Gestion de réseaux par politiques

Avril 2004

Tuteurs Arnaud PIERRE
Yulen SADOURNY

THALES
Land & Joint Systems

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 Arnaud PIERRE pour sa disponibilité, son écoute, ses conseils et pour m'avoir guidé dans la réalisation de ce document afin qu'il me soit le plus profitable possible pour mon travail à venir.

Je tiens à remercier tout particulièrement Hervé A. pour le temps qu'il a consacré à m'expliquer tous les points difficiles de la gestion des réseaux par politique. Sans son aide précieuse, cet état de l'art aurait été beaucoup moins complet.

Je remercie enfin Frédérique S.A. pour sa relecture.

TABLE DES MATIERES

TABLE DES FIGURES

PROLOGUE

Ce document a été fait principalement à partir de RFC (Request For Comment) de l'IETF (Internet Engineering Task Force) et de quelques livres plus généralistes comme en témoignent les références. Cet état de l'art décrit donc des technologies normalisées ou en voie avancée de normalisation. Il n'a pas pour objectif de présenter toutes les recherches actuelles sur le sujet.

Bien que réalisé en français, ce document utilise au maximum les désignations anglophones pour tous les termes techniques. Cela peut engendrer quelques problèmes de français mais présente deux avantages :

- Séparation sans ambiguïté entre les termes techniques et les mêmes termes utilisés dans un contexte non technique. Pour renforcer cela, puisque certains mots ont la même orthographe dans les deux langues, les termes techniques commencent par une majuscule.
- La terminologie anglophone est beaucoup plus utilisée que son homologue française (quand elle existe).

INTRODUCTION

Aujourd'hui les utilisateurs des réseaux sont de plus en plus exigeants. Ils ne veulent plus se contenter d'un simple service Best-Effort. Ils veulent de la qualité de service, de la sécurité, de la disponibilité etc… Des solutions sont apparues assez récemment pour répondre à ces problèmes. Cependant celles-ci sont toujours assez complexes à administrer. Il est dangereux, voire impossible pour de grands réseaux, de configurer manuellement l'ensemble des équipements du réseau à cause de l'abondance des informations dont il faut tenir compte et surtout de leur nature dynamique.

Le recours à des outils de gestion autonomes et dynamiques se révèle donc indispensable. C'est dans cette optique que fut créée la gestion par politiques, couramment appelée PBNM (Policy-Based Network Management).

Bien qu'originellement destinée à être déployée dans des réseaux pour gérer la qualité de service, elle a été conçue pour être adaptable à d'autres besoins comme par exemple la sécurité ou encore l'adaptation de contenu (sujet du stage). Cependant, dans le cadre de cet état de l'art, nous nous placerons principalement dans le contexte de la QoS (Quality of Service) auquel font référence la majeure partie des normes disponibles à ce jour.

Ce document présentera dans le premier chapitre les notions de base pour le PBNM et mettra en avant ses avantages, ainsi que ses limitations, dans quelques exemples d'utilisation.

Le chapitre 2 donnera les architectures possibles d'un réseau géré par politiques en mettant en avant le rôle des deux principales entités : le PDP (Policy Decision Point) qui est l'organe décisionnel et le PEP (Policy Enforcement Point) qui est l'organe exécutif.

Le chapitre 3 décrira le protocole COPS permettant aux PEPs et aux PDPs de dialoguer ensemble. Du fait de l'importance de COPS, sa description sera assez détaillée afin de pouvoir bien comprendre ses avantages et ses limitations. Ce chapitre doit aussi permettre d'éclaircir certains points de la norme pouvant prêter à confusion.

Enfin, le chapitre 4 indiquera comment représenter et stocker les politiques définies par l'administrateur, c'est à dire à un haut niveau.

Ce document a été conçu pour être lu de façon linéaire, cependant les chapitres 3 et 4 peuvent être intervertis. De plus, il est possible de se passer de la deuxième partie du chapitre 3 sans que cela ne pénalise la compréhension globale du chapitre. Cette partie permet de préciser certains points ambigus mais n'est pas indispensable pour qui ne veut pas connaître les détails de COPS.

Les principaux éléments de 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.

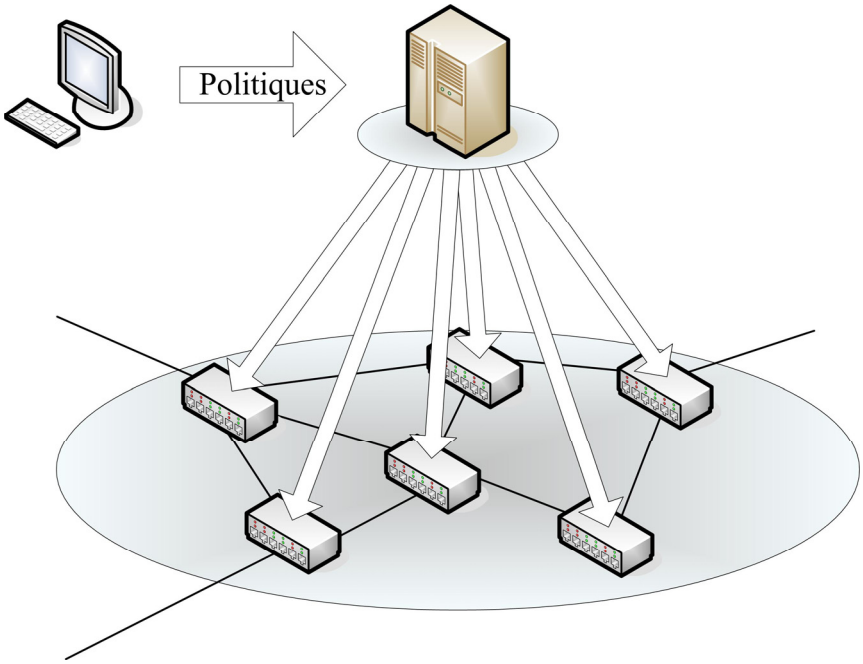


Figure 1 : Notion du PBNM

Figure 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 43.

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

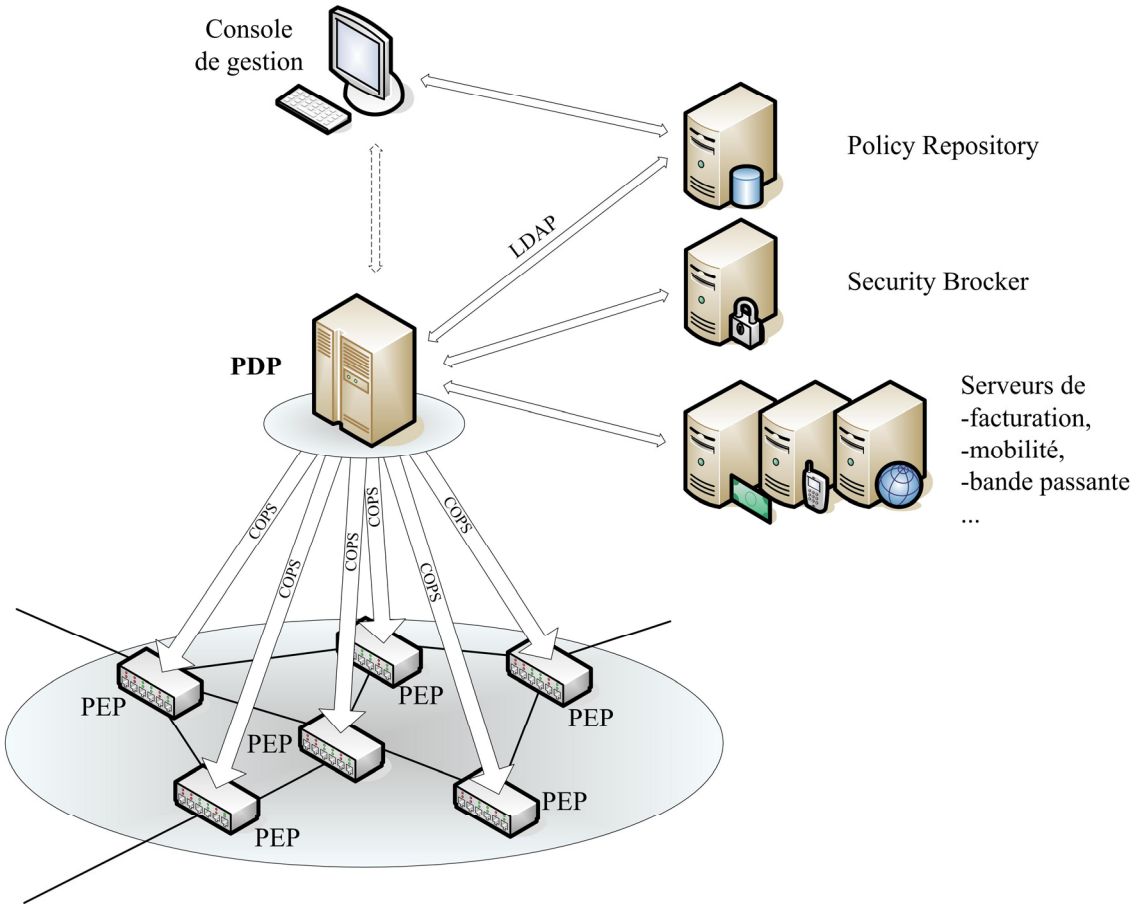


Figure 2 : Structure d'un PBN (modèle trois-tiers)

Figure 2 introduit le modèle dit « trois-tiers ». La section 2.2 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.

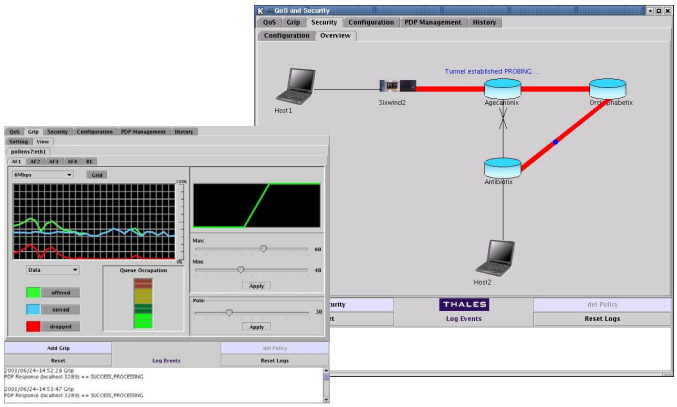


Figure 3 : Exemple de console de gestion avec interface graphique

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 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 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 (Mobility 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.

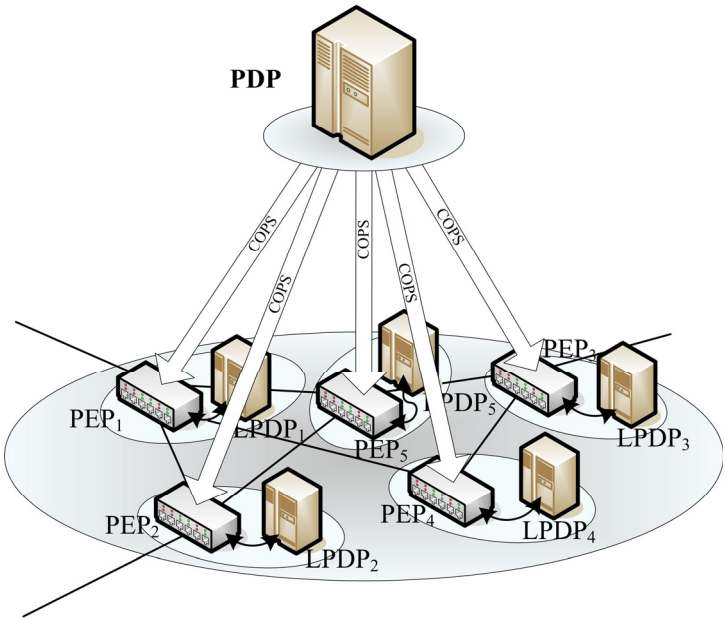


Figure 4 : PBN avec LPDP

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…).

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…

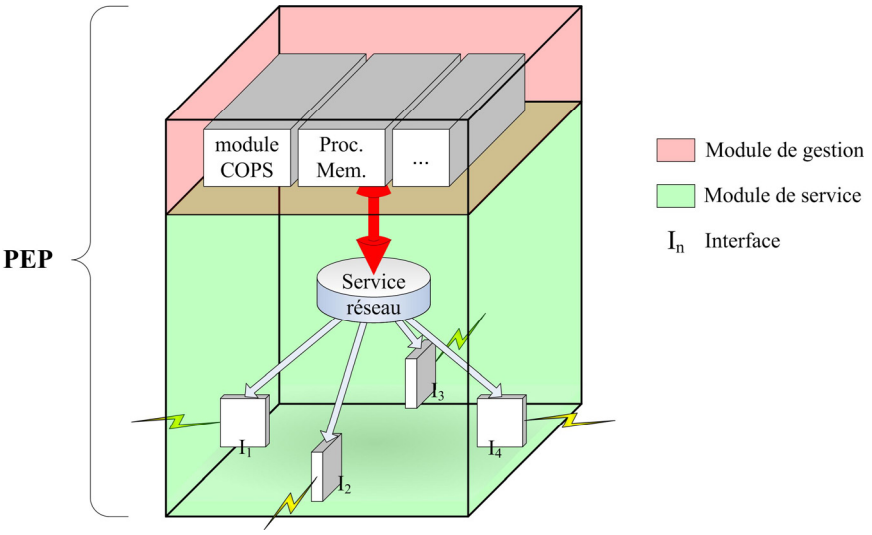


Figure 5 : Structure d'un PEP

Comme indiqué dans Figure 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 Modèle deux-tiers

Dans le modèle trois-tiers (Figure 2), les trois composantes principales sont constituées du PDP, du PEP et du répertoire de politiques. Le modèle deux-tiers en est une simplification comme le montre Figure 6.

