Gestion commune des politiques et de l'administration des réseaux

Guillaume Doyen

MINISTERE DE L'EDUCATION NATIONALE DE LA RECHERCHE ET DE LA TECHNOLOGIE

*****

INSTITUT NATIONAL DES SCIENCES APPLIQUEES

Département de Génie Electrique & Informatique

*****

DIPLOME D'ETUDES APPROFONDIES

GESTION COMMUNE DES POLITIQUES ET DE L'ADMNISTRATION DES RESEAUX

INRIA Lorraine
Technopôle de Nancy-Brabois – Campus Scientifique
615, rue du Jardin Botanique - B.P. 101
54600 Villers Lès Nancy Cedex, France

DOYEN Guillaume
Automatique, Electronique & Informatique
Réseaux & Télécommunications

Septembre 2002

Remerciements

Tout d'abord, je tiens à remercier Madame le Professeur Hélène Kirchner, Directrice du LORIA, et Monsieur le Professeur André Schaff, Directeur de l'équipe RESEDAS, d'avoir accepté de m'accueillir au sein de leur laboratoire.

Je remercie vivement Olivier Festor. J'ai sincèrement apprécié sa disponibilité et sa réceptivité alors que je cherchais mon stage de DEA.

Tous mes plus vifs remerciements vont à mon tuteur de stage, Emmanuel Nataf, pour son accueil, sa disponibilité et ses conseils avisés. Sa politique de gestion et ses qualités humaines ont largement contribué au bon déroulement de ma première expérience dans le milieu de la recherche.

Enfin, merci à Vincent Nicomette pour ses conseils, son écoute et leur caractère non-institutionnel !

Table des matières

1 Introduction **9**

2 Le LORIA **11**
2.1 Organisation du LORIA 11
2.2 Ressources et partenariats 11
2.2.1 Etablissements associés 11
2.2.2 Personnels 12
2.3 Les thématiques de recherche 12
2.4 Valorisation et transfert 12
2.5 L'équipe RESEDAS 13
2.5.1 Principaux axes de recherche 13
2.5.2 Relations scientifiques et industrielles 13

I Etat de l'art **15**

3 La gestion par politiques **17**
3.1 Introduction 17
3.1.1 Les types de politiques 18
3.2 architecture générale 18
3.2.1 Le niveau *business* 19
3.2.2 Le niveau technologique 20
3.3 La notion de rôles 22
3.4 Synthèse 23

4 Les modèles d'implémentation **25**
4.1 Le protocole COPS 25
4.1.1 Un scénario classique 26
4.1.2 Le PDU COPS 27
4.1.3 Les objets COPS 27
4.2 Le modèle de délégation 28
4.2.1 Modèle de fonctionnement 28
4.2.2 Exemple de politique de délégation : la gestion de RSVP 29
4.3 Le modèle de provision 30
4.3.1 Modèle fonctionnel 30
4.3.2 Le modèle de l'information : SPPI 31
4.3.3 La PIB 32
4.4 Les méta-politiques 33

4.4.1 Le concept de méta-politiques . . . 33
4.4.2 La PIB de méta-politiques . . . 33
4.5 Synthèse . . . 34

5 SMI Next Generation 35
5.1 Présentation générale . . . 35
5.2 Eléments de langage . . . 36
5.2.1 Les types de base et dérivés . . . 36
5.2.2 Définition de classe . . . 36
5.3 Traduction de SMIng vers SNMP et COPS-PR . . . 37
5.3.1 Définition du *mapping* de SMIng vers COPS-PR . . . 37
5.4 Synthèse . . . 39

II Travail de développement 41

6 Développement d'un agent de gestion commune 43
6.1 Architecture générale . . . 44
6.1.1 Approche fonctionnelle . . . 44
6.1.2 Approche opérationelle . . . 45
6.2 Implémentation . . . 45
6.2.1 Le *parser* . . . 45
6.2.2 L'agent *toolkit* . . . 47
6.3 Etat actuel et Développement futur . . . 49

7 Gestion par politique de l'environnement FLAME 51
7.1 FLAME . . . 51
7.2 Intégration d'une gestion par politique . . . 52
7.2.1 Gestion politique des opérations de l'EE . . . 52
7.2.2 La PIB FLAME . . . 53
7.3 Implantation du PDP FLAME . . . 56
7.3.1 Le noyau du PDP FLAME . . . 56
7.3.2 La couche de communication . . . 56
7.3.3 L'interface graphique . . . 57
7.4 Etat actuel et Développement futur . . . 57

III Conclusion 59

IV Annexes 63
7.5 Modélisation de la gestion de réseaux . . . 65
7.6 Le modèle de l'information : SMI . . . 66
7.7 La MIB . . . 67
7.7.1 La MIB II . . . 68
7.8 Le protocole SNMP . . . 68
7.8.1 Les différents services . . . 69

Table des figures

3.1 Exemple de réseau d'entreprise. 18
3.2 Architecture générale d'une gestion par politiques. 19
3.3 Exemple de politique de niveau *business* selon le formalisme PFDL. 20
3.4 Le modèle de classes PCIM. 21
3.5 L'instanciation d'une politique de niveau business. 22
3.6 Exemple de rôles. 23

4.1 Un exemple de scénario. 26
4.2 Le PDU COPS. 27
4.3 Le Format d'un objet COPS. 28
4.4 Le modèle de délégation. 28
4.5 Architecture d'un noeud avec services intégrés. 29
4.6 Le modèle RSVP de réservation. 29
4.7 Le contrôle d'admission. 30
4.8 Le modèle de provision. 31
4.9 Architecture d'un noeud avec services intégrés. 34

6.1 La vision unique de la gestion de réseau. 43
6.2 Architecure générale de l'agent *toolkit*. 44
6.3 Les différentes opérations. 45
6.4 Exemple d'arbre abstrait généré par le mapping de SMIng vers COPS-PR. 47
6.5 Schema d'implantation du processus de génération 48
6.6 Le service GET COPS-PR. 49

7.1 L'architecture FLAME. 52
7.2 Architecture de la PIB FLAME. 54
7.3 Exemple de traitement de requête FLAME. 55
7.4 L'architecture du PDP FLAME. 56
7.5 La fenêtre principale de l'interface graphique FLAME. 57
7.6 Exemple d'une architecture SNMP. 66
7.7 L'arbre d'enregistrement ISO. 67
7.8 Les principales branches de la MIB II. 68
7.9 Le PDU SNMPv1. 69

Chapitre 1

Introduction

Une des parts majeure de l'évolution de l'Internet semble tournée vers la qualité de service (QoS). Les réseaux sont toujours plus demandeurs en bande passante, et les applications multimédias se multiplient, de manière locale à chaque machine, mais aussi de manière distribuée. Ainsi, de nouveaux protocoles gérant la QoS voient le jour ; protocoles applicatifs de signalisation, protocoles de transport, et extensions du protocole IP permettent de garantir la QoS nécessitée par des applications toujours plus friantes de bande passante et des usagers toujours plus exigents.

Le trafic généré sur les réseaux n'étant plus équitable, il parait indispensable d'en contrôler son accès. En effet, l'utilisation de ressources du réseau ne doit pas s'effectuer de manière anarchique. C'est dans le but de résoudre ce problème que sont apparues les premiers éléments de gestion de réseaux par politique. Initialement, cette nouvelle approche avait pour but de contrôler l'accès mais aussi la QoS requis par des usagers en fonction de contrats établis au préalable.

Aujourd'hui, la gestion de réseaux par politiques s'applique à l'ensemble des éléments (services, usagers, équipements) administrables et concerne de nombreux types. Elle consiste à définir un ensemble de règles qui vont régir ces différents éléments. Il existe par exemple des politiques inhérentes à la sécurité du trafic comme d'autres permettant de gérer l'occurrence d'erreurs au sein d'un équipement. Néanmoins, ce nouveau domaine n'en n'est qu'à ses débuts. Les architectures déployées pour mettre en oeuvre cette approche de gestion sont variées, les protocoles utilisés, nombreux, et il existe beaucoup d'architectures propriétaires.

De plus, l'approche classique d'administration réseau fondée sur SNMP semble concurrente à ce nouveau système de gestion. En effet, l'IETF propose un modèle de gestion par politique, et un modèle d'administration réseau semblables en de nombreux points : modèles de l'information, protocoles et architectures reposent sur les mêmes principes.

Pour répondre à cette multitude d'approches, l'équipe RESEDAS du LORIA propose de développer une architecture de gestion de réseau qui soit indépendante de son implémentation. Celle-ci permet de gérer de manière commune les différentes approches citées précédemment et apporte ainsi une vision unique de la gestion de réseaux. Dans ce projet, mon travail consiste à intégrer l'approche de gestion de réseaux par politiques proposée par l'IETF à cette architecture générique.

Le plan de cette étude sera la suivant. Dans un premier temps, nous effectuerons l'état de l'art du domaine de la gestion de réseaux. Nous étudierons tout d'abord l'approche d'administra-

tion classique, fondée sur SNMP. Ensuite nous détaillerons l'approche de gestion de réseau par politiques. Enfin, nous nous intéresserons à SMIng, un langage qui propose une manière unique de décrire l'information utile à la gestion de réseaux, quelle que soit son implémentation. Dans un second temps, nous présenterons deux travaux développés au sein de l'équipe RESEDAS aux quels nous contribuons. Le premier concerne la réalisation d'une interface de gestion de réseaux commune au domaine de l'administration et la gestion par politiques, fondée sur SMIng. Le second est une application de cette interface : l'intégration d'une architecture de gestion dans l'environnement FLAME, environnement de réseau actif dédié à la supervision des réseaux IP.

Chapitre 2

Le LORIA

Le Laboratoire Lorrain de Recherche en Informatique et ses Applications (LORIA) est une unité mixte de recherche (UMR 7503) commune au CNRS (Centre National de la Recherche Scientifique), à l'INRIA (Institut National de Recherche en Informatique et en Automatique), à l'INPL (Institut National Polytechnique de Lorraine), et aux universités Henri Poincaré, Nancy 1 et Nancy 2. Cette unité, dont la création a été officialisée le 19 décembre 1997 par la signature du contrat quadriennal avec le Ministère de l'Éducation Nationale, de la Recherche et de la Technologie et par une convention entre les cinq partenaires, succède ainsi au Centre de Recherche en Informatique de Nancy (CRIN), et aux équipes communes entre celui-ci et l'Unité de Recherche INRIA de Lorraine.

