
Diana Rawlins, Lei Yao, Richard McClain, Amol Kulkarni
Mars 2002
[21] **Differentiated Services Quality of Service Policy Information Base**
RFC 3317
K. Chan, R. Sahita, S. Hahn, K. McCloghrie
Mars 2003
[22] **Framework Policy Information Base for Usage Feedback**
RFC 3571
D. Rawlins, K. Chan, , A. Kulkarni, M. Bokaemper, D. Dutt
Août 2003
[23] **Framework for Policy Usage Feedback for Common Open Policy Service with Policy Provisioning (COPS-PR)**
RFC 3483
D. Rawlins, A. Kulkarni, M. Bokaemper, K. Chan
Mars 2003

5. Netconf

[24] **NETCONF Configuration Protocol**
draft-ietf-netconf-prot-02
14 février 2004
R. Enns
[25] **BEEP Application Protocol Mapping for NETCONF**
draft-ietf-netconf-beep-00
E. Lear, K. Crozier
7 octobre 2003
[26] **Using the NETCONF Configuration Protocol over Secure Shell (SSH)**
draft-ietf-netconf-ssh-00
M. Wasserman, T. Goddard
19 octobre 2003
[27] **NETCONF Over SOAP**
draft-ietf-netconf-soap-01
T. Goddard
13 février 2004

6. Représentation et stockage des politiques

6.1. Représentation

[28] **Policy Core Information Model - Version 1 Specification**
RFC 3060
B. Moore, E. Ellesson, J. Strassner, A. Westerinen
Février 2001

[29] **Policy Core Information Model (PCIM) Extensions**
RFC 3460
B. Moore
Janvier 2003
[30] **Common Information Model (CIM) Standards**
http://www.dmtf.org/standards/cim/
[31] **Policy Quality of Service (QoS) Information Model**
RFC 3644
Y. Snir, Y. Ramberg, J. Strassner, R. Cohen, B. Moore
Novembre 2003
[32] **Ponder : A language for Specifying Security and Management Policies for Distribued Systems.**
Imperial College of Science, Technology and Medecine
N. Damianou, N. Dulay, E. Lupu, M.Sloman
2000
[33] **A rule language for network policies**
C and C Network Product Development Laboratories
J. Nicklisch

6.2. Stockage

[34] **LDAP**
http://www-sop.inria.fr/semir/personnel/Laurent.Mirtain/ldap-livre.html
Laurent Mirtain – INRIA
Octobre 1999
[35] **Lightweight Directory Access Protocol (v3)**
RFC 2251
M. Wahl, T. Howes, S. Kille
Décembre 1997
[36] **Policy Core Lightweight Directory Access Protocol (LDAP) Schema**
RFC 3703
J. Strassner, B. Moore, R. Moats, E. Ellesson
Février 2004

GLOSSAIRE ET ACRONYMES

A

Acronymes

AD	Administrative Domain
COPS	Common Open Policy Service
COPS-PR	COPS for Provisioning
COPS-RSVP	COPS for RSVP
CoS	Class of Service
DES	Directory Service Entry
DIT	Directory Information Tree
DMTF	Distributed Management Task Force
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
LDAP	Lightweight Directory Access Protocol
LPDP	Local Policy Decision Point
PBN	Policy-Based Network
PBNM	Policy-Based Network Management
PCIM	Policy Core Information Model
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIB	Policy Information Base
PIN	Policy Ignorant Node
QoS	Quality of Service
QPIM	QoS Policy Information Model
RAP	Ressource Allocation Protocol
RFC	Request For Comment
SNMP	Simple Network Management Protocol
SPPI	Structure of Policy Provisioning Information
SQL	Structured Query Langage
TLS	Transaction Layer Security

C

Client (COPS) 33
Client (Implémentation) 66
Client-Type (COPS) 33
Contexte (Implémentation) 72