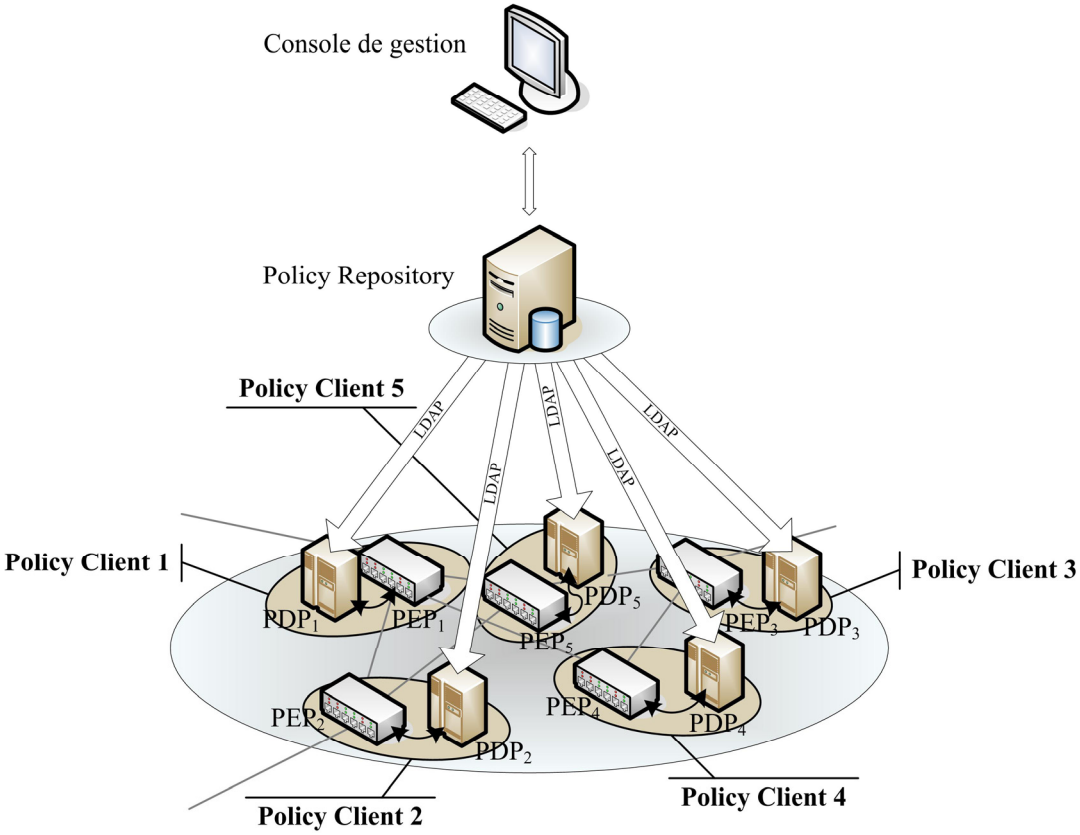


Figure 6 : Structure d'un PBN (modèle deux-tiers)

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.

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 à cause d'une 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 7 donne un exemple de ce à quoi peut ressembler un tel PBN.

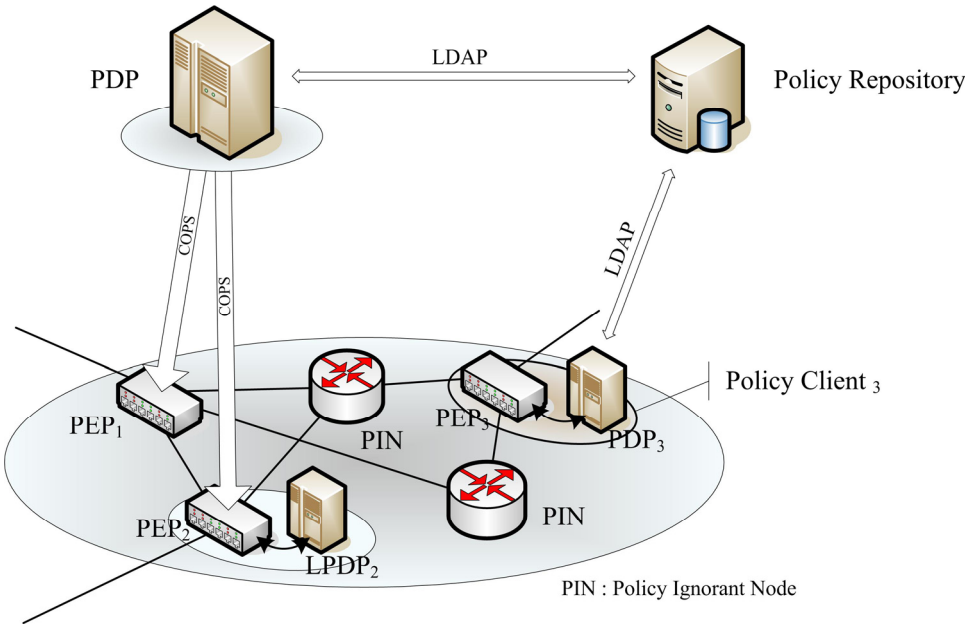


Figure 7 : Un exemple de PBN

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 11.

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).

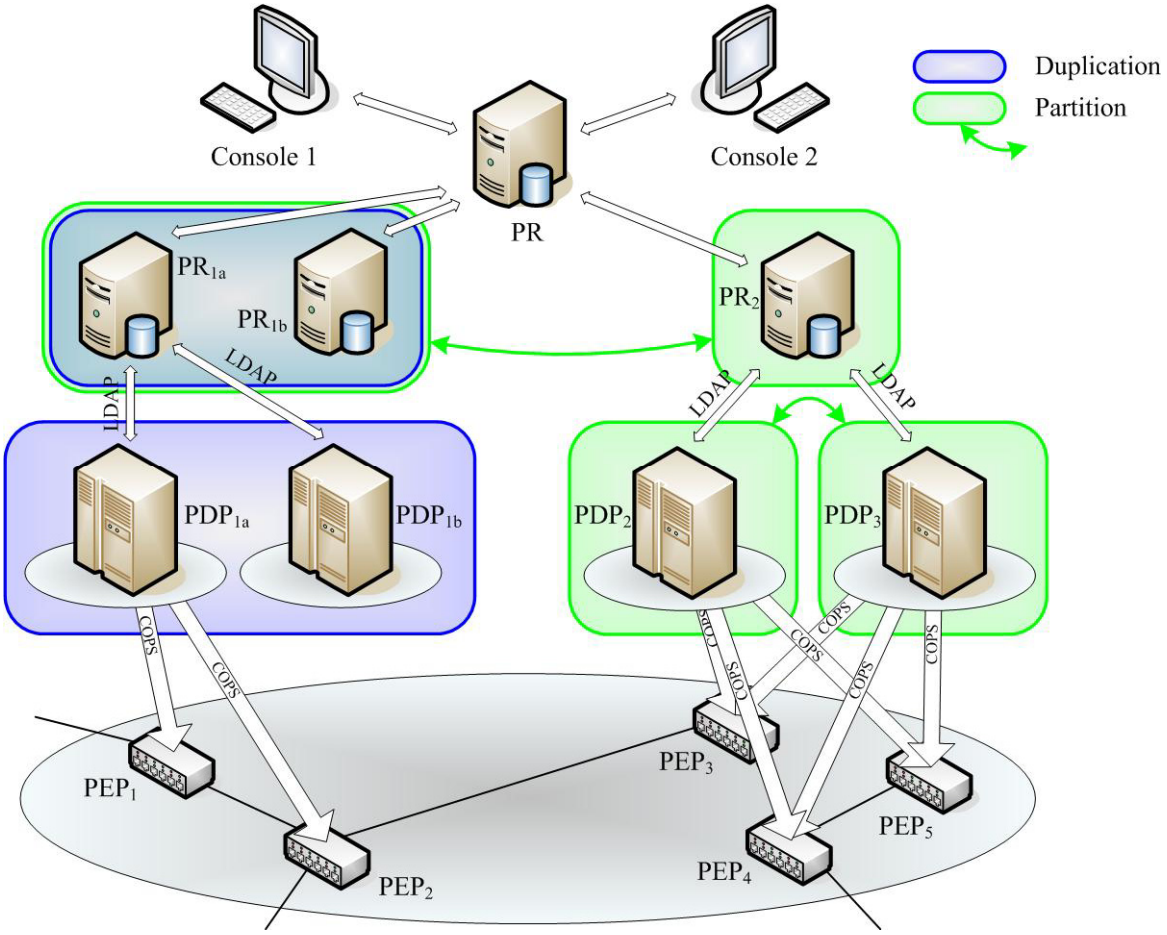


Figure Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-8 : Exemple d'un PBN évolué**

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

- PR_{1b} est un *policy repository* de secours au cas où PR_{1a} tomberait en panne.
- Les politiques de PR sont réparties entre PR_{1a} et PR_2 par exemple parce que PEP_1 et PEP_2 se trouvent à New York alors que PEP_3, PEP_4 et PEP_5 sont localisés à Paris.
- 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.

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 s'inscrivant le mieux dans l'architecture des réseaux gérer par politiques présentée précédemment.

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 envoie des requêtes, des mises à jour et des suppressions au PDP 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 : 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 par un ou plusieurs RFCs.

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, à 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.

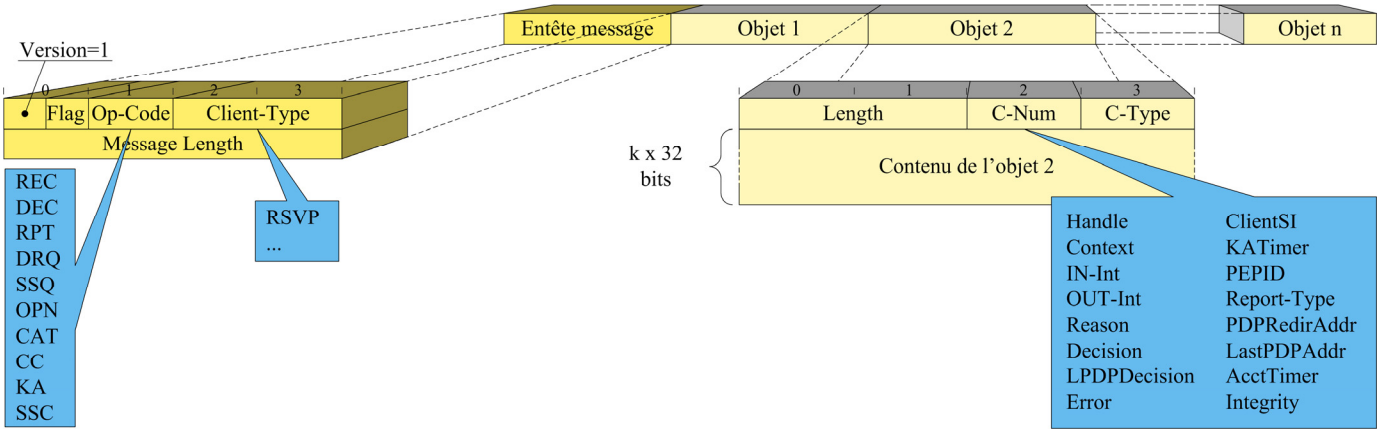


Figure 9 : Un message COPS

Tous les messages de COPS (et de ses extensions) ont la même structure définie dans Figure 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 9, Figure 10 donne l'entête commun à tous les objets de COPS.

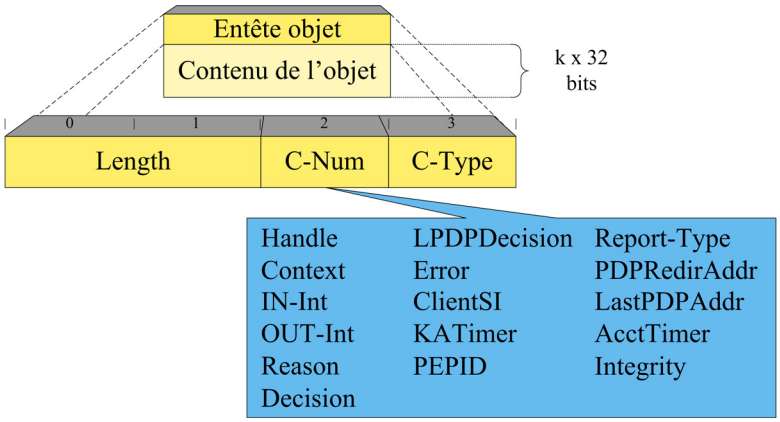


Figure 10 : Entête d'un objet COPS

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.

C-Type identifie soit le sous-type soit la version de l'information contenue dans l'objet. Il dépend donc de C-Num.

♦ *Objet Handle (Handle)*

Les Handles sont des éléments très importants dans COPS. En effet, ils permettent d'identifier de manière unique les états installés.

Concrètement un état correspond, dans le modèle de délégation (Outsourcing), à un type de flux RSVP et indique au PEP comment traiter tous les flux de ce type. Dans le modèle de l'approvisionnement (Provisioning), les états sont un ensemble de règles qui permettent au PEP de gérer un grand nombre de situations sans avoir recours au PDP. Dans le modèle de l'Outsourcing, il y a donc a priori un grand nombre d'états installés (un par type de flux) tandis que dans le modèle du Provisioning, il n'y en a que quelques-uns (voire un seul).

Il est donc très fréquent de trouver un objet Handle dans un message COPS puisque les messages COPS font très souvent référence à des états (voir Figure 12 page 32 pour s'en rendre compte).

Le format du Handle n'est pas imposé (la longueur de l'objet n'est donc pas connue à priori). Le PEP est responsable de sa création, de sorte qu'il soit unique, et de sa destruction, lorsqu'il n'est plus utilisé.

♦ *Objet Context (Context)*

Les objets Context servent à indiquer le contexte dans les messages de requête du PEP et de décision du PDP.

Le contexte indique le type d'événement qui a déclenché la requête. Aujourd'hui, c'est à dire dans [6], seuls quatre contextes sont définis :

- Arrivée d'un message
- Besoin d'allocation de ressources
- Sortie d'un message
- Besoin de configuration

Les trois premiers contextes sont utilisés dans le modèle de l'Outsourcing et le dernier dans celui du Provisioning.

♦ *Objet In-Interface (IN-Int)*

Les objets IN-int servent à indiquer l'interface (rappelons qu'un PEP gère à priori plusieurs interfaces) concernée par la requête du PEP et la décision du PDP. Ils permettent aussi de donner l'adresse d'origine du message reçu ayant déclenché la requête. Il existe donc deux formats pour cet objet selon qu'il s'agit d'une adresse IPv4 ou IPv6.

♦ *Objet Out-Interface (OUT-Int)*

Les objets OUT-Int sont identiques aux objets IN-Int sauf qu'ils concernent les messages sortants.

♦ *Objet Reason (Reason)*

Il est possible qu'un état installé ne puisse pas, ou plus, être maintenu par le PEP par exemple à cause d'une panne sur une de ses interfaces. Dans ce cas, le PEP doit avertir le PDP par un message DRQ (page 30). L'objet Reason, utilisé uniquement dans ces messages, permet d'indiquer la raison pour laquelle l'état n'est plus applicable.

Les valeurs possibles sont :

- 1 = Unspecified
- 2 = Management
- 3 = Preempted (Another request state takes precedence)
- 4 = Tear (Used to communicate a signaled state removal)
- 5 = Timeout (Local state has timed-out)
- 6 = Route Change (Change invalidates request state)
- 7 = Insufficient Resources (No local resource available)
- 8 = PDP's Directive (PDP decision caused the delete)
- 9 = Unsupported decision (PDP decision not supported)
- 10= Synchronize Handle Unknown
- 11= Transient Handle (stateless event)
- 12= Malformed Decision (could not recover)
- 13= Unknown COPS Object from PDP

Il est de plus possible de donner des précisions supplémentaires grâce à des sous-raisons. Ces sous-raisons sont à définir dans des extensions de COPS.