2.1 Organisation du LORIA

Depuis le 1er janvier 2001, Hélène KIRCHNER dirige le LORIA. Actuellement plus de trois cents personnes travaillent dans le laboratoire. Ces personnels sont répartis en 21 équipes de recherche et 7 services d'aide à la recherche. Chaque équipe rassemble des chercheurs, des doctorants et des assistants techniques ou administratifs, pour la réalisation d'un projet de recherche. Cinq instances ont été mises en place pour assister le directeur du laboratoire, garantir la cohérence de la politique scientifique et le bon fonctionnement au quotidien :

- L'équipe de direction : composée de plusieurs membres du laboratoire, elle assiste la Directrice dans ses fonctions.
- Le comité de gestion : composé des chefs de service, il assiste la directrice dans le fonctionnement journalier du LORIA.
- Le comité des projets : il conseille le Directeur sur la politique scientifique du LORIA, participe à l'évaluation des projets et équipes, et instruit les restructurations nécessaires.
- Le conseil du LORIA : il émet des avis sur la politique scientifique mise en oeuvre par le Comité des Projets. Sa composition est fixée par les statuts d'UMR.
- Le conseil des orientations scientifiques : composé de représentants des équipes de recherche, il conseille la Direction dans la gestion scientifique du laboratoire.

2.2 Ressources et partenariats

2.2.1 Etablissements associés

Le LORIA associe les personnels et moyens des cinq établissements suivants :

- CNRS : Centre National de la Recherche Scientifique
- INPL : Institut National Polytechnique de Lorraine
- INRIA :Institut National de Recherche en Informatique et en Automatique
- UHP : Université Henri Poincaré, Nancy 1
- Nancy 2 : Université Nancy 2

2.2.2 Personnels

LORIA regroupe plus de 300 personnes dont :
- 109 chercheurs permanents,
- 94 thésards,
- 61 ITA,
- 53 Post-Doc, ingénieurs experts et membres associés.

2.3 Les thématiques de recherche

Dans le secteur des sciences et technologies de l'information et de la communication, le LORIA possède, à travers dix neuf équipes de recherche, cent cinquante chercheurs et une centaine de doctorants, des compétences reconnues dans des secteurs en pleine évolution et porteurs de développement économique potentiel. Les activités de ces équipes sont centrées autour de cinq thématiques principales transversales sur lesquelles elles développent des recherches fondamentales et appliquées. Bien entendu, des équipes ont des activités dans plusieurs de ces thématiques dont voici la liste :
- Calculs, réseaux et graphismes à hautes performances
- Télé-opérations et assistant intelligents
- Ingénierie des langues, du document et de l'information scientifique et technique
- Qualité et sûreté des logiciels et systèmes informatiques
- Bioinformatique et applications à la génomique

2.4 Valorisation et transfert

Le LORIA développe de nombreuses relations industrielles (plus de 200 contrats en cours fin 1999) et a une importante activité de diffusion de logiciels. Six logiciels et une marque ont été déposés en 1998. Fort de près de 90 enseignants-chercheurs, le LORIA participe activement à la formation universitaire dans les domaines des sciences et nouvelles technologies. Conscient de la nécessité d'être un élément moteur du contexte socio-économique régional, le LORIA a décidé de créer le LoriaTech en janvier 1999, club des partenaires du LORIA, qui a pour objectifs :
- de donner accès plus rapidement aux informations sur les évolutions de la recherche et du secteur, c'est-à-dire améliorer la veille technologique des entreprises. En particulier, les membres du LoriaTech ont un accès privilégié au centre de documentation du LORIA et de l'INRIA-Lorraine.
- de monter des partenariats divers, citons en particulier :
  - dans le cadre des établissements universitaires (ESIAL, ESSTIN, ISIAL, écoles d'ingénieurs de l'INPL, IUT Informatique de Nancy2, IUP-Miage de Nancy), la possibilité de stages de projets industriels.
  - et dans le contexte du LORIA, les sujets DRT (Diplôme de Recherche Technologique), des conventions CIFRE.
- ou encore d'offrir la possibilité de partenariat LORIA-entreprises en réponse à des appels d'offre français ou européens.

Un Espace-transfert a été également créé fin 1999 à proximité du LORIA. Après 2 créations d'entreprises début 2000, 5 autres projets sont en cours de montage.

2.5 L'équipe RESEDAS

Concepts et Outils logiciels pour les télécommunications Et les systèmes distribués

- Responsable scientifique : André Schaff ; Tél : 03 83 59 20 11 ; Andre.Schaff@loria.fr
- Responsable permanent : Olivier Festor ; Tél. : 03 83 59 20 16 ; Olivier.Festor@loria.fr

La réalisation d'applications informatiques dépend de plus en plus de l'utilisation de réseaux. Cette nouvelle dimension ajoute un niveau de complexité à la maîtrise des logiciels de (télé-) communications. Le projet RESEDAS, commun à l'INRIA et au LORIA, a pour cadre général l'étude de solutions innovantes la conception d'outils logiciels expérimentaux pour faciliter le développement, le déploiement et l'exploitation de services, protocoles et applications distribuées sur des réseaux de télécommunications et des réseaux locaux

2.5.1 Principaux axes de recherche

Dans ce cadre, le groupe développe des activités sur les trois thèmes suivants :

- Gestion des réseaux et des services (application au Réseau de Gestion des Télécommunications),
- Calcul répartion sur des réseaux de stations hétérogènes,
- Protocole IP nouvelle génération (Ipv6), expérimentation, validation de plate-forme et conception de nouvelles classes d'applications.

Deux prototypes logiciels sont diffusés sur WWW :

- MODERES est un environnement logiciel dédié au développement et à la manipulation de modèles de l'information issus de différentes approches de gestion (http//www.loria.fr/-exterieur/-equipe/resedas/mode.html).
- Para++ est une bibliothèque C++ dont l'objectif est de faciliter l'accès aux bibliothèques traditionnelles de communication par passages de messages (http//www.loria.fr/para++).

2.5.2 Relations scientifiques et industrielles

Depuis septembre 1996, nous travaillons dans le domaine de la gestion des réseaux et services avec le département réseaux de l'ENSIAS à Rabat au Maroc.

Depuis plusieurs années, nous maintenons des contacts suivis avec le NIST (National Institute of Standards and Technology, Maryland, USA) dans le domaine du calcul distribué sur réseaux de station hétérogènes et sur l'utilisation des agents mobiles pour la supervision d'applications multimédias.

Nous maintenons de nombreux autres contacts avec des laboratoires et universités américaines ou canadiennes (Delaware, Montréal, U. du Québec à Montréal, CRIM, INRS Télécom).

Nous accueillons régulièrement des chercheurs étrangers pendant des périodes de 2 semaines à 1 an.

Nos contacts industriels sont BULL, Netmansys, NIST, TDF-C2R, Boeing, CS-Télécom.

Première partie

Etat de l'art

Chapitre 3

La gestion par politiques

L'objectif de ce chapitre est d'introduire les différents concepts définis dans le cadre d'une gestion de réseau par politiques. Dans un premier temps, nous présenterons l'architecture générale d'une telle approche, ses éléments, leur interaction ainsi qu'un modèle de formalisme de politiques. Dans un second temps, nous étudierons la notion de rôle qui permet d'associer une règle politique à ses éléments.

3.1 Introduction

La gestion de réseaux est une tâche de plus en plus complexe. D'un côté, les équipements se diversifient, d'un autre l'échelle des réseaux croît sans cesse. L'approche SNMP [1] est efficace pour modéliser les ressources et avoir une vision de leur état, mais elle ne permet pas de gérer leur comportement. C'est là qu'est l'intérêt d'une gestion par politique : munir les ressources de comportements fondés sur la définition d'une politique de gestion [**?**].

Plus formellement, une politique de gestion est un ensemble de règles qui permet de fixer le comportement des éléments qu'elle administre. Par élément, on entend service, usager ou équipement.

Afin d'illustrer cette première définition, considérons l'exemple de réseau local d'entreprise représenté sur la figure **??**. Le réseau est relié à l'Internet, comporte un routeur d'accès ainsi que deux sous-réseaux. L'un est utilisé par des employés administratifs, l'autre par des ingénieurs. D'un point de vue fonctionnel, une gestion par politiques va permettre de répondre au cahier des charges suivant :

- Différenciation du traitement de trafic : on désire munir les usagers de classes de services différentes (au sens bande passante et délai de transit) en fonction de leurs besoins. Ainsi, les ingénieurs devront bénéficier d'un service *Premium* (QoS garantie) et les employés administratifs d'une gestion standard de type *Best Effort*.
- Définition de contraintes horaires : on veut autoriser l'accès des usagers à l'Internet seulement durant les heures de travail, c'est à dire de 8h00 à 18h00.
- Sécurisation du trafic : les ingénieurs ont besoin de communiquer avec ceux d'un sîte distant. On veut assurer la confidentialité des données échangées en créant un canal privé virtuel.
- Définition de comportements critiques : en cas de congestion du routeur d'accès, on veut pouvoir reclasser le trafic généré par les deux sous-réseaux de la manière suivante : Le trafic ingénieur initialement muni d'une classe de service *Gold* sera traité d'une manière standard (*Best Effort*) ; le trafic généré par les employés administratifs sera supprimé.

[1] Un chapitre de présentation de l'approche SNMP est donné en annexes

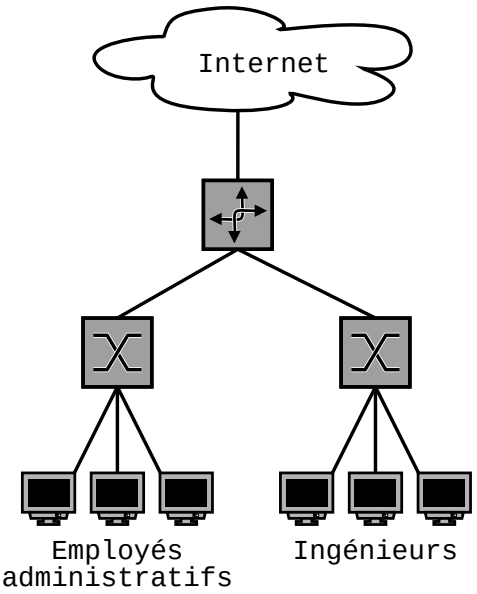


FIG. 3.1 – Exemple de réseau d'entreprise.