E

Etat (COPS) 31, 33, 40

F

Fichier de politique (Implémentation) 75

P

Policy Repository 24, 50

S

Session (COPS) 35, 37

Annexe A

EXEMPLES DE MESSAGES COPS

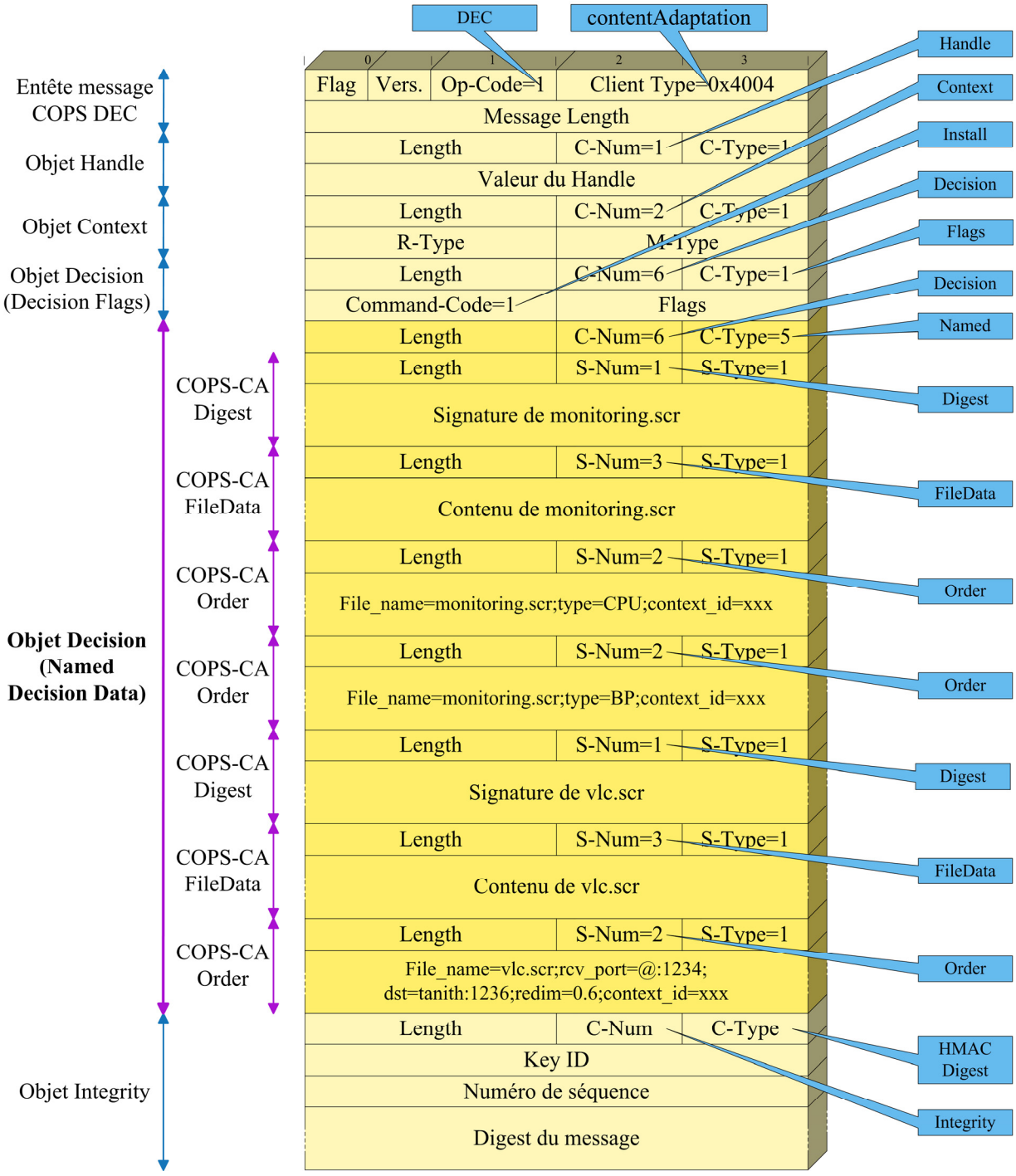
UTILISANT DES OBJETS COPS-CA

Cette annexe illustre l'utilisation des objets COPS-CA présentés page 87 dans des message COPS DEC de type *install*, *modify* (*null*) et *remove*.