♦ *Objet Decision (Decision)*

Les objets Decision servent à indiquer la nature du message envoyé au PEP par le PDP. Ils permettent par exemple de spécifier si le message a été envoyé pour répondre à une requête du PEP, s'il s'agit d'une installation ou d'une suppression d'une configuration, s'il s'agit de l'acceptation ou du refus d'admission d'un flux…

Les décisions dépendant fortement de la nature des politiques et de leur gestion, le format de ces objets est souvent augmenté dans des extensions de COPS afin de s'en servir autrement. Ils sont cependant toujours utilisés dans des messages DEC et concernent donc toujours des décisions du PDP.

♦ *Objet LPDP Decision (LPDPDecision)*

Les objets LPDPDecision possèdent le même format que les objets Decision. Ils servent à transmettre, dans une requête du PEP, la décision prise par le LPDP (décision locale qui aide le PDP à prendre une décision globale).

♦ *Objet Error (Error)*

Les objets Error servent à identifier la nature d'une erreur lors d'une prise de décision du PDP.

Les code-erreurs possibles sont :

- 1 = Bad handle
- 2 = Invalid handle reference
- 3 = Bad message format (Malformed Message)
- 4 = Unable to process (server gives up on query)
- 5 = Mandatory client-specific info missing
- 6 = Unsupported client-type
- 7 = Mandatory COPS object missing
- 8 = Client Failure
- 9 = Communication Failure
- 10= Unspecified
- 11= Shutting down
- 12= Redirect to Preferred Server
- 13= Unknown COPS Object
- 14= Authentication Failure
- 15= Authentication Required

Il est également possible de donner des précisions supplémentaires grâce à un champ de sous-code d'erreur. Ces sous-codes sont définis dans des extensions de COPS.

♦ *Objet Client Specific Information (ClientSI)*

Les objets ClientSI sont très importants pour les extensions de COPS. Il est intéressant de préciser les notions de Client et de Client-Type dans COPS pour le comprendre.

La notion de Client est identique dans la terminologie du PBNM et dans celle de COPS. Dans les deux cas, un Client désigne simplement un PEP. Cette appellation provient de l'architecture des PBNs basée sur un modèle client/serveur où le serveur est le PDP et les clients sont les PEPs. La notion de Client-Type, elle, n'existe que dans la terminologie de COPS. Elle y désigne un service (QoS par DiffServ, QoS par IntServ, sécurité…) présent sur un Client. Cette appellation n'est peut être pas très heureuse car elle peut laisser croire qu'un Client, donc un PEP, ne peut gérer qu'un seul service. Ce n'est bien sûr pas le cas. Comme cela a déjà été dit à la section 2.1.3, un PEP peut s'occuper de plusieurs services. Autrement dit, un Client peut avoir plusieurs Client-Types.

Les objets ClientSI permettent de transporter des informations spécifiques à un Client-Type donné. Par exemple, certains des nouveaux objets définis par COPS-PR sont transportés dans des objets ClientSI (voir Annexe A). Les objets ClientSI sont donc une sorte de boites vides que chaque Client-Type peut utiliser comme il l'entend pour augmenter les possibilités de COPS. D'où leur importance.

Le format des objets ClientSI dépend complètement du Client-Type.

♦ *Objet Keep-Alive Timer (KATimer)*

Les objets KATimer servent à indiquer l'intervalle maximum en secondes 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.

La valeur 0 représente l'infini.

♦ *Objet PEP Identification (PEPID)*

Les objets PEPID servent à indiquer, de manière unique au sein d'un domaine de politiques, l'identité du PEP auquel le message s'adresse. Cette identité doit être persistante au redémarrage du PEP. Cela peut être par exemple une adresse IP ou un nom DNS assigné statiquement.

♦ *Objet Report-Type (Report-Type)*

Les objets Report-Type servent à indiquer, dans un message de rapport du PEP (RPT), la nature de celui-ci :

- succès
- échec
- comptabilité

♦ *Objet PDP Redirect Address (PDPRedirAddr)*

Il est possible que le PDP ferme une connexion vers un PEP pour un Client-Type donné (message CC). Les objets PDPRedirAddr servent à indiquer au PEP l'adresse d'un autre PDP (principalement, un PDP spécialisé pour ce Client-Type ou le PDP principal qui vient de récupérer d'une panne) auquel le PEP peut se reconnecter.

Il existe deux formats différents pour ces objets suivant que l'adresse soit en IPv4 ou en IPv6.

♦ *Objet Last PDP Address (LastPDPAddr)*

Quand un PEP se connecte à un PDP (message OPN) pour un Client-Type particulier, il est souhaitable qu'il indique dans son message l'adresse du dernier PDP auquel il a réussi à se connecter depuis son dernier redémarrage. Cette information peut être utile par exemple en cas de panne (voir 3.3.3). S'il s'agit de la première connexion du PEP après redémarrage, aucun objet LastPDPAddr ne doit être envoyé.

Il existe deux formats différents pour ces objets suivant que l'adresse soit en IPv4 ou en IPv6.

♦ *Objet Accounting Timer (AcctTimer)*

Les objets AcctTimer servent à indiquer au PEP l'intervalle minimum en secondes entre deux rapports (messages RPT). Bien que cette interdiction ne soit pas absolue (le PEP peut passer outre dans certains cas de figure), cela permet au PDP de contrôler la quantité de trafic généré par la comptabilité.

La valeur 0 permet d'interdire les rapports non explicitement demandés par le PDP.

♦ *Objet Message Integrity (Integrity)*

Les objets Integrity servent à authentifier et valider l'intégrité d'un message COPS. Ces objets doivent toujours se placer à la fin du message. Ils contiennent :

- Un numéro de séquence pour éviter les attaques par rejoue
- Un condensé (Digest) du message pour garantir l'intégrité
- Un identificateur de clef (Key ID) pour indiquer quelle clef et quel algorithme utiliser pour vérifier le Digest. En effet, il est possible, et même fortement recommandé, de changer

régulièrement la clef et même l'algorithme de hachage utilisé pour calculer le Digest afin d'éviter par exemple les attaques par force brute.

Le Digest est calculé sur tout le message excepté le Digest lui-même (mais incluant l'entête de l'objet Integrity, le Key ID et le numéro de séquence). L'utilisation de cet objet est décrite section 3.3.1.2.

Toutes les implémentations du PEP et du PDP doivent, au minimum, pouvoir calculer le Digest grâce à HMAC-MD5-96 qui est un mécanisme HMAC (Keyed-Hashing for Message Authentication) [9] utilisant l'algorithme MD5 [10] et ne gardant que les 96 premiers bits du résultat.

3.2.2 Les messages

Les messages COPS sont constitués d'un entête (Figure 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 L'entête

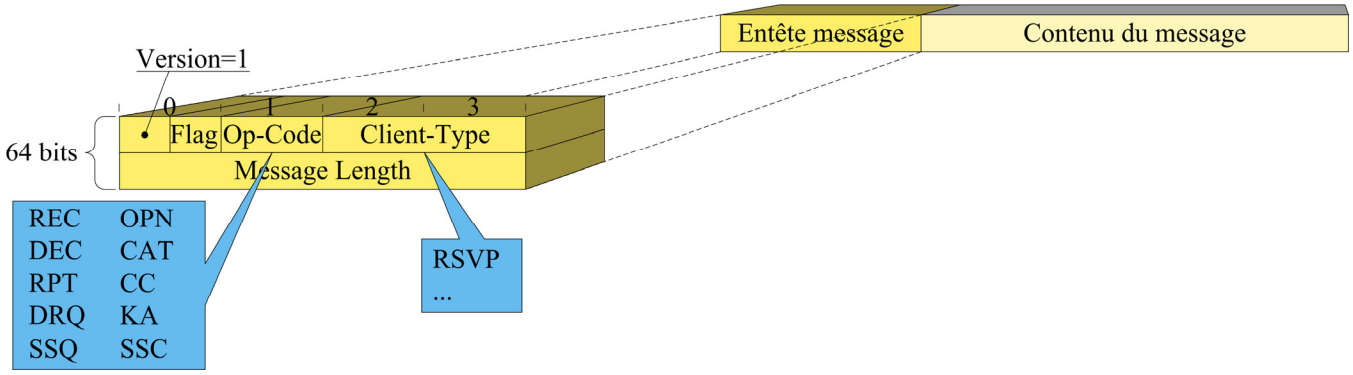


Figure 11 : Entête d'un message COPS

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 : Nature de l'opération réalisée par ce message.
  - 1 = Request (REQ)
  - 2 = Decision (DEC)
  - 3 = Report State (RPT)
  - 4 = Delete Request State (DRQ)
  - 5 = Synchronize State Req (SSQ)
  - 6 = Client-Open (OPN)
  - 7 = Client-Accept (CAT)
  - 8 = Client-Close (CC)
  - 9 = Keep-Alive (KA)
  - 10= Synchronize Complete (SSC)

- Client-type : Identifie le Client-Type du Client. Aujourd'hui seul RSVP est normalisé et vaut 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. La valeur 0 est réservée soit aux messages Keep-Alive soit pour mettre en place une connexion sécurisée.
- 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 Les opérations

Cette section présente brièvement le rôle chaque message.

3.2.2.2.1 Opérations courantes

Les opérations courantes permettent de mettre en place les interactions de type question/réponse entre le PEP et le PDP.

♦ *Request (REQ) PEP -> PDP*

Deux situations peuvent amener le PEP à envoyer un message REQ.

- Soit il est confronté à un message qu'il ne sait pas traiter. Dans ce cas, il demande une décision au PDP. Puisque cette décision va engendrer un nouvel état, il indique dans le message un Handle libre à lui asssocier.
- Soit une modification du PEP (par exemple une panne d'interface) entraîne un changement dans un état déjà établis (déjà associé à un Handle). Le PEP doit alors immédiatement informer le PDP de ce changement via un message REQ.

Dans tous les cas, le message contient donc un objet Handle. Il contient également toujours un objet Context pour savoir comment interpréter les autres objets.

Si le PDP reçoit un message REQ mal-formé, il doit répondre au PEP par un message DEC contenant le code d'erreur approprié dans un objet Error.

♦ *Decision (DEC) PDP -> PEP*

Lorsque le PDP reçoit un message REQ, il y répond par un message DEC contenant le même Handle. Si la requête était mal formée, le message DEC contient un objet Error. Sinon, il contient un ou plusieurs groupes d'objets constitués de :

- Un objet Context pour savoir comment interpréter les autres objets de ce groupe.
- Un objet Decision de type Flag pour indiquer la nature de la décision :

  Aucune décision

  Installation

  Suppression
- Si le premier objet Decision n'indique pas 'aucune décision', un ou plusieurs autres objets Decision permettent de préciser la décision prise.

Le PDP peut également, dans certains cas, envoyer des messages DEC non sollicité par le PEP, c'est à dire ne répondant pas directement à un message REQ. Ces messages non sollicités servent à modifier un état déjà établi. Cela peut être utile par exemple si le réseau entre dans un état congestionné.

Si le PEP reçoit un message DEC mal-formé ou qu'il ne peut pas comprendre, il doit retourner au PDP un message DRQ spécifiant l'erreur adéquate dans un objet Reason.

♦ *Report State (RPT) PEP -> PDP*

Les messages RPT sont utilisés par le PEP pour indiquer au PDP si sa décision a pu être mise en place ou non. Ils peuvent également servir à envoyer périodiquement des informations de comptabilité.

Ils contiennent :

- Toujours un objet Handle pour savoir quel état est concerné par le rapport.
- Toujours un objet Report-type pour indiquer la nature du rapport : succès, échec ou comptabilité.
- Eventuellement un objet ClientSI, dépendant du Client-Type du PEP, pour donner des informations supplémentaires ou des mises en garde.

♦ *Delete Request State (DRQ) PEP -> PDP*

Les messages DRQ servent à indiquer au PDP que le PEP ne peut plus appliquer un état installé et qu'il l'a donc supprimé. Il est indispensable de signaler cela au PDP pour maintenir une synchronisation parfaite car le PDP conserve en mémoire tous les états tant qu'ils n'ont pas été explicitement supprimés par le PEP ou que la connexion n'a pas été fermée.

Un message DRQ contient toujours un objet Handle pour identifier l'état supprimé et un objet Reason pour expliquer la raison de cette suppression et aider le PDP à prendre éventuellement des mesures adéquates.

3.2.2.2.2 Gestion de la session

Dans ce document, on appelle session l'espace de dialogue, de niveau applicatif, entre le PEP et le PDP. Un PEP a autant de sessions ouvertes qu'il supporte de Client-Types.

♦ *Client-Open (OPN) PEP -> PDP*

Le PEP envoie au PDP un message OPN pour ouvrir une session pour un Client-Type donné. Il envoie donc autant de messages OPN qu'il supporte de Client-Types, et éventuellement un de plus s'il désire sécuriser les sessions (Client-type = 0).

Dans ce message, le PEP indique son identité dans un objet PEPID. Il peut aussi spécifier l'adresse du dernier PDP dont il conserve encore des états en mémoire dans un objet LastPDPAddr et fournir d'autres informations globales relatives à son Client-Type dans un objet ClientSI.

Si le message OPN reçu par le PDP est mal-formé, le PDP doit répondre par un message CC contenant le code d'erreur approprié (Bad message format) dans un objet Error.

♦ *Client-Accept (CAT) PDP -> PEP*

Les messages CAT permettent au PDP d'indiquer au PEP que sa demande d'ouverture de session a été acceptée. Dans ce cas, le PDP précise obligatoirement le Keep-Alive Timer (objet KATimer page 27). Il peut éventuellement préciser le Accounting Timer (objet ACCTTimer page 27).

Si le PEP reçoit un message CAT mal-formé, il doit retourner au PDP un message CC spécifiant le code d'erreur adéquate.

♦ *Client-Close (CC) PEP -> PDP, PDP -> PEP*

Les messages CC peuvent être utilisés dans deux situations :

- A la réception d'un message OPN, si celui-ci est mal-formé ou si le PDP ne peut accepter cette nouvelle session (par exemple parce qu'il ne supporte pas le Client-Type ou qu'il manque de ressource mémoire ou autre…).
- Après établissement de la connexion, lorsque l'un des participants, le PEP ou le PDP, ne supporte plus le Client-Type en cours.

Les messages CC contiennent toujours un objet Error pour indiquer la raison d'être du message. Lorsqu'ils sont émis par le PDP, ils peuvent en plus contenir un objet PDPRedirAddr pour rediriger le PEP vers un nouveau PDP.

♦ *Keep-Alive (KA) PEP -> PDP, PDP -> PEP*

Les messages KA permettent au PDP et au PEP de s'assurer que la connexion TCP, supportant une ou plusieurs sessions, fonctionne correctement. Pendant les périodes de silence (où aucun message n'est échangé), le PEP doit envoyer un message KA à des instants choisis aléatoirement entre ¼ et ¾ du Keep-Alive Timer. Lorsque le PDP reçoit ce message, il doit y répondre également par un message KA.