La mise en oeuvre des besoins cités ci-dessus va s'effectuer sous la forme de règles qui vont régir le comportement, les droits et les interdictions des services, équipements et usagers inscrits dans le cadre de la gestion du réseau.

3.1.1 Les types de politiques

D'après l'exemple précédent, on entrevoit l'étendue du domaine d'application des politiques. Afin d'ordonner les différents types de politiques, l'IETF propose la classification fonctionnelle de politiques suivantes [**?**] :

- **Politiques de motivation** : concernent le fait et la manière d'atteindre un objectif politique. Les politiques de configuration et d'utilisation sont des types de politiques de motivation. Une politique de configuration définit la configuration par défaut d'un élément, alors que la politique d'utilisation définit la configuration d'un élément basée sur des données d'utilisation.
- **Politiques d'installation** : définissent ce qui peut être installé ou non sur un équipement.
- **Politiques d'erreurs et évènementielles** : définissent les actions à effectuer lorsqu'un élément ou réseau disfonctionne.
- **Politiques de sécurité** : concernent les mécanismes à appliquer aux clients, concernant l'autorisation, l'authentification, et la facturation de l'accès aux ressources.
- **Politiques de service** : caractérisent le réseau et les services disponibles.

3.2 architecture générale

D'une manière très générale, on a défini quatre points comme étant les fondements d'une architecture de gestion par politiques. Ainsi, toute mise en œuvre de cette approche doit définir et implanter [**?**] :

- **Un modèle d'information** pour les élements du réseau, les services, les réseaux, et les clients du réseau.
- **Un langage de spécification** de politiques qui puisse représenter les besoins d'une manière indépendante de leur implémentation.
- **Une structure** supportant le facteur d'échelle d'administration, de gestion, de distribution et de vérification de politiques.
- **Un moyen de traduire** les spécifications de politiques en commandes de configuration compréhensibles par un équipement.

L'IETF propose une architecture générale d'implantation [?] de gestion de réseau par politique, en accord avec les quatre besoins fonctionnels définis précédemment, qui se scinde en deux niveaux (figure ??) :

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

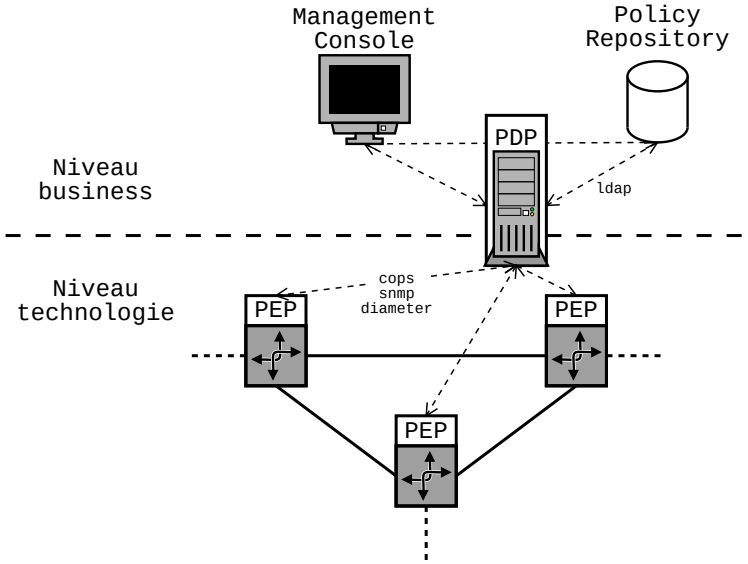


FIG. 3.2 – Architecture générale d'une gestion par politiques.

3.2.1 Le niveau *business*

Le niveau *business* permet la gestion de politiques compréhensibles par un administrateur. A ce niveau, les politiques sont représentées par des règles du type `Condition->Action`. Différents langages existent pour définir des régles (PCIM [?], Ponder [?], PFDL [?], ...) qui peuvent être stockées dans une base de données logique.

Considérons, par exemple le besoin de contraintes temporelles exprimé précédemment : dans un réseau local on veut autoriser l'accès à l'Internet aux usagers seulement durant leurs heures de travail. On va alors exprimer ce besoin sous forme d'une règle de la forme : "SI l'heure est

comprise entre 8h et 18h ET les adresses sont de destination *.*.*.* ALORS autoriser l'accès". La figure **??** montre l'expression de cette regle selon le formalisme PFDL.

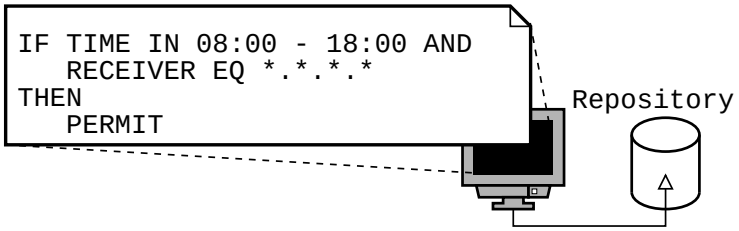


FIG. 3.3 – Exemple de politique de niveau *business* selon le formalisme PFDL.

Les différents éléments

Le niveau *business* comporte les trois éléments suivants :
- La console : Elle est l'interface par laquelle un opérateur peut gérer les politiques à mettre en oeuvre.
- La base de données : communémment appelée *repository*, elle permet de stocker l'ensemble des politiques inhérentes à un domaine. Différents protocoles sont utilisables pour accéder aux informations (LDAP, SQL, ...).
- Le PDP (*Policy decision point*) : d'un point de vue *business*, son rôle est de lire les politiques enregistrées dans le *repository* et de les traiter.

Le modèle PCIM

Le modèle PCIM (*Policy Core Information Model*), développé et standardisé par l'IETF [**?**], permet de modéliser les politiques à déployer dans une gestion de réseau. Il est basé sur le modèle CIM (*Common Information Model*), développé par le DMTF (*Distribued Management Task Force*). Dans la figure **??**, PCIM propose une modélisation orientée objet avec une représentation de type UML. Il définit deux types de classes d'objets : les classes structurelles, représentant des informations politiques et leur contrôle, et les classes d'associations indiquant comment les instances de classes structurelles sont reliées entres-elles (liens (a)...(j)).

Chaque classe de règle politique (*Policyrule*) est constituée de conditions (*PolicyCondition* et *PolicyTimePeriodCondition*) et d'actions (*PolicyAction*). L'héritage utilisé pour les *PolicyCondition* ouvre la possibilité de créer de nouvelles conditions sans redéfinir un comportement spécifique de type `Condition->Action`. La classe *PolicyRepository* permet de stocker les conditions et actions qui seront réutilisées par la suite pour définir une nouvelle règle. Les règles peuvent être assemblées pour définir un groupe (*PolicyGroup*). Une démarche consiste à définir un groupe par catégorie de politique (Motivation, Installation, Erreur, Sécurité, Service). Enfin, la classe *System* contient l'ensemble des groupes définis.

La démarche proposée pour établir un système de gestion par politiques consiste à modéliser le réseau et son évolution comme une machine à états, dont chaque état représente l'instanciation de groupes de politiques, et chaque transition les règles permettant de les instancier.

3.2.2 Le niveau technologique

Le niveau technologique, constitué des éléments à administrer, a pour rôle d'appliquer les politiques définies au niveau *business*. La figure **??** illustre l'instanciation des règles. On a ici

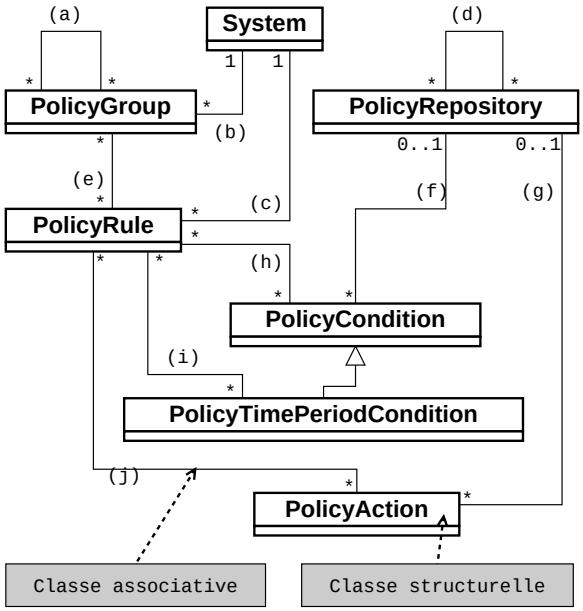


FIG. 3.4 – Le modèle de classes PCIM.

considéré la règle PFDL définie sur la figure **??** et sa traduction, selon le modèle de provision (voir plus loin) : le routeur de frontière du réseau local possède une table représentant les adresses IP de destination qui sont autorisées. A 8h00, le PDP y installe la valeur `*.*.*.*` qui autorise l'accès à Internet. En revanche, à 18h00, le PDP supprime cette valeur et bloque ainsi tout trafic vers Internet.

Les différents éléments

Le niveau technologique comporte les éléments architecturaux suivants :

- Le PEP (*Policy Enforcement Point*) : c'est le point d'application des politiques. Concrétement, un PEP est un processus s'éxecutant dans un routeur, un serveur, un pare-feu, ...Un PEP peut être régi par un ou plusieurs PDP ; multiplier les PDP est utile afin de fiabiliser l'architecture (par l'utilisation de PDP's de secours) ou de séparer les différents types de politiques (sécurité, facturation, contrôle d'admission, ...).
- Le PIN (*Policy Ignorant Node*) : actuellement, la plupart des équipements ne sont pas conçus pour supporter une gestion de réseau par politique. Néanmoins, il est possible d'implémenter entre un PDP et un PIN, un proxy ayant pour charge de convertir les instructions politiques en commandes SNMP ou CLI (*Command Line Interface*) propriétaires.
- Le PDP : d'un point de vue technologie, le PDP se charge de faire appliquer les politiques de gestion aux différents PEP qu'il administre.
- Le LPDP (*Local PDP*) : situé dans le même élément qu'un PEP, le LPDP sert à suppléer le PDP dans la prise de décisions. Il peut être utile dans plusieurs cas d'utilisation :
  - Si un PEP perd toute connexion avec son PDP et ceux de secours, alors le LPDP prendra le rôle de PDP. Lorque du retour de la connexion, il devra néanmoins rendre compte de toutes ses décisions au PDP initial.
  - Certains types de fonctionnement peuvent nécessiter, localement, une décision partielle, qui conditionnera la décision finale prise par le PDP.