♦ *Message COPS DEC install*

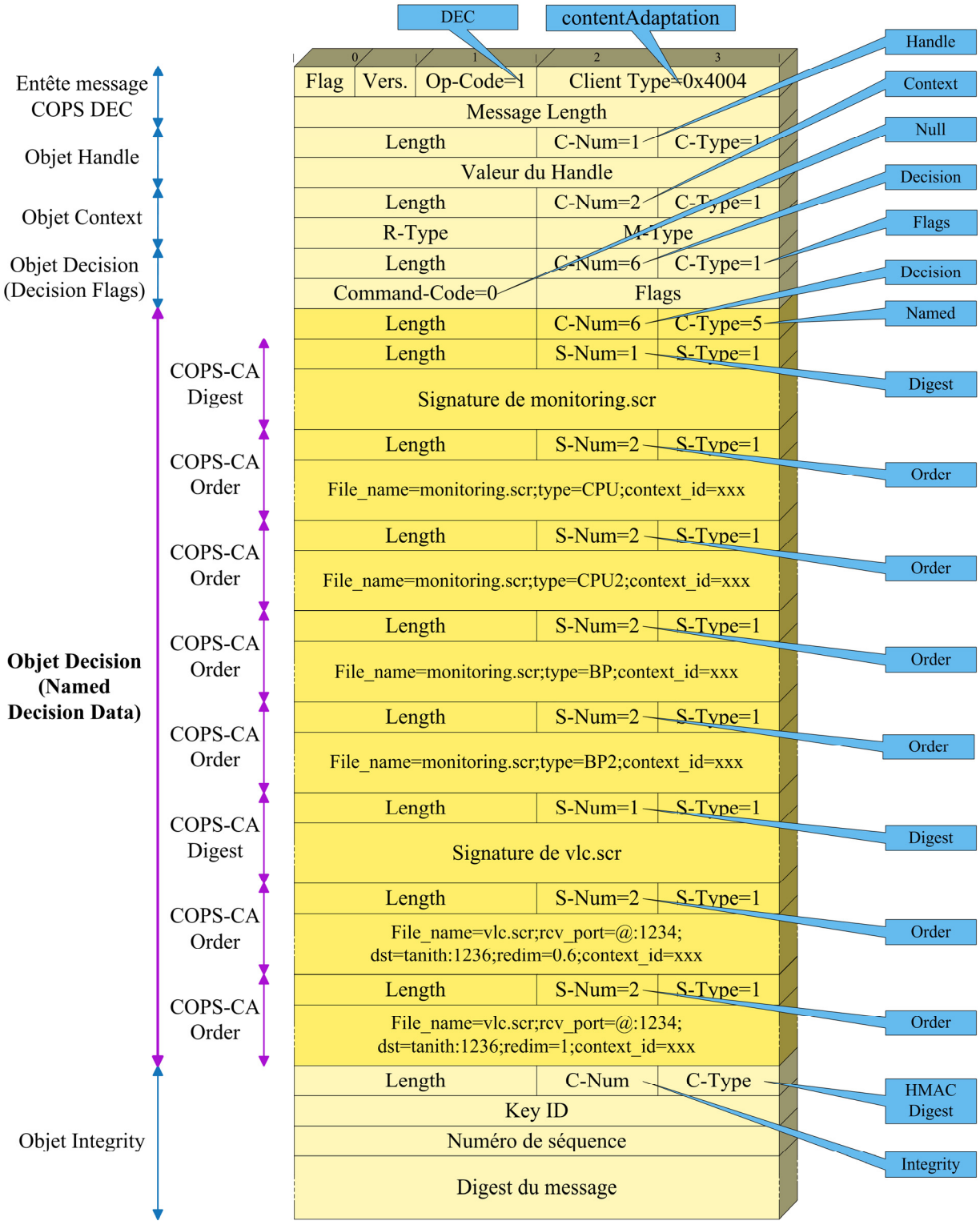
Le message COPS DEC de cet exemple demande au PEP destinataire d'installer les trois ordres élémentaires suivants :

- `file_name=monitoring.scr;type=CPU;context_id=xxx`
- `file_name=monitoring.scr;type=BP;context_id=xxx`
- `file_name=vlc.scr;rcv_port=@:1234;dst=tanith:1236;redim=0.6;context_id=xxx`



♦ *Message COPS DEC modify*

Ce message COPS DEC *null* (interprété par le client contentAdaptation comme un message *modify*) modifie les ordres élémentaires précédemment installés.



Ce message effectue donc les modifications suivantes:

- `file_name=monitoring.scr;type=CPU;context_id=xxx`

  → `file_name=monitoring.scr;type=CPU2;context_id=xxx`

- `file_name=monitoring.scr;type=BP;context_id=xxx`

  → `file_name=monitoring.scr;type=BP2;context_id=xxx`

- `file_name=vlc.scr;rcv_port=@:1234;dst=tanith:1236;redim=0.6;context_id=xxx`

  → `file_name=vlc.scr;rcv_port=@:1234;dst=tanith:1236;redim=1;context_id=xxx`

♦ *Message COPS DEC remove*

Pour terminer, ce message supprime tous les ordres élémentaires modifiés.