Lorsque l'un des participants n'a pas reçu de message (KA ou autre) depuis plus longtemps que le Keep-Alive Timer, il doit considérer que l'autre participant n'est plus joignable (le problème peut venir du réseau ou de l'autre participant).

Le champ Client-Type des messages KA doit toujours être mis à 0 car ces messages concernent toutes les sessions ouvertes au-dessus d'une connexion TCP entre un PEP et un PDP.

3.2.2.2.3 Synchronisation des états

Comme cela a déjà été dit plus haut, COPS est un protocole "stateful", c'est à dire basé sur des états. Les opérations courantes (3.2.2.2.1) permettent de créer, modifier et supprimer des états. Cependant elles ne sont pas suffisantes pour garder le PEP et le PDP synchronisés en cas de défaillance (panne réseau, anomalie dans le fonctionnement du PDP ou du PEP…).

♦ *Synchronize State Request (SSQ) PDP -> PEP*

Les messages SSQ permettent au PDP de demander au PEP une synchronisation de leurs états pour un Client-Type donné. Si le message contient un objet Handle, la synchronisation ne porte que sur l'état identifié par cet Handle, sinon, elle porte sur tous les états.

Ayant reçu un message SSQ, le PEP retourne un message REQ pour chaque état qu'il a à synchroniser. Une fois tous ces messages REQ envoyés, il termine par un message SSC.

♦ *Synchronize State Complete (SSC) PEP -> PDP*

Les messages SSC permettent au PEP d'indiquer au PDP qu'il a terminé la synchronisation demandée dans un message SSQ. Si ce message SSQ contenait un objet Handle, alors le message SSC en contient également un. Sinon il n'en contient pas.

3.2.2.3 Relations entre les messages et les objets

Figure 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.

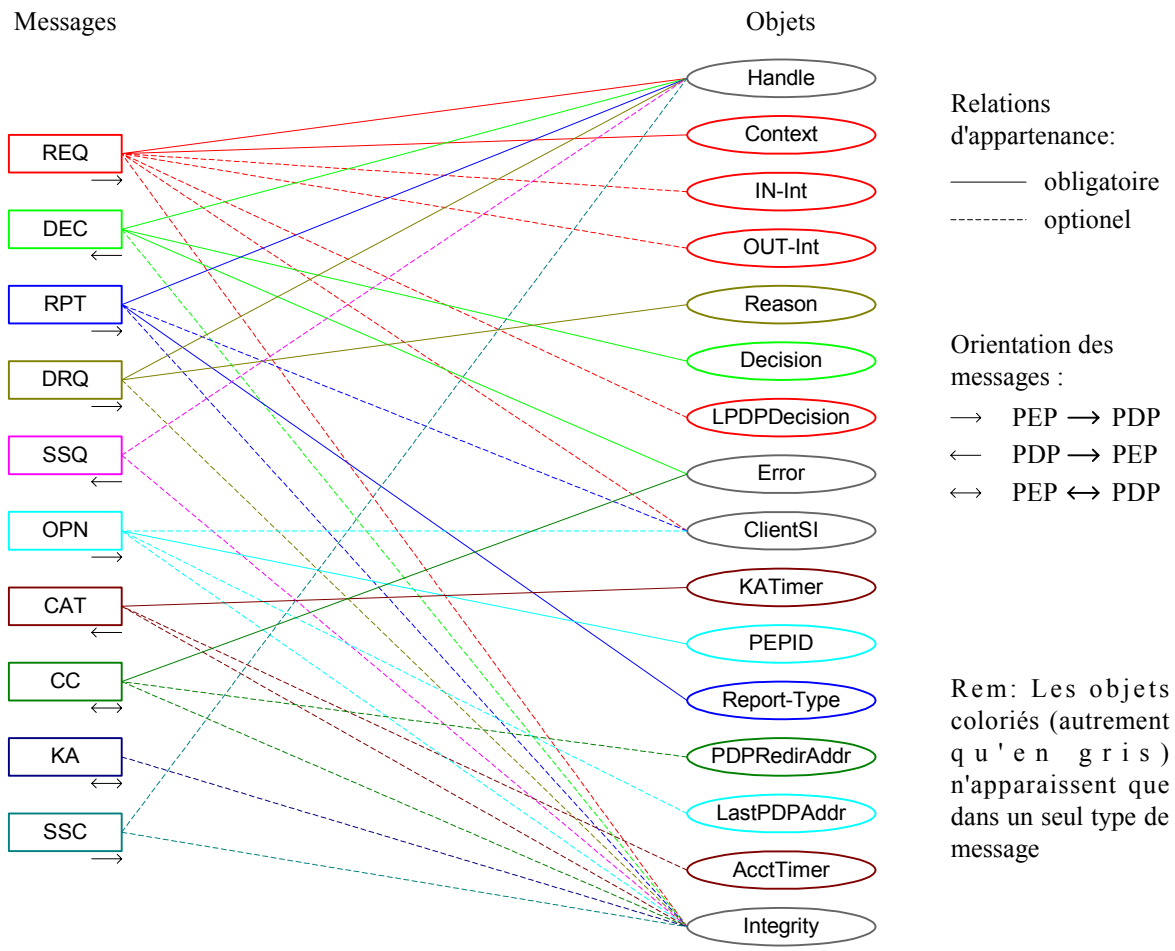


Figure 12 : Relations entre messages et objets COPS

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*

Comme cela a déjà été dit, COPS est basé sur TCP. Donc avant d'ouvrir une session COPS, il faut d'abord ouvrir une connexion TCP comme le montre Figure 13.

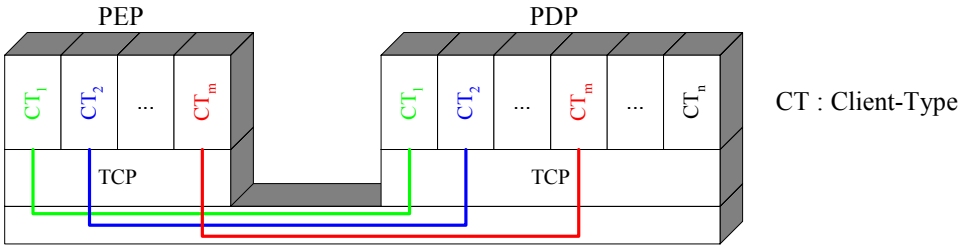


Figure 13 : Sessions COPS et connexions TCP

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 a 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.

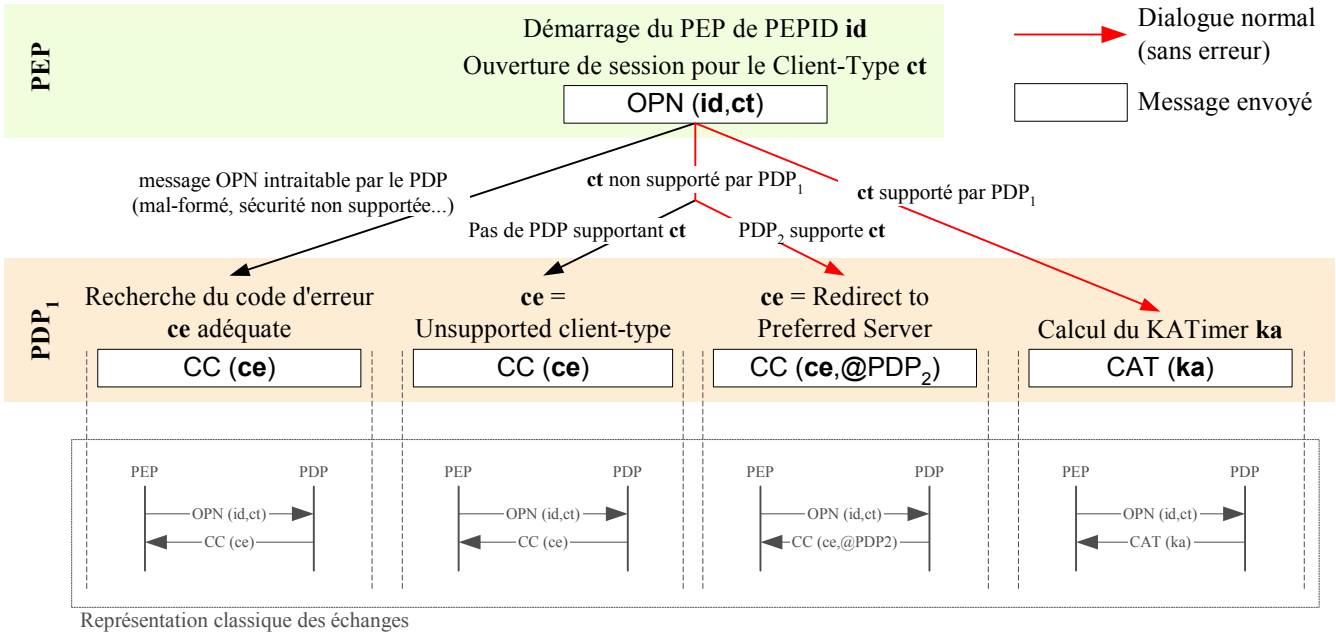


Figure 14 : Echanges COPS : Ouverture de session

Pour chaque message OPN, s'il est bien formé et que le PDP supporte ce Client-Type, alors 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 (envoie 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 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 15.

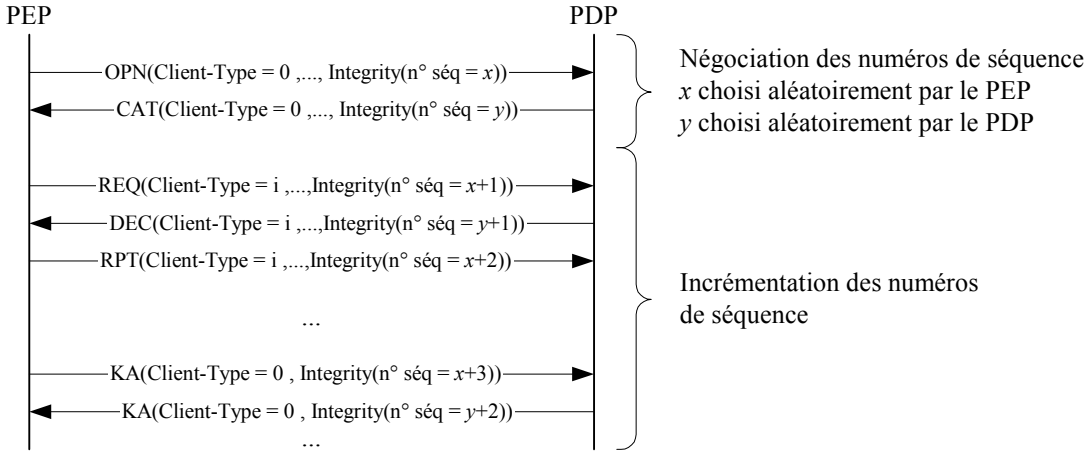


Figure 15 : Echanges COPS : Négociation des numéros de séquence

♦ *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, on décrit les échanges de messages sans entrer dans les détails de leur contenu propre à chaque modèle de gestion de politiques (Provisioning ou Outsourcing).

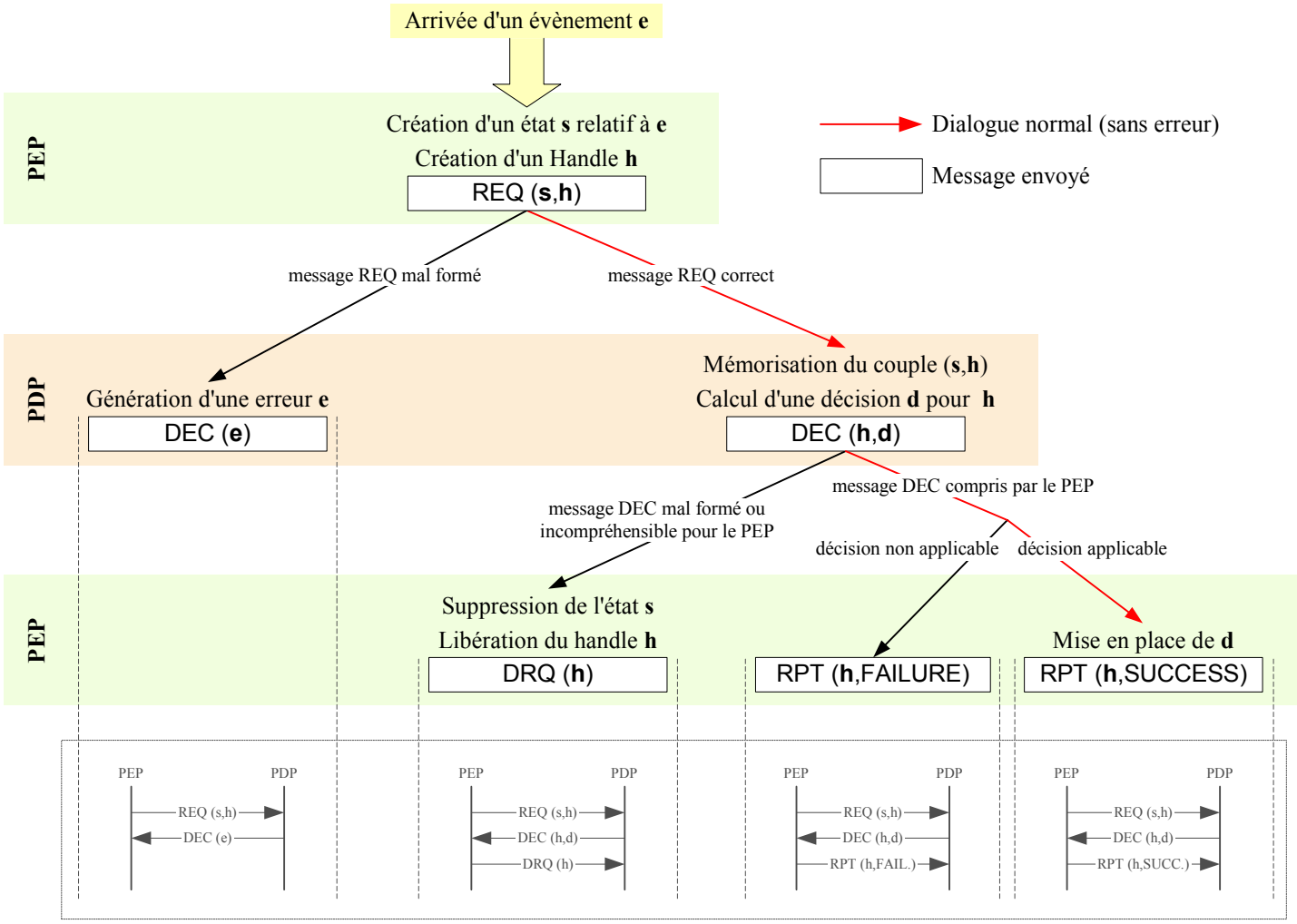
3.3.2.1 La configuration du réseau

Pour éviter toute confusion, il faut signaler que 4.2présente 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 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.