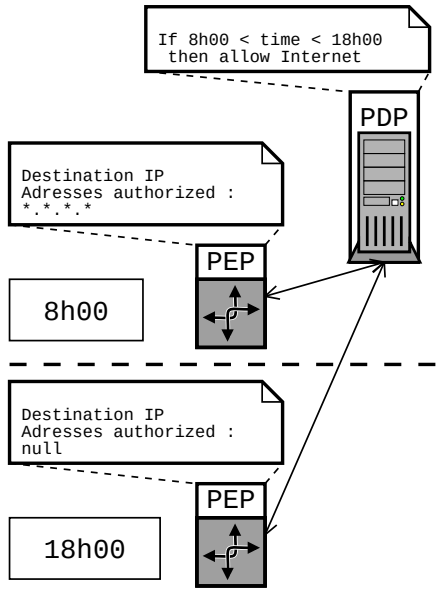


FIG. 3.5 – L'instanciation d'une politique de niveau business.

Remarque : élément central d'une gestion par politiques, le PDP est l'interface entre les niveaux *business* et technologique, d'où sa double définition. Il a pour tâche de vérifier la cohérence des politiques définies au niveau *business*, mais aussi de les traduire en données de configuration compréhensibles par les PEP qu'il administre.

3.3 La notion de rôles

Le niveau *business* permet de définir les politiques à mettre en oeuvre au sein d'un domaine. Le niveau technologique a pour rôle de les instancier en passant par un élément de traduction : le PDP. Ainsi, on pourrait croire que toutes les politiques de niveau *business* sont implantées dans chacun des éléments du réseau. Cette manière de fonctionner n'est pas viable, car si elle fonctionne, elle surcharge considérablement chaque équipement d'une manière inutile. En effet, les services déployés au sein d'un réseau ne sont pas implantés dans chaque équipement, de même qu'un usager n'a pas forcément accès à toutes les ressources. Il est donc intéressant de pouvoir distinguer les éléments soumis à des règles de politique. On a ainsi défini la notion de rôle [**?**]. Un rôle est un moyen de rassembler des éléments (usagers, ressources, services) sous un terme commun. Une règle politique ne s'applique alors qu'à un rôle ou un ensemble de rôles et pas directement à un élément. De plus, chaque élément peut supporter plusieurs rôles, que l'on appelle combinaison de rôles.

La figure **??** montre l'exemple de rôles qui peuvent être attribués à différents éléments d'un réseau local. On attribue ici un rôle pour représenter les interfaces de communication (type et situation). Ainsi, le routeur d'accès se verra attribuer la combinaison `Ethernet + WAN + Local` car il est de technologie Ethernet, présente une interface côté Réseau local, et une autre côté WAN. Par contre, les postes de travails auront la combinaison `Ethernet + Local`.

Etant donné cette attribution de rôles, chaque élément se verra appliquer les règles politiques inhérentes à sa combinaison, aux termes simples de sa combinaison, mais aussi à chaque sous-

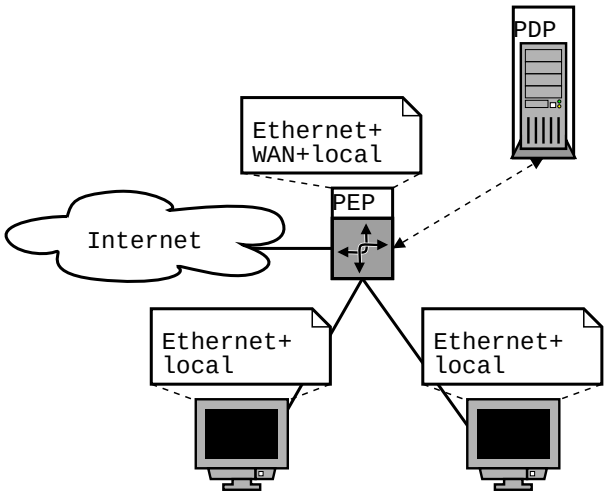


FIG. 3.6 – Exemple de rôles.

combinaison possible de la combinaison initiale. Le routeur d'accès appliquera ainsi toutes les règles définies pour les rôles suivants :

```
Ethernet  +  WAN  +  Local
Ethernet +   WAN
Ethernet +  Local
WAN + Local
Ethernet
WAN
Local
```

En conclusion, l'attribution de rôles aux éléments permet de simplifier la tâche d'administration en évitant la configuration un à un des éléments administrés et en supprimant la surcharge due à une attribution systématique de toutes les politiques à chacun des éléments régis.

3.4 Synthèse

La gestion de réseaux par politiques est un modèle qui permet de définir un ensemble de règles qui vont régir le comportement d'éléments administrés. Dans le cadre de l'approche de l'IETF, différents types de politiques existent permettant ainsi de les classifier par domaine fonctionnel (motivation, installation, erreur, sécurité et service).

L'architecture générale définie afin de mettre en oeuvre un tel type de gestion comporte deux niveaux :

- un niveau *business* qui permet la gestion de politiques d'une manière intelligible pour l'homme, selon un formalisme de spécification. Par exemple, PCIM est un modèle qui permet la spécification de politiques selon une architecture orientée objet.
- un niveau *technologique* qui vérifie, distribue et applique les politiques définies au niveau *business*. Ses principaux constituants sont le PDP, élément central qui effectue le travail de vérification et distribution des politiques, et le PEP, qui a pour rôle de faire appliquer les politiques au sein de l'élément dans lequel il est implanté.

Enfin, la notion de rôle permet de lier chaque règle définie au niveau *business* aux éléments auquels elle s'applique. En effet, une règle ne s'applique pas directement à un élément mais à un rôle, qui est lui même associé à un ou plusieurs éléments.

Chapitre 4

Les modèles d'implémentation

Le niveau technologique, constitué des éléments à administrer a pour rôle d'appliquer les politiques définies au niveau *business*. Dans le cadre de l'IETF, deux modèles d'implémentation sont mis en oeuvre pour assurer l'application des politiques de gestion de réseau :

- L'approche de délégation (*outsourcing*) : dans ce modèle, à chaque occurrence d'un évènement, le PEP demande au PDP la décision à prendre. L'approche d'*outsourcing* fut développée afin d'être appliquée à RSVP (*Ressource reSerVation Protocol*) [**?**].
- L'approche de provision (*provisioning*) : dans ce modèle, le PDP pourvoit ses PEP de règles qu'ils vont pouvoir appliquer localement. Cette approche initialement fut développée afin d'être appliquée au modèle de services différenciés (*diff-serv*).

Si les deux approches précédentes sont clairement définies, et vouées à des modèles de gestion de qualité de service (RSVP et *diff-serv*), elles sont suffisamment génériques pour être appliquées à l'ensemble des types de politiques définies au niveau *business*. D'ailleurs, au niveau technologique, on utilise le terme de "type de client" (*client-type*) pour désigner chaque instanciation de politique de type différent.

Ce chapitre a pour objectif de présenter les différents modèles d'implémentation du niveau technologique. Nous allons tout d'abord présenter COPS, le protocole de communication entre PEP et PDP. Ensuite, nous détaillerons les deux approches d'implémentation définies : *outsourcing* et *provisioning*. Enfin nous introduirons le concept de métapolitiques, concept qui assouplit le modèle de provisionning.

4.1 Le protocole COPS

Situé au dessus de TCP (port n°3288) dans la pile de protocoles, COPS (*Common Open Policy Service*) est le protocole standardisé par l'IETF [**?**] pour dialoguer entre un PDP et ses PEP. Néanmoins, d'autres protocoles peuvent être utilisés, comme SNMP [**?**] ou Diameter. La particularité de COPS est de n'être qu'une simple enveloppe pouvant véhiculer des informations quelle que soit l'approche de politique implémentée.

Le principe d'interaction défini par COPS est le suivant : Lorsqu'un un PEP reçoit un message qui requiert une décision politique, il formule une requête COPS (REQ) qu'il envoie au PDP. Ce dernier renvoie sa décision (DEC), au PEP qui l'applique.

Afin de sécuriser le transfert de données, COPS utilise un système d'authentification. Chaque couple (PEP, PDP) possède une série de clés partagées qui va être utilisée afin de créer une signature située à la fin de chaque message.

Enfin, COPS est un protocole à état. En effet, l'état des différents éléments est mémorisé.

Le PDP mémorise ainsi toutes les requêtes reçues, jusqu'à leur demande de suppression par le PEP inquisiteur. Les états ainsi installés sont dépendants, c'est-à-dire qu'une prise de décision est fonction de l'état des précédentes.

4.1.1 Un scénario classique

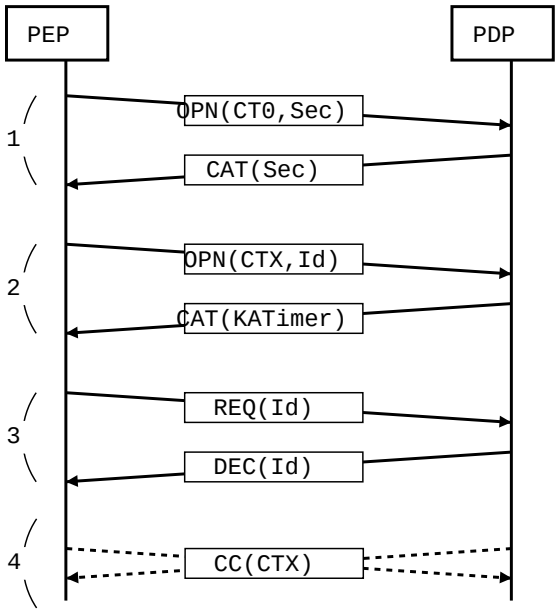


FIG. 4.1 – Un exemple de scénario.

L'exemple de scénario suivant est décrit sur la figure **??**. Il présente les échanges classiques qui sont effectués entre un PEP et un PDP. Concernant la représentation des messages, seuls les arguments utiles à la compréhension sont indiqués.

Chronologiquement, un scenario COPS peut comporter les quatre phases suivantes :

1. Etablissement des paramètres de sécurité : Cette phase facultative est située avant toute ouverture d'un client réel. Elle permet de négocier les paramètres de sécurité (Identifiant de clé, numéro de séquence, algorithme) qui vont être utilisés dans les objets sécuritaires des messages suivants. Elle se compose des deux messages suivants :
   - Ouverture d'un client de type 0 (réservé à la sécurité) (OPN) : Le PEP y indique un numéro de séquence, un identifiant de clé et l'algorithme de cryptage qui seront utilisés par la suite.
   - Acceptation de l'ouverture (CAT) : le PDP accepte les conditions de sécurité en retournant les paramètres de sécurité précédemment reçus.
2. Ouverture d'un client : Cette phase ouvre un nouveau client, au sens type de politique. Plusieurs clients peuvent être ouverts simultanément, selon les services à assurer au niveau du PEP. Pour distinguer les différents clients ouverts, on utilise le champ `Client-type` du PDU COPS (cf 2.1.2). La phase d'ouverture se compose des deux messages suivants :
   - Ouverture du client (OPN) : Requête d'ouverture du client.
   - Acceptation de l'ouverture (CAT) : le PDP accepte l'ouverture.

3. Opérations : Cette phase symbolise le cœur du fonctionnement de la gestion par politique. Selon le type de client ouvert et le modèle de politique usité, la contenance et la sémantique des échanges peut différer. Le scénario standard de la phase d'opération est le suivant :
   - Requête (REQ) : l'occurrence d'un évènement au sein du PEP (Timer, valeur d'un objet stocké dans une MIB, réception de trafic) le contraint à effectuer une demande de décision politique qu'il effectue sous forme d'un message REQ, paramétré par un identifiant de requête (Id).
   - Décision (DEC) : Le PDP retourne une décision.
4. Fermeture du client : La fermeture (CC) peut être formulée par les deux entités ; Le PEP peut fermer un client car il n'est plus utilisé. Un PDP peut le fermer en redirigeant le PEP vers un autre PDP.

4.1.2 Le PDU COPS

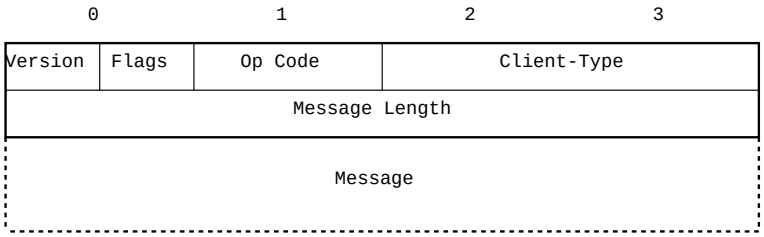


FIG. 4.2 – Le PDU COPS.

Le PDU COPS, représenté sur la figure **??** se compose d'un en-tête suivi d'une succession d'objets. L'en-tête comporte les champs suivants :
- `Version` (4 bits) : indique la version du protocole (actuellement 1).
- `Flags` (4 bits) : permet de véhiculer des indicateurs. Actuellement, un seul indicateur est codé : `0x01` qui désigne un message spontané, c'est-à-dire non déclenché par un autre message.
- `Op Code` : représente le code de l'opération effectuée. COPS reconnait dix codes différents.
- `Client-Type` : spécifie le type de client, au sens type de politique, que le message véhicule. Par exemple, la gestion politique d'allocation de ressource via RSVP possède le numéro `0x0001` .
- `Message Length` : indique la taille totale du message COPS, l'en-tête étant inclus.

4.1.3 Les objets COPS

Le message véhiculé par le protocole est formé d'une succession d'objets dont la présence est indiquée par l'opération (cf `OpCode`). D'après la figure **??**, chaque objet COPS comporte l'en-tête suivant :
- `Length` : indique la longueur de l'objet, l'en-tête étant inclus.
- `C-Num` : identifie le type d'objet. COPS reconnait seize objets différents.
- `C-Type` : définit pour chaque objet des sous-catégories.

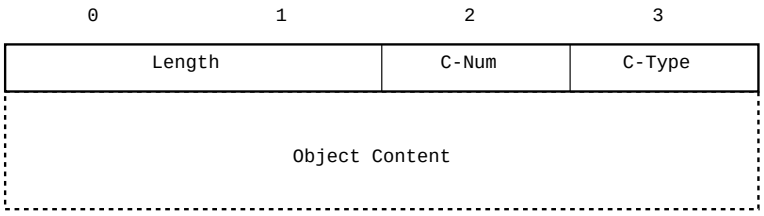


FIG. 4.3 – Le Format d'un objet COPS.

4.2 Le modèle de délégation

Le modèle de délégation (*outsourcing*) fut le premier implanté dans une architecture de gestion de réseau par politiques. Son but initial est d'effectuer une gestion politique des protocoles de signalisation, et particulièrement RSVP. RSVP est un protocole qui permet de réserver les ressources nécessaires à un trafic spécifié. Chaque requête de réservation ne pouvant être acceptée systématiquement, le modèle de délégation permet d'effectuer un contrôle d'admission qui décide de satisfaire ou non la requête initialement reçue.

Cette section présente le modèle de délégation défini par l'IETF [?] ainsi que son application à l'approche RSVP.

4.2.1 Modèle de fonctionnement

Le modèle de délégation répond à un fonctionnement général du type `Evénement->Requête->Décision` Les événements considérés sont variés et choisis dans le cadre de la gestion politique implantée. Ils peuvent être par exemple l'occurence d'une exception dans le programme du PEP, le dépassement d'un seuil dans une MIB, la réception d'un paquet de supervision, ...

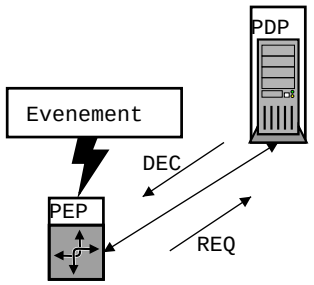


FIG. 4.4 – Le modèle de délégation.

La requête effectuée par le PEP est paramétrée par l'événement ainsi que toute grandeur pouvant être un critère de décision du PDP. En retour, le PDP envoie une décision qui est une acceptation ou un refus.

4.2.2 Exemple de politique de délégation : la gestion de RSVP

Le protocole RSVP

Situé au dessus d'UDP (port n°3455), ou directement d'IP dans la pile de protocoles, RSVP est un protocole de signalisation qui permet de réserver des ressources tout au long du chemin emprunté par un trafic spécifié [?]. Il a été développé par le groupe de travail *intserv* de l'IETF, qui vise à définir une architecture de service intégrés. Supportant le *multicast*, RSVP est particulièrement utilisé dans le domaine du multimédia, domaine qui nécessite un fort contrôle du délai de transit de ses paquets, mais aussi une bande passante garantie.

L'implantation de RSVP nécessite une modification de la structure des routeurs classiques (figure **??**). En effet, la fonction de routage (classification, ordonnancement) des paquets est maintenant paramétrée par la signalisation précédemment effectuée par RSVP, elle-même régie par un système de contrôle d'admission.

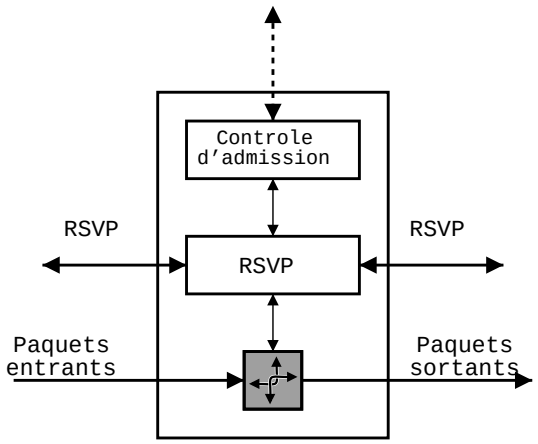


FIG. 4.5 – Architecture d'un noeud avec services intégrés.

Considérons dans son ensemble une connexion nécéssitant une réservation de ressources, (figure **??**). Le scénario établi par RSVP se déroule en deux temps :

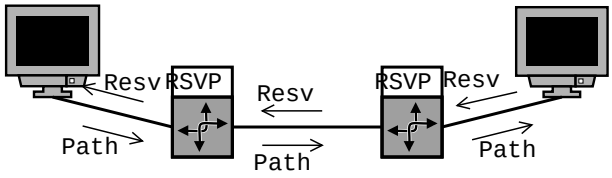


FIG. 4.6 – Le modèle RSVP de réservation.

1. La source émet un message de type `Path` tout au long du chemin (ou arbre *multicast*) qui la relie à ses destinataires. Cette phase permet de découvrir le chemin de réservation.
2. Les destinataires répondent par un message `Resv` qui effectue alors la réservation au niveau de chaque élément parcouru.

Le contrôle d'admission

Le contrôle d'admission a pour but de gérer les demandes de réservation effectuées par RSVP. Il nécessite d'implanter un PEP au sein de chaque nœud RSVP. Son fonctionnement est le suivant (figure **??**) : pour chaque requête de réservation reçue, le PEP envoie une requête COPS à son PDP pour savoir s'il doit accepter ou refuser la requête initiale.

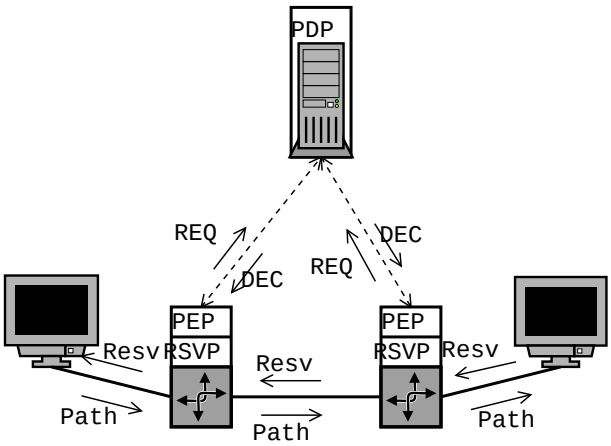


FIG. 4.7 – Le contrôle d'admission.