Représentation classique des échanges

Figure 16 : Echanges COPS : Création d'un état

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 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.

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,
- …

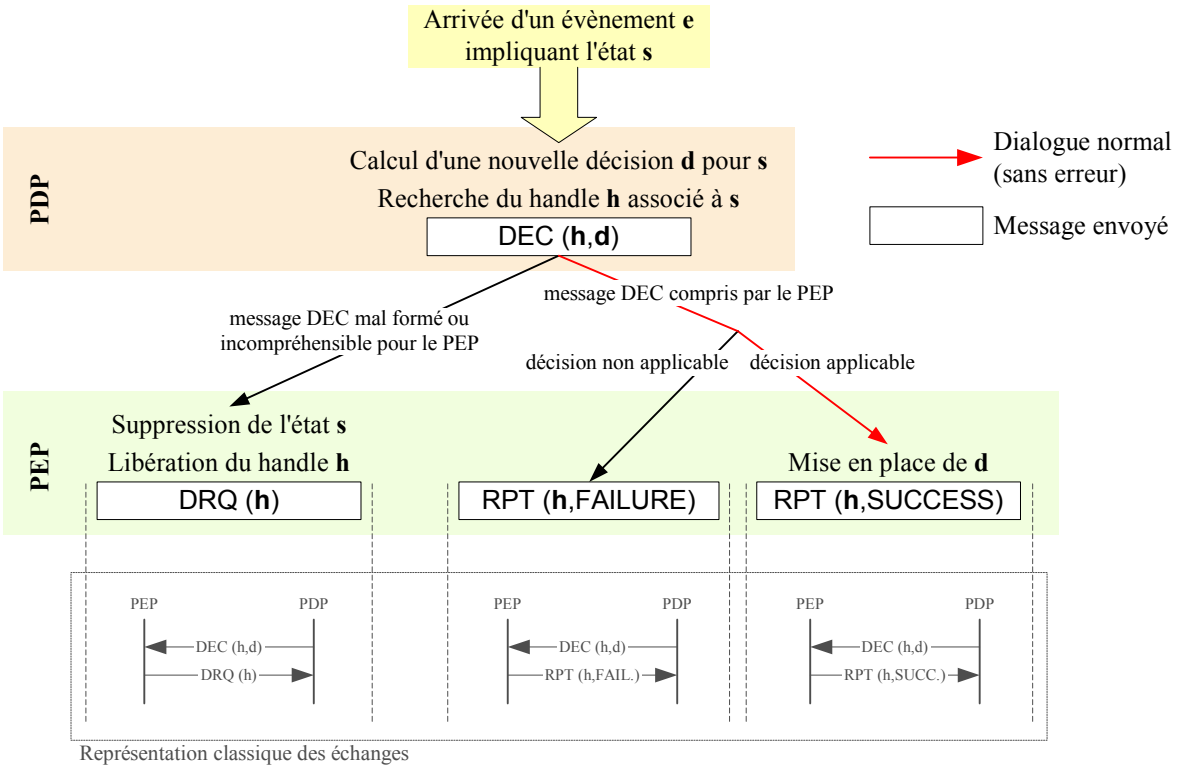
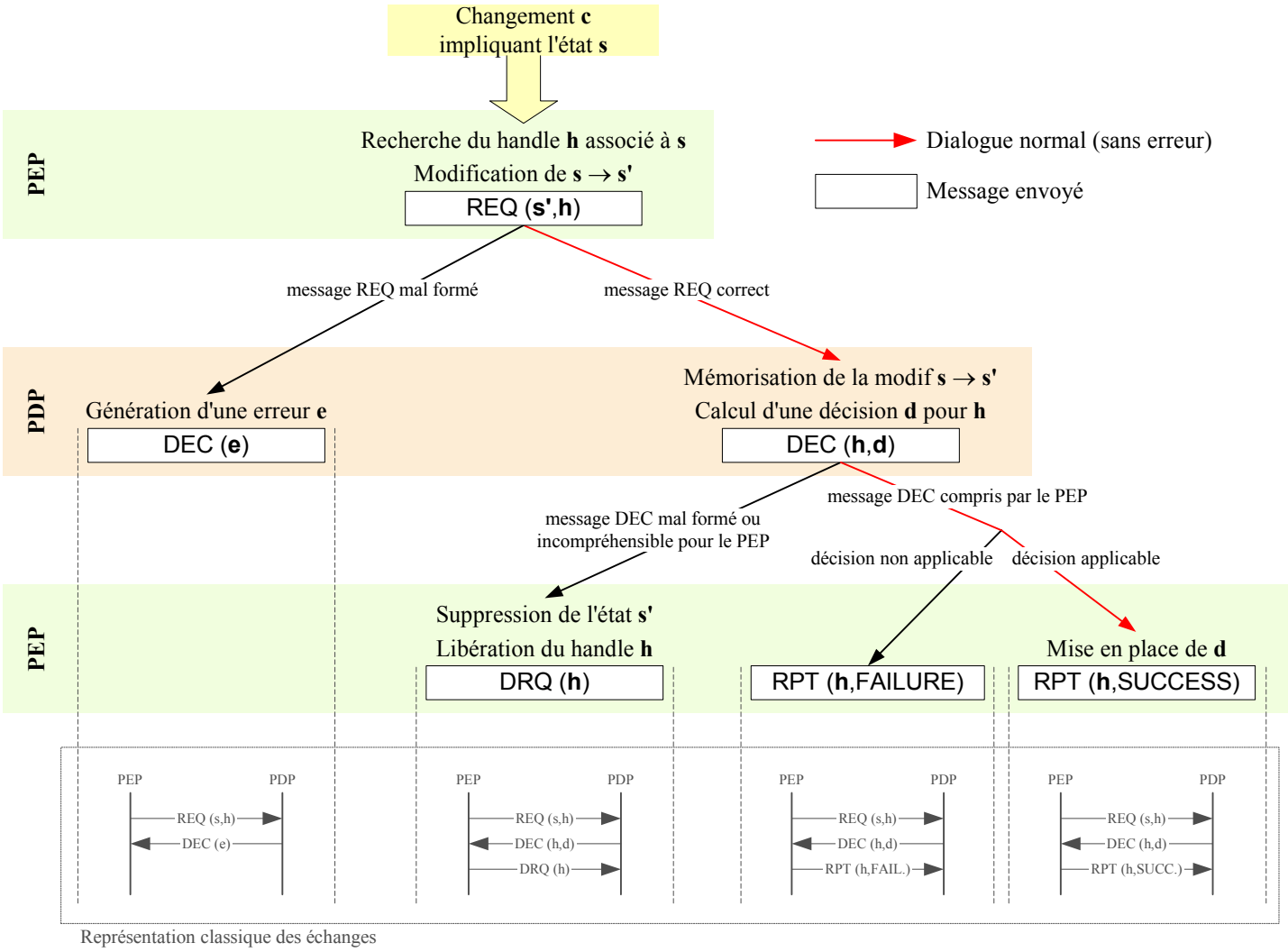


Figure 17 : Echanges COPS : Modification d'un état par le PDP

♦ *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
- …



Représentation classique des échanges

Figure 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.

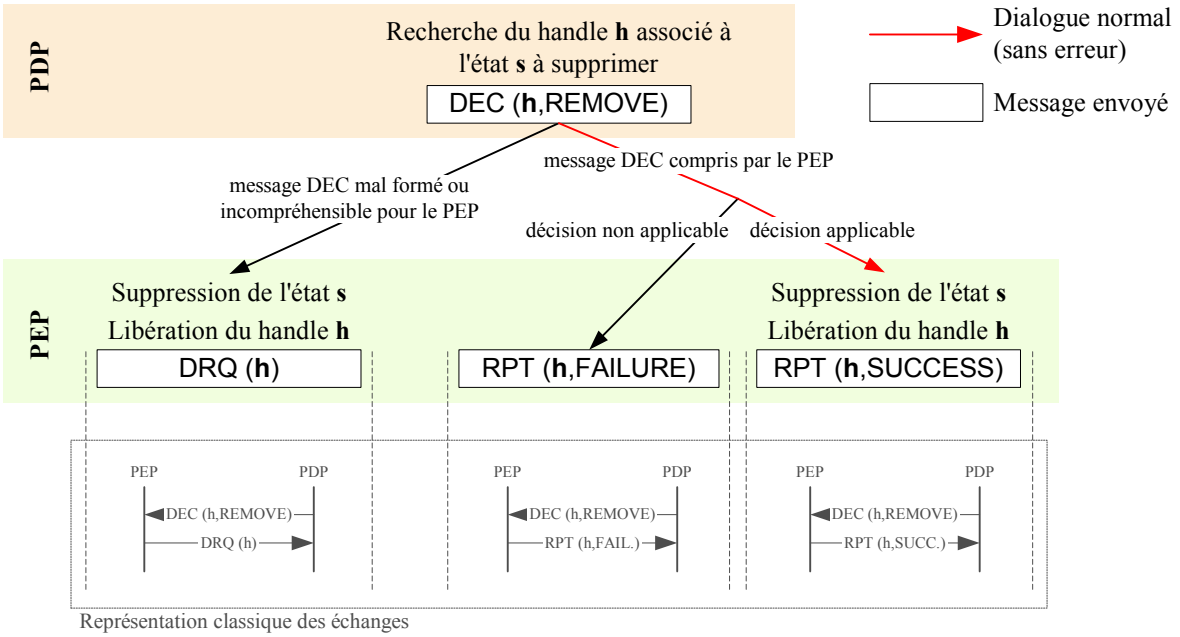


Figure 19 : Echanges COPS : Suppression d'un état par le PDP

Il est intéressant d'apporter quelques précisions à Figure 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 envoie 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.

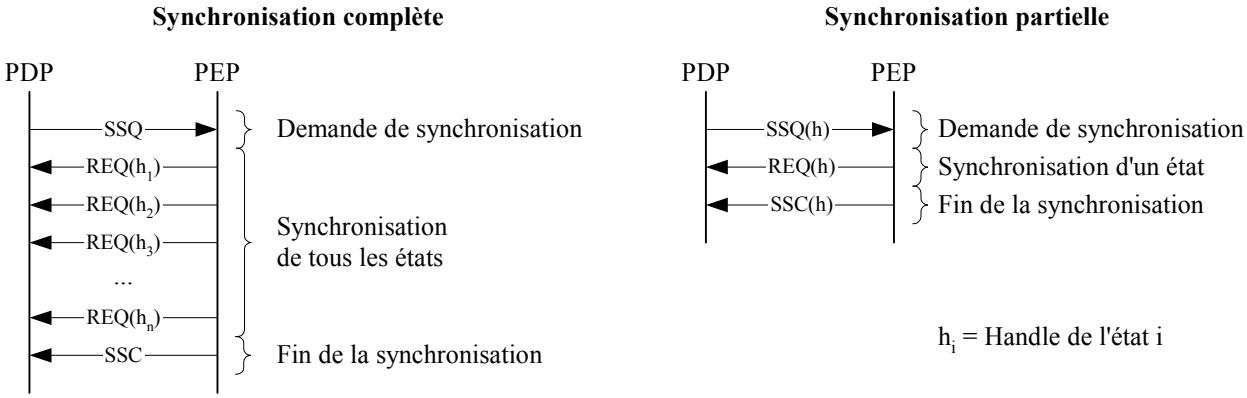


Figure 20 : Echanges COPS : Synchronisation

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 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. Comme cela a déjà été dit plus haut il existe d'autres langages pour spécifier des règles (Ponder [32], PFDL [33]…). Cependant, seul PCIM sera abordé dans ce document car il a été développé en étroite collaboration avec le WG (Working Group) RAP qui est à l'origine de COPS et qui a défini le cadre de travail du PBNM.

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

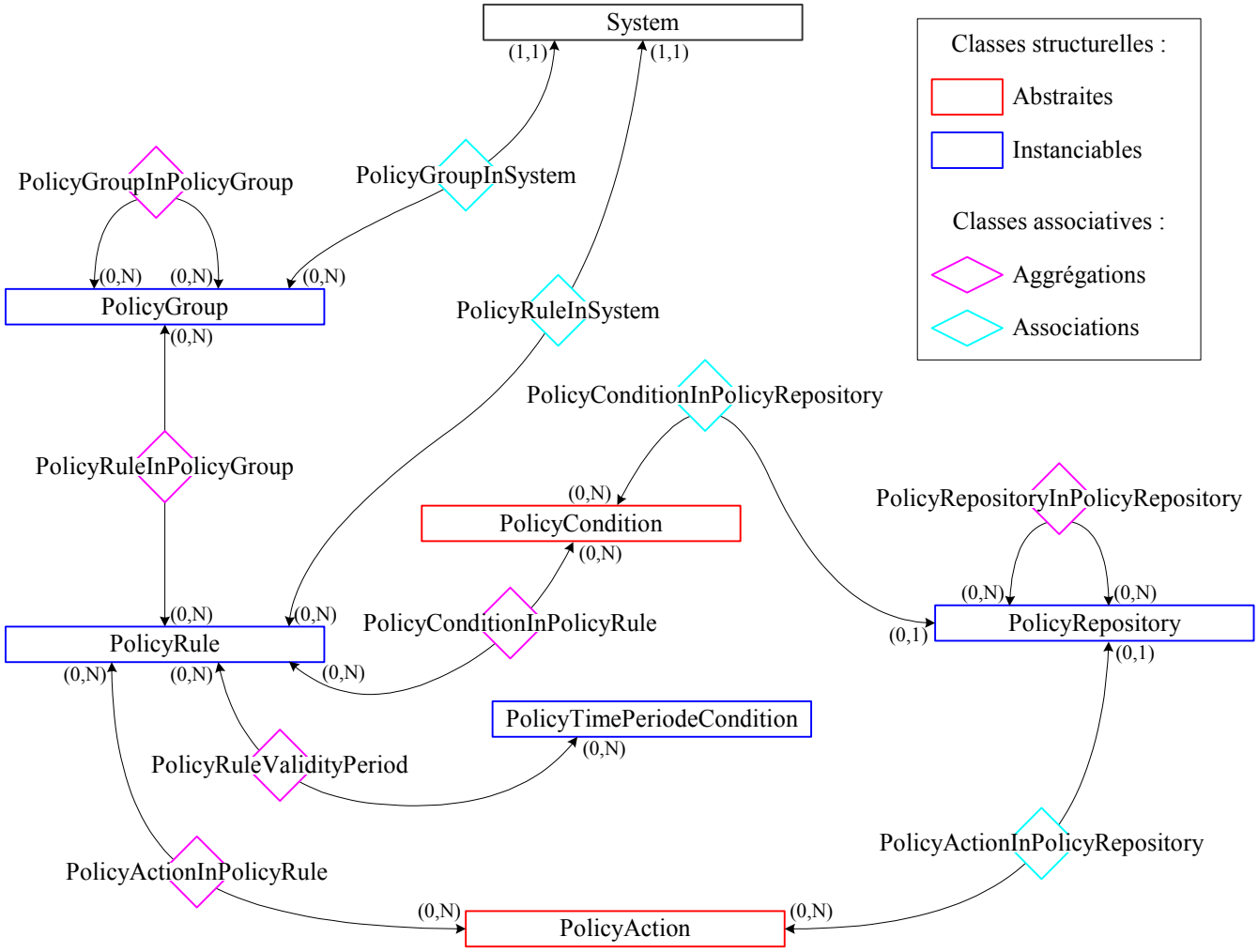


Figure 21 : Relations entre les classes PCIM

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.