4.3 Le modèle de provision

Le modèle de provision (*provisionning*) a été développé afin de pouvoir effectuer la gestion politique d'une approche de qualité de service différente de RSVP : l'approche de services différenciés (*diffserv*). L'approche *diffserv* [**?**] consiste à marquer chaque datagramme IP d'une classe de service qui va conditionner son traitement dans chaque élément traversé. Son interêt principal réside en l'absence de protocole de signalisation supplémentaire qui assure la gestion de ce service. RSVP semble en effet présenter une surcharge protocolaire très importante notamment dans les éléments de coeur du réseau, où la densité de trafic est conséquente. De plus, d'un point de vue politique, on comprend qu'il est impossible à un équipement traversé par du trafic *diffserv* de requérir une décision politique pour chaque datagramme IP reçu. Le modèle de provision permet donc à un PEP de prendre des décisions politiques localement. Pour se faire, durant une phase d'initialisation, Le PDP pourvoit ses PEP d'une base de données de décision applicables aux services déployés sur le réseau que l'on appelle la PIB (*Policy Information Base*).

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

4.3.1 Modèle fonctionnel

De manière analogue à l'approche d'administration de réseau basée sur SNMP, le modèle de *provisioning* est fondé sur quatre modèles fondamentaux :

- Modèle de l'information : Le langage utilisé pour définir les informations stockées dans un PEP est SPPI (*Structure of Policy Provisionning Information*).

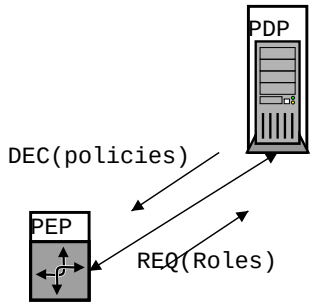


FIG. 4.8 – Le modèle de provision.

- Modèle de la communication : COPS-PR (*COPS for PRovisioning*), une extension de COPS est utilisée pour communiquer entre PEP et PDP [?].
- Modèle de l'organisation : L'architecture standard de gestion par politique est conservée ; L'application des politiques est effectuée par un PEP, le PDP étant toujours un organe de décision.
- Modèle des fonctions : Dans le modèle de *provisioning*, tous les types de politiques (Sécurité, Usage, Installation, ...) peuvent être gérés.

4.3.2 Le modèle de l'information : SPPI

Il existe une forte analogie entre le modèle d'information utilisé en administration réseau et dans la gestion par politiques. Ainsi, SPPI [?], langage dérivé d'ASN.1, reprend de nombreux concepts définis dans SMI. Si SMI utilise la notion d'objets gérés pour désigner toute grandeur stockée (quelle soit de type simple, complexe ou tableau), de manière analogue, SPPI utilise la notion classe de provision (PRC) qui représente toute définition de table ou de ligne. De même, toute instance de définition de ligne est appelée instance de provision (PRI).

Prenons l'exemple de la définition d'une table contenant une liste d'adresse IP. Cette table contient deux colonnes, la première représente l'index des lignes, l'autre les adresses IP stockées. L'expression SPPI de cette définition va utiliser la macro `OBJECT-TYPE` pour définir chaque PRC, soit :

- PRC de table (*IpFilterTable*)
- PRC de ligne (*IpFilterEntry*)
- PRC d'objets de ligne (*IpFilterPrid* et *IpFilterAddr*)

L'expression complète de la définition est décrite ci-après :

Listing 4.1 – Extrait de spécification SPPI

```
IpFilterTable OBJECT-TYPE
    SYNTAX          SEQUENCE OF FrwkIpFilterEntry
    PIB-ACCESS      install
    STATUS          current
    DESCRIPTION
        "Filter definitions.  A packet has to match all fields in a
        filter.  Wildcards may be specified for those fields that
        are not relevant."
    ::= { frwkClassifierClasses 2 }

IpFilterEntry OBJECT-TYPE
    SYNTAX          FrwkIpFilterEntry
    STATUS          current
```

```
14          DESCRIPTION
                "An instance of the frwkIpFilter class."
16          PIB-INDEX  { IpFilterPrid }
            UNIQUENESS { IpFilterAddr }
18          ::= { frwkIpFilterTable 1 }

20      IpFilterEntry ::= SEQUENCE {
                IpFilterPrid          InstanceId,
22              IpFilterAddr          InetAddress
        }
24
        IpFilterPrid OBJECT-TYPE
26          SYNTAX          InstanceId
            STATUS          current
28          DESCRIPTION
                "The index used to reference each row."
30          ::= { IpFilterEntry 1 }

32      IpFilterAddr OBJECT-TYPE
            SYNTAX          InetAddress
34          STATUS          current
            DESCRIPTION
36              "The IP address [INETADDR] to match against the packet's
                 destination IP address. frwkIpFilterDstPrefixLength
38               indicates the number of bits that are relevant. "
            ::= { IpFilterEntry 2 }
```

4.3.3 La PIB

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

Chaque PRI et noeud de l'arbre sont munis d'un nombre entier qui constitue leur identifiant. On peut ainsi référencer de manière unique chaque PRI en concaténant son identifiant à celui de chacun de ses noeuds pères.

Enfin, les MIBs et PIBs sont stockées dans un arbre identique. Ainsi, si l'identifiant de chaque MIB a pour entête `1.3.6.1.2.1.`, chaque PIB a pour en-tête `1.3.6.1.2.2.`.

La PIB framework

Le modèle de provisionning peut supporter de nombreux types de politiques qui peuvent être installés de manière simultanée. On définit une PIB par type de politique à gérer. Neanmoins, il existe des informations qui sont communes à toutes les politiques en vigueur. La PIB *framework* [**?**] a pour but de stocker ce genre d'informations. Elle permet de stocker les informations inhérentes à un PEP (rôles, ...) un jeu de filtres standards (filtre sur adresse IP, adresse MAC, numéro de port TCP, ...) et des informations relatives aux autres PIBs actuellement instanciées.

La PIB *framework* contient les cinq groupes de base suivants :

- Groupe des classes de base : contient des PRC qui décrivent les limitations, la configuration courante et les PRC supportées par le PEP.
- Groupe des capacités : contient une description des caractéristiques du PEP ainsi que la combinaison de rôles qui lui sont affectés.
- Groupe des classificateurs : contient un ensemble de jeux de filtres qui permettent de trier des datagrammes.

– Groupe des marqueurs : contient un ensemble de PRC qui permettent de marquer des datagrammes, par exemple en fonction de leur priorité, dans le cas d'un fonctionnement dans un environnement *diffserv*.

4.4 Les méta-politiques

Le concept de méta-politiques [?] est en cours de développement. Il est apparu afin de faire évoluer le modèle de gestion par politiques actuel. En effet, celui-ci présente quelques limitations.

Tout d'abord, Le modèle actuel place son intelligence au sein du PDP ; le PEP n'est qu'un exécutant des politiques ne possédant aucune mémoire. Cette caractéristique présente un fort inconvénient que nous pouvons mettre en évidence en prenant l'exemple de l'apparition d'une congestion au sein d'un réseau local administré par une gestion politique. Initialement, tous les PEP sont dans un état de fonctionnement dit normal. Une congestion apparaît. Le PDP provisionne alors chaque PEP avec des valeurs particulières correspondant à l'état de congestion. La congestion disparaît. Les PEP n'ont aucun moyen de retourner dans leur état initial. Le PDP doit alors à nouveau provisionner chacun d'entre-eux avec les données de fonctionnement standard. Ceci provoque une certaine lourdeur de fonctionnement.

Ensuite, la PIB présente certaines limitations. Elle ne permet pas de représenter clairement les politiques en cours d'application ; chaque règle de niveau *business* étant traduite sous forme de donnée dépendante de son implantation. En outre, la PIB présente une structure figée. Une fois son architecture définie, il n'est pas possible de l'étendre, la modifier. Ainsi, pour ajouter de nouveaux types de politiques au sein d'un réseau, il faut redéfinir la PIB induite.

La notion de méta-politique a pour objectif de répondre à ces limitations, en proposant un modèle d'interaction plus souple entre PEP et PDP.

4.4.1 Le concept de méta-politiques

Une méta-politique est une règle qui décrit la manière dont sont appliquées les politiques. Elle s'écrit selon la forme suivante : `Condition->Action`. Une condition est une expression logique, et une action un ensemble de commandes qui installent des PRIs dans la PIB.

Les méta-politiques sont générées par le PDP et utilisées par le PEP. Ce dernier évalue les conditions de chaque meta politique et en cas de succès installe les PRIs conséquentes. L'idée étant que le PEP ne comprend pas les critères d'évaluation de politiques. Il se contente d'évaluer des conditions et d'appliquer les actions conséquentes. De plus, les conditions peuvent provenir de diverses sources : Le PDP, une MIB ou un service quelconque.

Pour reprendre l'exemple d'un accès Internet accordé de 8h00 à 18h00, on pourra trouver au sein d'un PEP la règle illustrée sur la figure **??** :

Lorsque la condition `WorkTime` est vrai, le PEP installe dans une table d'autorisation des adresses destination l'instance `*.*.*.*` qui permet l'accès d'un usager à Internet. Le point important de cet exemple est la forme de la condition ; pour le PEP, elle n'a aucune signification, c'est toutjours le PDP qui connait son sens. Si l'on envisage de changer le temps de travail dans l'intervalle 9h00 :17h00, le PDP adaptera alors ses instants de commandes, mais pour le PEP, le changement sera transparent.

4.4.2 La PIB de méta-politiques

Le modèle d'interaction défini par les méta-politiques est celui de provisionning. La PIB de méta-politiques (MetaPIB) peut être définie à l'aide de SPPI, et COPS-PR est le protocole utilisé pour la communication entre PEP et PDP. La metaPIB est composée de cinq groupes :

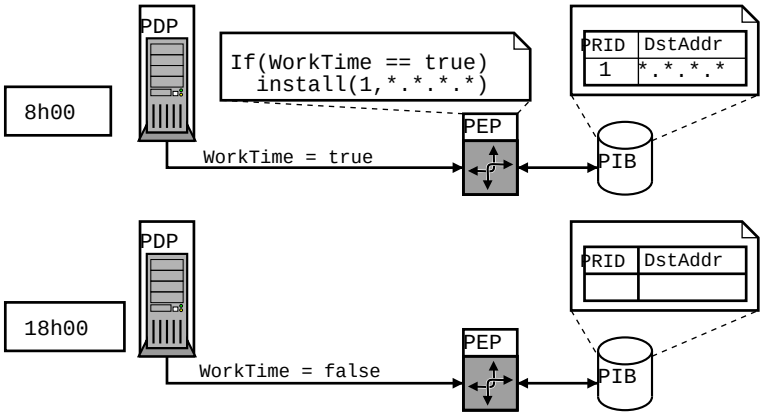


FIG. 4.9 – Architecture d'un noeud avec services intégrés.