4.1.3.1 Les classes structurelles

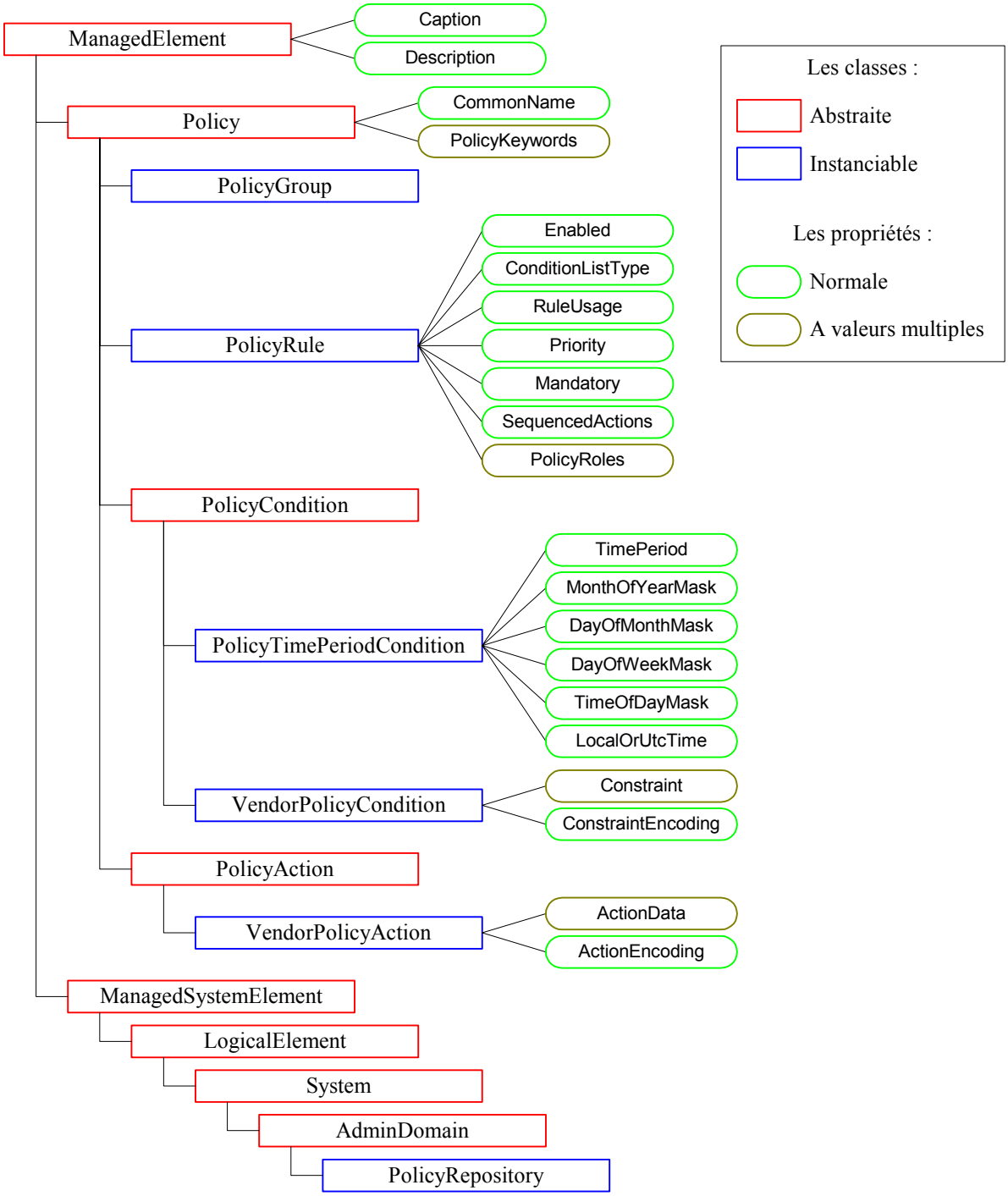


Figure 22 : Hiérarchie des classes structurelles PCIM

Figure 22 présente succinctement toutes les classes structurelles de PCIM avec leur ordre hiérarchique et leurs propriétés. La suite de la section présente un peu plus en détail ces classes.

♦ *La classe abstraite Policy*

La classe Policy est racine de toutes les classes structurelles de politiques. Elle dérive de la classe CIM ManagedElement dont elle hérite les propriétés Caption (description de la classe en une ligne) et Description

(description complète de la classe). Etant abstraite, la classe Policy n'a pas pour objectif d'être instanciée directement mais permet d'établir une base commune pour toutes les classes représentant des objets de politiques.

Elle fournit à toutes ces classes deux propriétés:

- La propriété CommonName (CN) indique un nom convivial pour désigner un objet de politiques.
- La propriété PolicyKeywords fournit un ensemble de un ou plusieurs mots clef pour caractériser et classer un objet de politiques. Les mots clef suivants sont normalisés : "UNKNOWN", "CONFIGURATION", "USAGE", "SECURITY", "EVENT", "SERVICE", "MOTIVATIONAL", "INSTALLATION" et "POLICY".

♦ *La classe PolicyGroup*

La classe PolicyGroup permet de mettre en place des groupes de règles de politiques afin de les structurer. Un groupe peut contenir soit un ensemble de règles, soit un ensemble de groupes.

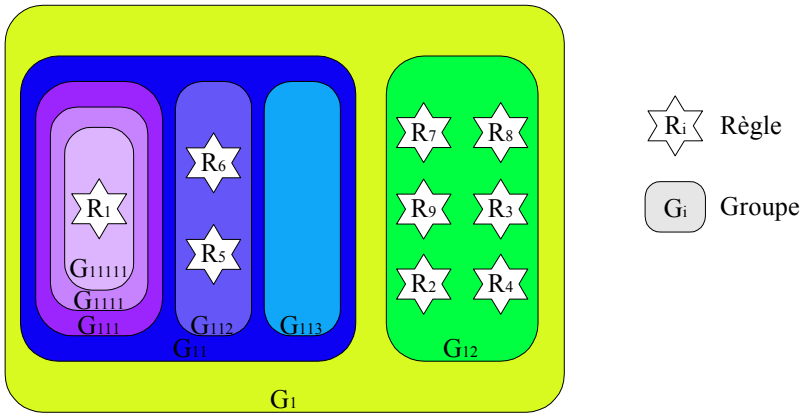


Figure 23 : Exemple de groupes de politiques

Les seules limites pour l'imbrication des groupes (Figure 24) sont :

- Un groupe ne peut pas contenir à la fois des groupes et des règles (cf. G_{11}).
- Les boucles ne sont pas autorisées (cf. G_{111}).

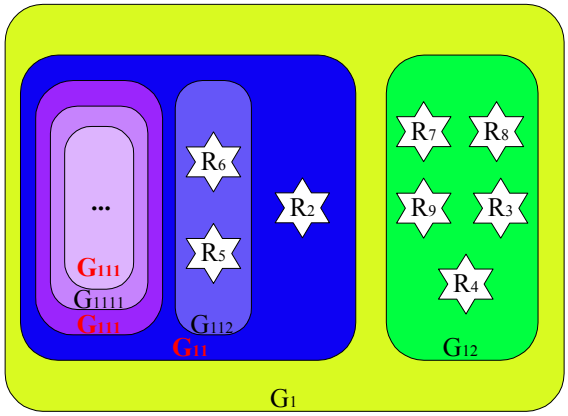


Figure 24 : Exemple de groupes de politiques interdits

♦ *La classe PolicyRule*

La classe PolicyRule permet de créer des règles de politiques. Ces règles sont constituées d'un ensemble de conditions (objets de type PolicyCondition) et d'un ensemble d'actions (objets de type PolicyAction). Il faut

signaler que ces ensembles peuvent être vides si les attributs de la règle (son nom, sa description…) sont suffisamment parlant pour que les conditions ou les actions soient implicites.

La classe Policyrule définit sept propriétés :

- La propriété Enabled sert à désactiver la politique sans la supprimer. Elle fournit également des facilités pour le débugage.
- La propriété ConditionListType indique si la liste des conditions est présentée sous forme normale conjonctive ou sous forme normale disjonctive (voir exemple page 52).
- La propriété RuleUsage donne des indications sur la manière d'utiliser cette règle. Le format de cette propriété est libre.
- La propriété Priority indique la priorité de la règle par rapport aux autres règles de la politique. Ce mécanisme permet de résoudre certains types de conflits.
- La propriété Mandatory spécifie si la règle est obligatoire ou facultative. Dans ce dernier cas, c'est l'interpréteur de PCIM qui fera le choix de l'appliquer ou non.
- La propriété SequencedActions indique si l'interpréteur doit suivre l'ordre des actions (défini dans la classe PolicyActionInPolicyRule) associées à la règle. La propriété peut prendre trois valeurs : Mandatory, Recommended et DontCare.
- La propriété PolicyRoles indique le ou les rôles de la règle. Lorsqu'elle contient plusieurs rôles (on parle alors de combinaison de rôles), ces rôles sont classés par ordre alphabétique. Cette propriété est de type "multi-valued". Cela signifie qu'il est possible d'associer plusieurs combinaisons de rôles à une seule règle.

Le concept de rôle est important dans PCIM et mérite d'être précisé. Formellement, un rôle peut être défini de la façon suivante :

> Un rôle est un type d'attribut qui est utilisé afin de sélectionner, pour un ensemble particulier de composants du réseau, une ou plusieurs politiques dans un vaste ensemble de politiques.

Ainsi, par exemple si une interface a les rôles "Campus" et "Ethernet", alors elle sera soumise aux règles qui ont un rôle "Campus", ainsi qu'à celles qui ont un rôle "Ethernet" et enfin, à celles qui ont une combinaison de rôles "Campus+Ethernet".

L'utilisation des rôles permet également de résoudre certains conflits. En effet, il est possible de préciser que certains rôles sont incompatibles (par exemple, "Ethernet" et "FrameRelay"). Dès lors, toute règle ou interface possédant des rôles incompatibles soulève une erreur.

♦ *La classe abstraite PolicyCondition*

Une instance de la classe PolicyCondition représente une condition élémentaire. Une règle de politique associe généralement plusieurs conditions élémentaires en les combinant soit sous forme normale conjonctive, soit sous forme normale disjonctive. Dans le cas d'une combinaison, seule l'évaluation globale est d'importance pour PCIM, les évaluations individuelles ne sont jamais prises en compte.

Une condition élémentaire peut être dédiée à une règle ou être partagée par plusieurs règles.

La classe PolicyCondition est abstraite. Elle n'a pas pour but d'être instanciée directement. D'ailleurs, elle ne définit aucune propriété qui lui permettrait de contenir de "vraies" conditions. Les propriétés nécessaires sont à définir dans les sous-classes de PolicyCondition comme PolicyTimePeriodCondition.

♦ *La classe PolicyTimePeriodCondition*

La classe PolicyTimePeriodCondition permet de créer des conditions basées sur une horloge. Les instances de cette classe définissent des périodes pendant lesquelles les règles associées sont valides. En dehors de ces périodes, ces règles n'ont aucun effet.

La classe propose cinq propriétés pour définir ces périodes. Il n'est pas obligatoire d'utiliser toutes les propriétés en même temps. Si une propriété n'est pas utilisée, elle est traitée comme si sa valeur indiquait "toujours active". Si plusieurs propriétés sont utilisées simultanément, la condition n'est vraie que si elles sont toutes valides.

La classe définit en tout six (5+1) propriétés :

- La propriété TimePeriod spécifie une période de validité sur le calendrier. Par exemple, du 1er mars 2004 9h00 au 31 août 2004 19h35. Il est aussi possible que la période soit du type "A partir de maintenant, jusqu'à…" ou du type "A partir de …, jusqu'à l'infini.".
- La propriété MonthOfYearMask restreint la période à certains mois de l'année. Par exemple, janvier, mars et septembre.
- La propriété DayOfMonthMask restreint la période à certains jours du mois (pour n'importe quel mois). Par exemple, le 1er et le 3ème (en partant du début) et le dernier (le 1er en partant de la fin).
- La propriété DayOfWeekMask restreint la période à certains jours de la semaine. Par exemple, le lundi et le vendredi.
- La propriété TimeOfDayMask restreint la période à certaines heures du jour. Par exemple, de 21h30 57s à 8h00 00s le lendemain. Dans cet exemple, la période indiquée s'étend sur deux jours, si l'un de ces jours (par exemple le deuxième) est en dehors d'une période définie par une autre propriété, alors la période de TimeOfDayMask est restreinte au seul jour valide (c'est à dire ici, le premier jour de 21h30 57s à 23h59 59s).
- La propriété LocalOrUtcTime offre la possibilité de choisir entre une horloge locale et une horloge UTC.

Si une règle n'est associée à aucun objet PolicyTimePeriodCondition, alors elle est valide à tout moment. Elle peut cependant être désactivée par d'autres moyens.

♦ *La classe VendorPolicyCondition*

Le but de cette classe est de fournir un mécanisme générique pour permettre aux entreprises d'étendre PCIM selon leurs besoins. Les extensions standardisées de PCIM (telles que QPIM) ne sont pas sensées utiliser cette classe.

Les objets de cette classe peuvent représenter n'importe quel type de condition grâce à deux propriétés:

- La propriété ConstraintEncoding indique dans quel format la condition est encodée.
- La propriété Constraint représente une contrainte encodée selon le format spécifié par la propriété ConstraintEncoding. Constraint est de type "multi-valued", il est donc possible d'utiliser plusieurs contraintes pour créer une condition complète, cependant toutes ces contraintes doivent avoir le même format d'encodage.

♦ *La classe abstraite PolicyAction*

Une instance de la classe PolicyAction représente une action élémentaire. Une règle de politique peut être

associée à plusieurs actions élémentaires afin de réaliser une action plus complexe. Dans ce cas, PCIM offre la possibilité d'ordonner ces actions (voir la classe PolicyActionInPolicyRule page 53).

Une action élémentaire peut être dédiée à une règle ou être réutilisée dans plusieurs règles.

La classe PolicyAction est abstraite. Elle n'a pas pour but d'être instanciée directement. D'ailleurs, elle ne définit aucune propriété qui lui permettrait de contenir de "vraies" actions. Les propriétés nécessaires sont à définir dans ses sous-classes.

♦ *La classe VendorPolicyAction*

La classe VendorPolicyAction fournit un mécanisme générique pour permettre aux entreprises d'étendre PCIM selon leurs besoins. Les extensions standardisées de PCIM ne sont pas sensées utiliser cette classe.