- le groupe `Capabilities` : Il contient les PRC qui stockent les capacités et limitations du PEP. Les PRIs de ce groupe, envoyes au PDP lorsque le PEP se connecte, permettent au PDP d'installer une MetaPIB adaptée au PEP.
- le groupe `Base Metapolicies` : contient l'ensemble des classes de metapolitiques en vigueur.
- le groupe `Condition` : contient l'ensemble des classes de conditions des metapolitiques. Les conditions peuvent être simples, complexes ou booléennes.
- le groupe `Action` : contient l'ensemble des classes d'action des métapolitiques, c'est à dire l'ensemble des PRIs à installer pour définir une instance de politique.
- le groupe `Parameter` : indique la provenance des conditions à évaluer et la fréquence d'évaluation. Une condition peut provenir du PDP ou d'une MIB.

4.5 Synthèse

Le niveau *technologique* d'une gestion de réseau par politiques est principalement constitué de deux équipements : le PDP et le PEP. Le protocole standardisé par l'IETF pur dialoguer entre ceux-ci est COPS, un protocole de type client/serveur qui répond à deux modèles de fonctionnement :

- le modèle de délégation : dans ce modèle, à chaque occurence d'un évènement, le PEP demande au PDP la décision à prendre. Ce modèle fur initialement développé afin d'être appliqué à RSVP.
- le modèle de provision : conçu initialement pour être appliqué au modèle de services différentiés, il consiste à pourvoir le PEP d'une base de données de décisions politiques que celui-ci va pouvoir appliquer indépendamment. Cette base de données s'appelle la PIB et présente de fortes similitudes avec la MIB SNMP. En effet, celle-ci est décrite à l'aide d'un langage de spécification : SPPI, une extension de SMIv2.

Dernier concept qui apporte une évolution au modèle de provision du niveau *technologique* : celui de méta-politiques. Ce dernier permet à un PEP de stocker directement des règles de politiques ainsi que leur conditions d'application. Ceci réduit la charge protocolaire induite par le modèle de provision et apporte une vision plus explicite des politiques stockées dans une PIB. Encore en cours d'étude, cette nouvelle approche n'est pas encore passée à l'état de standard.

Chapitre 5

SMI Next Generation

L'approche de gestion de réseau traditionnelle, fondée sur SNMP, utilise un modèle d'information défini avec un langage de description des objets gérés (SMI) ainsi qu'une architecture de stockage (la MIB). La gestion de réseau par politiques, dans le cadre du modèle de *provisioning* définit de même SPPI comme langage de description des PRC et la PIB comme architecture de stockage des PRI. Ces deux modèles sont similaires en de nombreux points et leur domaines d'application est commun : la gestion de réseau. En plus de ces deux approches standardisées par l'IETF, d'autres modèles sont définis dans le cadre de gestions propriétaires. Finalement, le domaine de la gestion de réseau comporte une multitude d'approches et certaines applications présentent des définitions multiples. Par exemple, la gestion de *diffserv* est définie de manière concurrente par une une MIB [?] et une PIB [?].

Conjointement à cette multitude de modèles, les langages de spécifications actuels présentent certaines limitations [?]. Au niveau structure de données, les objets gérés par SMI sont des valeurs simples ou des tables ; la définition SPPI de PRC, quant à elle, ne permet de spécifier que des tables de valeurs simples. Enfin, SMI et SPPI ne sont pas des langages très intelligibles, la définition d'une table est complexe et fastidieuse.

Le groupe de travail *sming* de l'IETF a pour tâche de définir un nouveau langage de définition qui unifie les approches de gestion de réseau et de politique : SMIng (*Structure of Management Information Next Generation*). SMIng est un langage orienté objet qui permet de définir des modèles de données d'une manière indépendante de leurs implémentations.

Ce chapitre a pour objectif de présenter les principaux concepts du langage SMIng. Nous détaillerons tout d'abord les éléments du langage qui permettent de construire une spécification de données. Ensuite nous présenterons un élément important du langage : la traduction de définitions SMIng en données interprétables par une approche de gestion de réseau par politique.

5.1 Présentation générale

SMIng est un langage qui a pour but de spécifier des informations de gestion de réseau d'une manière intelligible pour un être humain et interprétable pour une machine. Les informations sont décrites sous forme de classes qui peuvent être construites intégralement ou héritées (seul l'héritage simple est autorisé). Chaque classe possède un certain nombre d'attributs qui peuvent être un type simple, un sous-type, ou une autre classe. Les types simples de SMIng sont similaires à ceux définis en SMI ou SPPI (Integer, OctetString, Pointer, ...), et peuvent être redéfinis (on parle alors de type dérivé) par la clause `typedef`. Les classes et définitions de types sont définies dans un module ; en règle générale, on définit un module par fichier de spécification.

5.2 Eléments de langage

Cette section présente les fonctionnalités basiques de SMIng : les types, les classes et enfin la structure des fichers de définition [?].

5.2.1 Les types de base et dérivés

SMIng reconnait les douze types de base suivants :

Type	Exemple
OctetString	"This is an example"
Pointer	snmpUDPDomain
Object Identifier	1.3.6.1
Integer32	-123
Integer64	0x80000000
Unsigned32	0xabc
Unsignde64	0x8080000000
Float32	-2.5E+3
Float64	3.1415
Float128	0.001
Enumeration	Enumeration (up(1), down(2), testing(3))
Bits	Bits (read(1), write(2))

TAB. 5.1 – Les types de base SMIng.

A partir de la définition des types de base, SMIng permet la création de types dérivés par le biais de la clause `typedef`. Voici l'exemple d'une définition de type qui permet d'exprimer une adresse IP, en la modélisant par une chaîne de quatre octets :

```
 typedef InetAddress {
  type    OctetString(4);
status  current;
        description
                "Represents a IPv4  Internet address.";
       };
```

5.2.2 Définition de classe

Une classe SMIng est constituée d'attributs et d'évènements. Un attribut représente un élément d'information de la classe, alors qu'un évènement permet de spécifier la possibilité pour une instance de signaler l'occurrence d'un évènement qui peut apparaître de manière asynchrone. De plus, chaque définition de classe SMIng comporte les clauses facultatives suivantes :

- Clause de statut : définit l'état de la classe au sens de sa validité. Les trois valeurs supportées par ce statut sont `current, obsolete, deprecated`.
- Clause de description : décrit le contenu de la classe.
- Clause de référence : indique une référence de document relatif à la classe.

Prenons l'exemple de l'extrait **??** qui permet de décrire une interface réseau, sa vitesse et son statut. Un évènement permet de spécifier le changement d'état de l'interface.

Listing 5.1 – Extrait de spécification SMIng

```
class Interface {
```

```
2
      attribute InstanceId id {
4       access      readonly;
        status      current;
6       description
            "A unique number that reference an Interface object.";
8     };

10    attribute Gauge32 speed {
        access      readonly;
12      units       "bps";
        status      current;
14      description
            "An estimate of the interface's current bandwidth
16           in bits per second.";
      };
18
      attribute OperStatus operStatus {
20      access      readonly;
        status      current;
22      description
            "The current operational state of the interface.";
24    };

26    event linkDown {
        status      current;
28      description
              "A linkDown event signifies that the OperStatus
30             attribute for this interface instance is about to
               enter the down state from some other state (but not
32             from the notPresent state).  This other state is
               indicated by the included value of OperStatus.";
34    };

36    status        current;
      description
38            "A physical or logical network interface.";

40  };
```

5.3 Traduction de SMIng vers SNMP et COPS-PR

SMIng est un langage indépendant de toute implémentation. En effet, toutes les classes contenues dans un module sont définies de manière abstraite. Le seul type qui se rapporte à un modèle d'information de gestion de réseau est le type `ObjectIdentifier` qui permet de stocker l'OId (*Object Identifier*) d'un objet SNMP ou COPS-PR.

Ainsi pour que les définitions SMIng puissent être appliquées à un modèle de gestion de réseau, on a recours à une traduction des éléments définis (le *mapping*). Actuellement, les *mappings* de SMIng vers SNMP [?] et COPS-PR [?] sont définis. On présentera ici celui de SMIng vers COPS-PR.

5.3.1 Définition du *mapping* de SMIng vers COPS-PR

L'expression du *mapping* s'effectue sous forme d'une clause inscrite à la fin d'un module SMIng. Elle est composée des clauses facultatives suivantes :

- Clause`oid` : définit l'OId du module COPS-PR.
- Clause `Clienttype` : spécifie les clients politiques à qui s'applique le *mapping*.

- Clause `node` : définit les noeuds de la PIB qui permet de repérer la position des objets traduits dans l'arbre de nommage.
- Clause `prc` : définit la traduction des classes SMIng en PRC COPS-PR.
- Clause `group` : définit un groupe de noeuds dans l'arbre de la PIB.
- Clause `conformance` : défini l'OId de PRC COPS-PR qui représente le groupe de conformité de COPS-PR.

On peut noter que les évènements associés à une classe SMIng ne sont pas mentionnés dans le *mapping*. Ceux-ci sont néanmoins présents dans l'implémentation du PEP ou PDP hôte de la spécification définie. Par exemple, dans le cas de la classe `Interface` définie précédemment, l'occurrence de l'événement pourra déclencher une requête COPS-PR spontanée qui sera du type demande de configuration (REQ), pour un PEP, et provision de décisions (DEC) pour un PDP.