Les objets de cette classe peuvent représenter n'importe quel type d'action grâce à deux propriétés :

- La propriété ActionEncoding indique dans quel format l'action est encodée.
- La propriété ActionData représente un élément d'action. Elle est de type "multi-valued", il est donc possible d'utiliser plusieurs de ces éléments pour spécifier une action complète, cependant tous ces éléments doivent être exprimés dans le même format.

♦ *La classe PolicyRepository*

La classe PolicyRepository est la seule classe ici à ne peut être dérivée de la classe Policy. Une instance de cette classe représente un Policy Repository.

Il faut noter que la définition d'un Policy Repository dans le contexte de PCIM est un peu particulière. [7] donne la définition suivante : « Un Policy Repository est un conteneur logique qui permet de stocker des informations relatives aux politiques (règles, conditions, actions et autres données importantes) **et** qui est associé à un domaine administratif bien défini. ».

4.1.3.2 Les classes associatives

Il existe deux types de classes associatives: les associations et les agrégations.

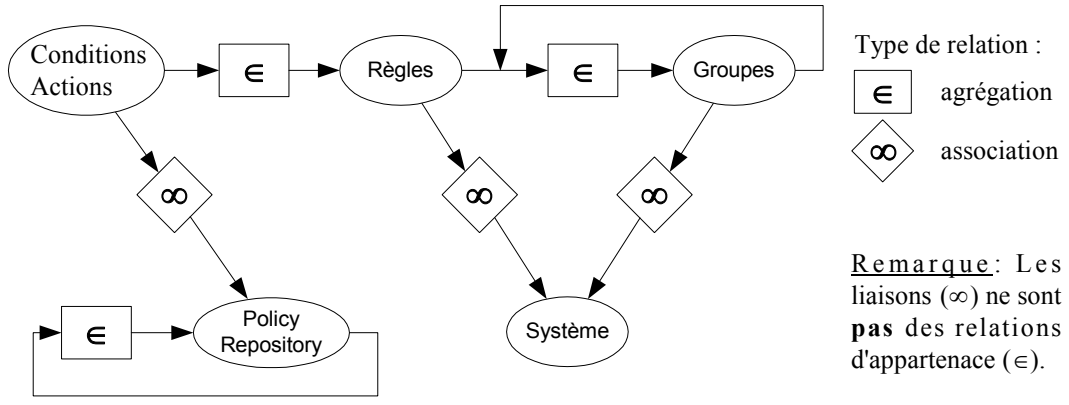


Figure 25 : Types de relations PCIM

Une association est une construction CIM représentant une relation entre deux (voire plus) objets. Elle est définie par une classe contenant typiquement deux références. Généralement une association est définie entre deux classes sans qu'aucune modification sur l'une ou l'autre de ces classes ne soit nécessaire.

Une agrégation est une forme d'association plus forte. Elle représente généralement une relation d'appartenance entre une entité contenant (par exemple, une règle de politique) et des entités contenues (par

exemple, des conditions). Une agrégation de ce type implique souvent, même si ce n'est pas obligatoire, que les objets agrégés sont mutuellement dépendants (les règles ne peuvent exister sans condition, explicite ou non, et les conditions n'ont d'utilité qu'au sein d'une règle).

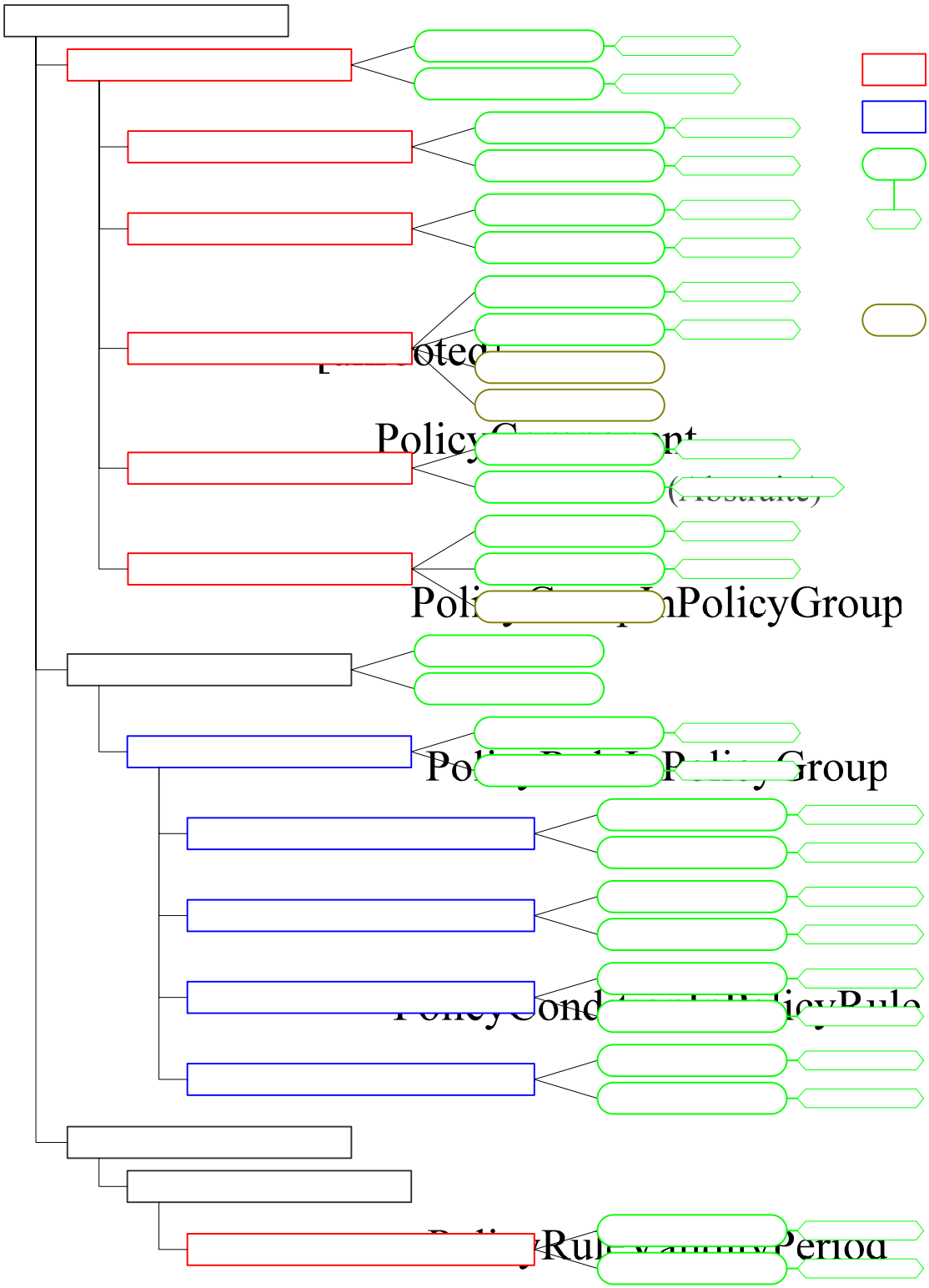


Figure 26 : Hiérarchie des classes associatives PCIM

4.1.3.2.1 Les agrégations

♦ *L'agrégation abstraite PolicyComponent*

La classe PolicyComponent est racine de toutes les agrégations relatives aux politiques. Etant abstraite, elle n'a pas pour objectif d'être instanciée directement mais permet d'établir une base commune pour toutes ses sous-classes et notamment d'imposer une sémantique commune pour leurs références.

Elle définit en effet deux références vers des objets de type Policy qui seront surchargées dans ses sous-classes (voir Figure 26). Ces références sont:

- La référence GroupComponent sert à désigner l'objet O_0 contenant l'objet O_1.
- La référence PartComponent sert à désigner sur l'objet O_1 contenu par l'objet O_0.

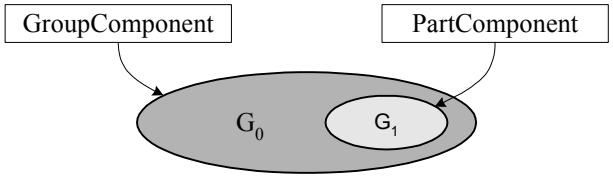
♦ *L'agrégation PolicyRuleInPolicyGroup*

Il a déjà été dit qu'une instance de la classe PolicyRule représente une règle de politique. Cette instance peut être utilisée seule. Cependant, afin que le modèle de gestion par politiques puisse passer à l'échelle, il est nécessaire de pouvoir regrouper cette règle avec d'autres règles afin de créer une nouvelle règle plus complexe. La classe PolicyRuleInPolicyGroup, avec l'aide de la classe PolicyGroupInPolicyGroup, permet donc d'établir des groupes structurés de règles élémentaires (voir Figure 23 page 47).

♦ *L'agrégation PolicyGroupInPolicyGroup*

La classe PolicyGroupInPolicyGroup permet de structurer entre eux les groupes de règles.

Elle surcharge les références de PolicyComponent tel que :



♦ *L'agrégation PolicyConditionInPolicyRule*

Une instance de la classe PolicyConditionInPolicyRule permet de lier une condition à une règle. Comme déjà dit plus haut, il est possible de lier plusieurs conditions à une règle et de lier une condition à plusieurs règles. Dans les deux cas, il suffit d'utiliser plusieurs instances de PolicyConditionInPolicyRule.

Si plusieurs conditions sont liées à une même règle, il faut pouvoir les structurer. La propriété ConditionListType de la classe PolicyRule indique si la liste de conditions est présentée sous forme normale conjonctive (CNF) ou sous forme normale disjonctive (DNF). Il est donc nécessaire de disposer d'un mécanisme de parenthèsage et d'un moyen de représenter la négation. La classe PolicyConditionInPolicyRule possède à cet effet les deux propriétés GroupNumber (pour le parenthèsage) et ConditionNegated (pour la négation).

Voici un exemple pour en comprendre le fonctionnement. Soit une règle liée à cinq conditions C_1 à C_5 par l'intermédiaire de cinq instances de PolicyConditionInPolicyRule telles que:

$C_1 \leftrightarrow$ GroupNumber = 1, ConditionNegated = FALSE
$C_2 \leftrightarrow$ GroupNumber = 1, ConditionNegated = TRUE
$C_3 \leftrightarrow$ GroupNumber = 1, ConditionNegated = FALSE
$C_4 \leftrightarrow$ GroupNumber = 2, ConditionNegated = FALSE
$C_5 \leftrightarrow$ GroupNumber = 2, ConditionNegated = FALSE

Donc si ConditionListType = DNF, alors la condition globale vaut :

$(C_1$ AND (NOT $C_2)$ AND $C_3)$ OR $(C_4$ AND $C_5)$

Et si ConditionListType = CNF, alors la condition globale vaut :

$(C_1$ OR (NOT $C_2)$ OR $C_3)$ AND $(C_4$ OR $C_5)$

♦ *L'agrégation PolicyRuleValidityPeriod*

Les concepteurs de PCIM ont choisi de ne pas mélanger les conditions temporelles (de type PolicyTimePeriodCondition) avec les autres conditions. A cause de cela, pour lier une condition temporelle à une règle, il faut utiliser une relation de type PolicyRuleValidityPeriod, à la place d'une relation de type PolicyConditionInPolicyRule.

Si plusieurs conditions temporelles sont liées à une même règle, alors celle-ci est active si au moins une de ces conditions est valide.

♦ *L'agrégation PolicyActionInPolicyRule*

Une instance de la classe PolicyActionInPolicyRule permet d'associer une action à une règle. Comme déjà dit plus haut, il est possible de lier plusieurs actions à une règle et de lier une action à plusieurs règles. Dans les deux cas, il suffit d'utiliser plusieurs instances de PolicyActionInPolicyRule.

Si plusieurs actions sont liées à une même règle, il peut être nécessaire de les ordonner (voir la propriété SequencedActions de la classe PolicyRule page 47). La classe PolicyActionInPolicyRule utilise la propriété ActionOrder pour cela. Cette propriété contient un entier naturel qui indique la position relative d'une action dans la séquence d'actions associées à la règle. La valeur 0 signifie « n'importe où ». Les valeurs supérieures indiquent la place d'une action telle que les actions ayant les plus petites valeurs sont exécutées en premier. Si plusieurs actions ont la même valeur, alors leur position relative n'a pas d'importance.

♦ *L'agrégation PolicyRepositoryInPolicyRepository*

La classe PolicyRepositoryInPolicyRepository est la seule agrégation à ne pas descendre de la classe PolicyComponent. En effet, elle ne traite pas directement de politiques. Son but est de permettre l'imbrication de Policy Repositories dans d'autres Policy Repositories afin de les structurer.

Elle définit deux références (ces références ne sont pas héritées de PolicyComponent !) :

- La référence GroupComponent désigne le Policy Repository P_0 contenant le Policy Repository P_1.
- La référence PartComponent désigne le Policy Repository P_1 contenu par le Policy Repository P_0.

4.1.3.2.2 Les associations

♦ *L'association abstraite PolicyInSystem*

La classe PolicyInSystem est racine de toutes les associations entre les politiques et le système qui les abrite. Etant abstraite, elle n'a pas pour objectif d'être instanciée directement mais permet d'établir une base commune pour toutes ses sous-classes et notamment d'imposer une sémantique commune pour leurs références.

Les références Antecedent et Dependent sont héritées de la classe CIM Dependency. La classe PolicyInSystem les surcharge pour les faire pointer respectivement vers des objets de type System et de type Policy. Figure 26 indique comment les sous-classes de PolicyInSystem surchargent de nouveau ses références selon leurs besoins.

♦ *L'association PolicyGroupInSystem*

Cette association lie un groupe de politiques à un système afin de préciser son domaine d'application.

♦ *L'association PolicyRuleInSystem*

Sans regarder si elle est contenue dans un (ou plusieurs) groupe(s) de politiques, une règle doit être définie, elle-même, par rapport au domaine d'application d'un système. Cette association lie cette règle à ce système.

♦ *L'association PolicyConditionInPolicyRepository*

Une condition qui est utilisée dans plusieurs règles doit toujours être liée à un unique Policy Repository. Cette liaison est représentée par l'association PolicyConditionInPolicyRepository.

Il faut noter qu'une condition dédiée à une règle particulière ne doit être liée à aucun Policy Repository.

♦ *L'association PolicyActionInPolicyRepository*

Une action qui est utilisée dans plusieurs règles doit toujours être liée à un unique Policy Repository. Cette liaison est représentée par l'association PolicyActionInPolicyRepository.

Il faut noter qu'une action dédiée à une règle particulière ne doit être liée à aucun Policy Repository.

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 ne sera pas traduit ici pour éviter toute confusion. En effet, les bases de données relationnelles sont un exemple de Policy Repository. Les annuaires (Directories) en sont un autre exemple. De plus, le Policy Repository est une entité logique ; rien n'empêche, et c'est même probable, qu'il corresponde à plusieurs entités physiques.

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' est compliquée par d'autres contraintes.

♦ *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, de rôles 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. Il est en effet assez évident que la localisation des données peut avoir un impact significatif sur les performances du système.

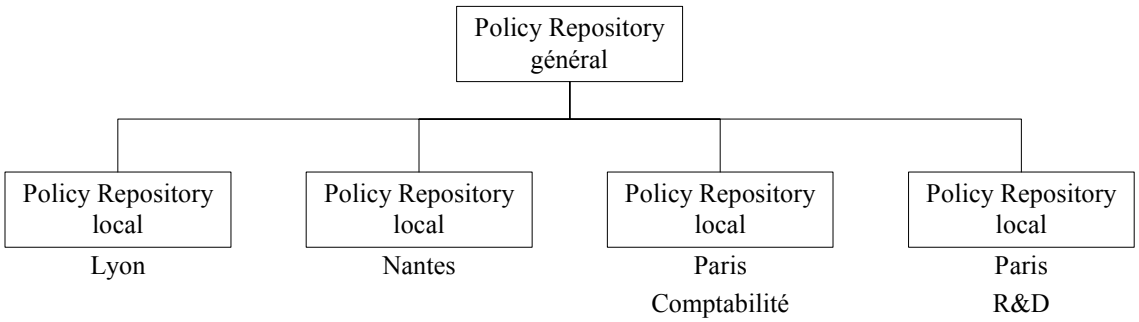


Figure 27 : Exemple de topologie d'un Policy Repository distribué

♦ *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). 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.

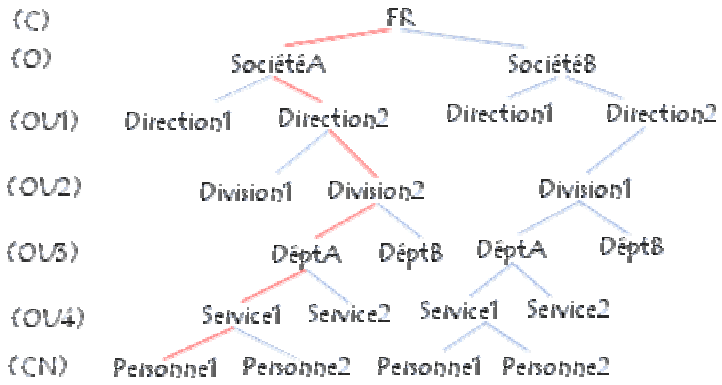


Figure 28 : Exemple de DIT

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 29 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.

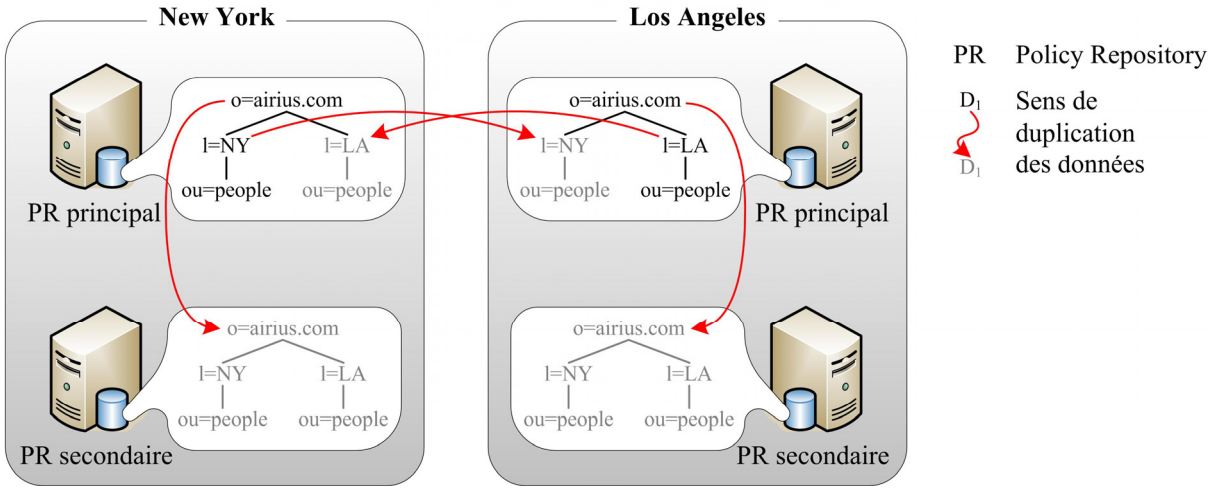


Figure 29 : Exemple d'utilisation du Replication Service de LDAP

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.

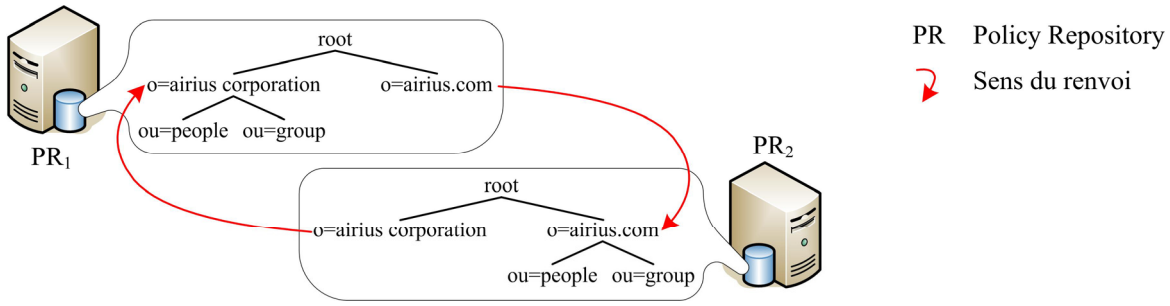


Figure 30 : Exemple d'utilisation du Referral Service de LDAP

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 généralement utiliser des bases de données car elles sont plus simples à mettre en œuvre et qu'il est encore rarement nécessaire de répartir le Policy Repository sur plusieurs serveurs.

CONCLUSION

La gestion par politique est un grand progrès pour les administrateurs de réseau. Jusqu'à une époque récente, les éléments du réseau étaient configurés à la main, en utilisant des lignes de commandes émises depuis une console opérateur. L'idée du contrôle par politique est d'automatiser ce processus depuis la demande de l'utilisateur jusqu'à la configuration ou au moins de masquer les particularités des matériels pour permettre à l'administrateur de se concentrer sur les applications et les utilisateurs.

Ainsi bien qu'initialement conçue pour la QoS, la gestion par politiques est également adaptable dans d'autres domaines, notamment la sécurité. Dans le cadre du projet RHODOS (Réseau Hybride Ouvert pour le DéplOiement de Services mobile) dans lequel s'inscrit mon stage au sein de Thales, elle servira à mettre en place un service d'adaptation de flux vidéo.

Il est cependant nécessaire de modérer ce bilan très positif. En effet, cette solution de gestion par un serveur centralisé se heurte à l'opposition des partisans (par exemple les militaires) pour un Internet totalement décentralisé dans lequel aucune ressource capitale n'est centralisée. Ainsi bien que plusieurs points aient encore besoin d'évoluer, la recherche sur les PBN est en perte de vitesse.

Un autre élément important à souligner est l'avenir assez sombre de COPS. Bien que poussé par l'IETF et assez prometteur, il semble devoir être remplacé dans les années à venir. Netconf est apparu récemment et paraît être un candidat sérieux pour prendre sa succession. Cependant, même si la mort de COPS a été annoncée par certains depuis déjà plus d'un an, COPS continue d'être utilisé et c'est ce protocole de communication entre PDP et PEP qui sera employé dans RHODOS.

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

Les PBNs recouvrent de nombreux domaines de recherche (COPS, PCIM, PIB…). Chacun de ces domaines a une terminologie propre. Certains termes existent dans plusieurs terminologies (et ont le même sens) mais d'autres ne sont valables que dans un domaine particulier. Dans ce cas, ce domaine est indiquer entre parenthèses dans ce glossaire afin d'éviter certaines confusions (voir par exemple Policy Repository).

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
- HMAC — Keyed-Hashing for Message Authentication
- 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
- WG — Working Group

C

E

H

M

O

P

R

S

Annexe A

EXEMPLE D'UTILISATION

DES OBJETS CLIENTSI

Pour illustrer l'utilisation des objets ClientSI, plaçons nous dans le cadre de COPS-PR. Sans entrer dans les détails, COPS-PR définit six objets supplémentaires par rapport à COPS :

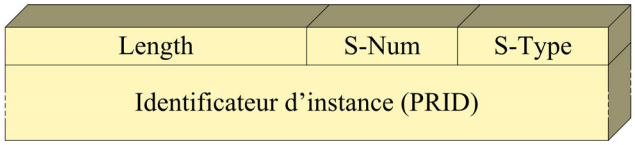
- Complete Provisioning Instance Identifier (PRID)
- Prefix PRID (PPRID)
- Encoded Provisioning Instance Data (EPD)
- Global Provisioning Error (GPERR)
- PRC Class Provisioning Error (CPERR)
- Error PRID (ErrorPRID)

En fait, il ne s'agit pas réellement de nouveaux objets car cela voudrait dire que COPS a été modifié de manière assez profonde ce qui n'est pas désirable. Plutôt que de parler d'objets, il serait en réalité plus judicieux de parler de sous-objets car les objets COPS-PR sont tous encapsulés soit dans des objets COPS Decicion, soit dans des objets COPS ClientSI selon le type du message dans lequel ils sont contenus. Lorsqu'il s'agit d'un message REQ ou RPT, les objets COPS-PR sont contenus dans un objet COPS ClientSI.

Afin de bien comprendre cet exemple, voyons un peu plus en détail les objets COPS-PR PRID et EPD.

♦ *Objet PRID*

L'objet PRID sert à transporter un PRID (Provisioning Instance Identifier) permettant de référencer une entrée dans la PIB (pour plus de détails voir [19] et [17]). Son format est le suivant :



Le champ Length est défini exactement comme le champ Length d'un objet COPS (voir page 24).

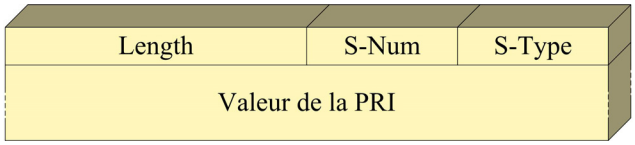
Le champ S-Num identifie le type d'objet COPS-PR et vaut ici PRID.

Le champ S-Type indique le type d'encodage est utilisé. Il s'agit par défaut de BER.

Le corps de l'objet contient le PRID, encodé selon S-Type. Ce corps de message doit avoir une longueur de *k*x32 bits et il peut être nécessaire de rajouter des bits de bourrage à la fin.

♦ *Objet EPD*

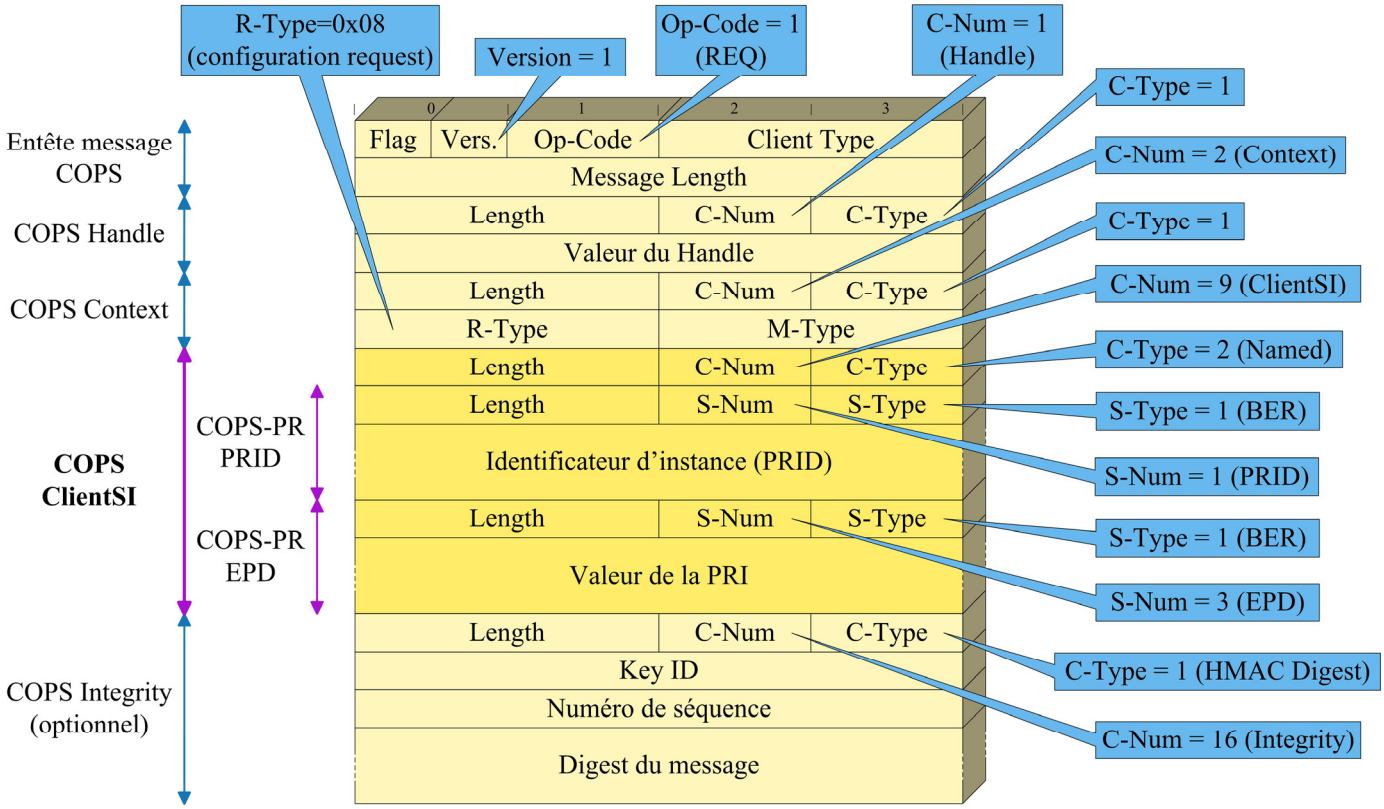
L'objet EPD sert à transporter la valeur d'une PRI (PRovisioning Instance) c'est à dire d'une entrée de la PIB (identifiée par un PRID). Son format est le suivant :



Les champs de l'entête sont identiques à ceux de l'objet PRID.

Le corps contient la valeur de la PRI. Il peut être nécessaire de rajouter des bits de bourrage pour que l'objet ait une taille de *k*x32 bits.

Voyons maintenant comment ces deux objets sont utilisés dans un message REQ et surtout comment ils sont intégrés dans un objet COPS ClientSI.



Sur la figure ci-dessus, on voit clairement que les objets COPS-PR PRID et EPD sont transportés dans le corps de l'objet COPS ClientSI (qui doit être de type "Named" pour cela).