L'exemple **??** illustre le mapping de la classe `Interface` définie en SMIng précédemment :

Listing 5.2 – Extrait du mapping de SMIng vers COPS-PR

```
// éDbut du mapping
copspr {

        // Mapping de tous les clients politiques
        clienttype { all };

        // éDfinition la position de la PIB ``example'' dans l'arbre de nommage.
        node iso        { oid 1;         status current; };
        node  org       { oid iso.3;     status current; };
        node   dod      { oid org.6;     status current; };
        node    internet{ oid dod.1;     status current; };
        node     mgmt   { oid internet.2; status current; };
        node      pib   { oid mgmt.2; status current; };
        node       example { oid pib.1; status current; };

        // mapping de la classe SMIng Interface
        prc interfaceClass {
                // éDfinition de son emplacement
                oid example.1;
                // Mapping de l'index
                pibindex id;
                // Mapping des attributs
                implements Interface {
                        object id;
                        object speed;
                        object status;
                        };
                // Indication du status de la PRC
                status current;
                // Commentaires de la classe
                description
                        "A physical or logical network interface.";
                // Mode d'èaccs entre PEP et PDP
                pibaccess installonly;

        };

        // Commentaires du mapping
        description "A mapping example of the SMIng Interface class.";

// Fin du mapping
};
```

5.4 Synthèse

SMIng est langage de spécification orienté objet qui permet de définir des modèles de données d'une manière indépendante de leurs implémentations. Il a notamment pour objectif de remplacer les langages SMIv2 et SPPI qui permettent respectivement la définition de MIB SNMP et de PIB COPS-PR.

La définition d'objets d'une spécification SMIng s'effectue par le biais de classes fondamentales ou héritées, chaque classe étant constituée d'attributs et d'évènenents.

Enfin, deux extensions du langage existent et permettent de traduire différentes classes définies dans une spécification SMIng en éléments intelligibles par un modèle concrèt de gestion de réseau (approche SNMP et gestion par politique).

Deuxième partie

Travail de développement

Chapitre 6

Développement d'un agent de gestion commune

SMIng est langage de description qui permet de spécifier des données d'une manière indépendante de leur implantation. Dans ce langage, la définition d'une MIB d'administration de réseau s'effectue de la même manière qu'une PIB de gestion par politique. Seul le *mapping* permet de distinguer leur implémentation. Cette unicité de spécification est un argument tendant à favoriser le développement d'une approche commune d'administration de réseaux et de gestion par politique.

C'est dans ce cadre que s'inscrit notre travail. Il consiste à fournir une vision unique de la gestion de réseau en développant une interface qui scinde l'aspect fonctionnel d'une gestion commune de son implémentation.

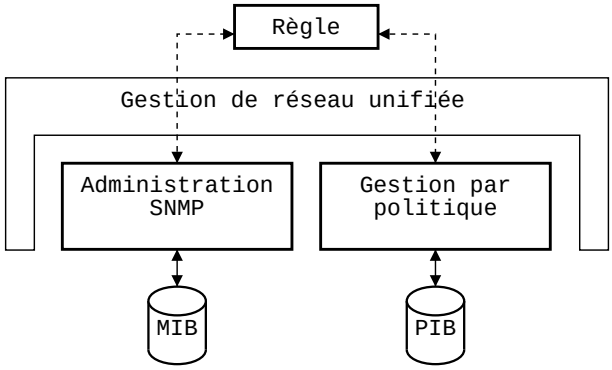


FIG. 6.1 – La vision unique de la gestion de réseau.

Ce type de vision permet de simplifier l'architecture de gestion de réseau et d'en accroître les possibilités. L'idée générale, illustrée par la figure **??** est de construire de manière élémentaire de règles de politiques de gestion où les conditions sont issues d'une MIB et les actions des instances de PRC, et inversement.

Ce chapitre présente *l'agent proxy* développé au sein de l'équipe RESEDAS. Nous étudierons tout d'abord son architecure générale, décrite en terme de fonctionnalités. Ensuite, nous présenterons son implémentation. Une dernière section permettra de conclure quant à l'état actuel

du développement de *l'agent proxy* et le travail restant à effectuer.

6.1 Architecture générale

Cette section présente l'architecture générale de *l'agent proxy*. Nous en présentons tout d'abord l'approche fonctionnelle, puis nous détaillons la structure opérationnelle.

6.1.1 Approche fonctionnelle

L'agent *toolkit* que nous développons est construit selon l'architecture décrite sur la figure **??**. Son objectif est de gérer les objets définis par SMIng comme des objets programmables (au sens objets Java, C++, ...) qui soient distribués. Dans le cadre de notre implémentation, nous avons choisi de générer des objets Java car ceux-ci sont très proches de la définition SMIng. De plus, Java est un langage portable et par conséquent très utilisé dans le cadre d'applications distribuées.

Considérons un objet Java généré à partir d'une classe SMIng ; cet objet va "contenir" la valeur d'une instance de PRC stockée dans un PDP, ou la valeur d'un objet géré par un agent SNMP. On comprend donc que les objets Java et les objets de gestion de réseaux (PRI ou objets gérés) sont liés : tout changement effectué sur l'un doit se répercuter sur l'autre. Chaque objet Java doit donc être muni de services qui puissent dialoguer avec les éléments sous-jacents (agent SNMP ou PDP COPS) afin d'assurer la cohérence des informations maintenues.

L'interface JMX (*Java Management eXtension*) [**?**] fournit une interface de distribution des objets Java, de manière analogue à CORBA (*Common Object Request Broker Architecture*). Néanmoins, si CORBA est dédié à la distribution d'objets applicatifs, JMX a pour domaine la distribution d'objets de gestion de réseau. Le principe de fonctionnement de JMX consiste à stocker les objets gérés (*MBean*) dans un serveur, chargé d'assurer leur accès à distance.

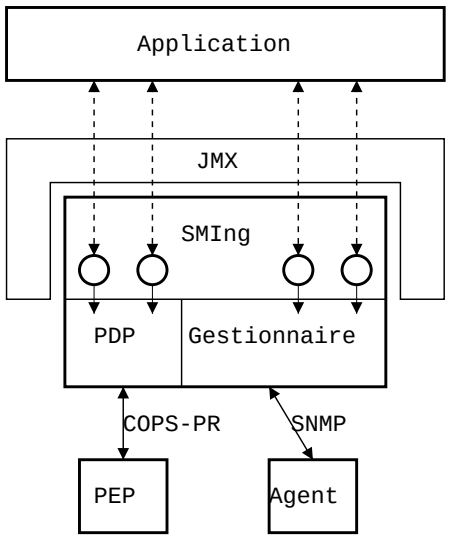


FIG. 6.2 – Architecure générale de l'agent *toolkit*.

6.1.2 Approche opérationelle

D'un point de vue opérationnel, la génération d'objets Java est présentée sur la figure **??**. et comporte les étapes suivantes :

1. Génération de l'arbre abstrait des classes SMIng : la première étape de la génération consiste à analyser les modules SMIng dont on veut générer les classes. Le travail effectué par le *parser* consiste à vérifier que les modules sont écrits en accord avec la définition de grammaire du langage SMIng, puis à générer l'arbre abstrait issu du parcours des modules. Cet arbre abstrait est une représentation arborescente des différentes spécifications lues dans le fichier SMIng source. Ce dernier est stocké dans un fichier appelé *repository*.
2. Génération des classes Java : cette seconde étape consiste à générer des classes Java, correspondant à la spécification SMIng initiale. L'*agent toolkit* va générer une classe Java par classe SMIng en respectant les relations d'héritage, les attributs et les définitions de types. D'autres classes sont générées afin d'assurer les opérations de Lecture/Ecriture nécessaires avec les protocoles d'application (SNMP, COPS-PR).

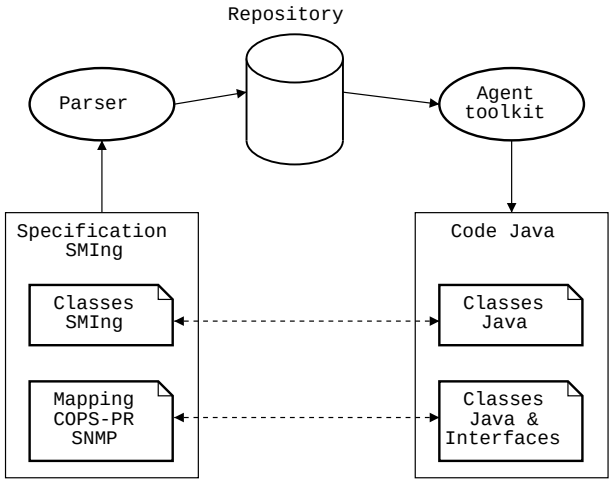


FIG. 6.3 – Les différentes opérations.

6.2 Implémentation

L'implémentation de l'agent *proxy* se scinde en deux parties : le *parser* et l'agent *toolkit*. Un travail prélable au sein de l'équipe a permis de réaliser l'implémentation complète de SMIng, et de son *mapping* vers SNMP. Le travail consiste à implémenter le parser du *mapping* de SMIng vers COPS-PR, ainsi que de completer *l'agent toolkit* afin qu'il puisse générer les nouvelles classes stockées dans le *repository*.

6.2.1 Le *parser*

La réalisation du *parser* est réalisée à l'aide de l'outil JavaCC (*Java Compiler Compiler*) [**?**], et la construction de l'arbre abstrait avec JJTree (préprocesseur de JavaCC). JavaCC est un compilateur de compilateur libre de droits qui a la particularité de permettre l'inclusion de code Java dans les fichiers sources de grammaire du compilateur à générer.

L'IETF fournit la description ABNF [?] (*Augmented Backus Normal Form*) de SMIng et de ses différents *mappings*. Le travail de génération du parser du *mapping* a donc consisté à traduire la grammaire ABNF en grammaire JavaCC. De plus, des éléments de codes supplémentaires ont été implantés afin de d'ajouter des fonctionnalités à l'arbre abstrait généré par JJtree.

Description JavaCC du *mapping* vers COPS-PR

L'exemple **??** est extrait de la description de la grammaire de la section de *mapping* de SMIng vers COPS-PR. Il permet de mettre en évidence les principales caractéristiques du langage de JavaCC qui sont les suivantes :

- La description des différentes sections (appelées non-terminaux) de grammaire est effectuée par l'appel successif de méthodes. Ici, la section de *mapping* de SMIng vers COPS-PR est décrite par la méthode `copsStatement()` et appelle les méthodes `lcIdentifier()` (ligne **??**), `oidStatement()` (ligne **??**) et `prcStatement()` (ligne **??**) qui correspondent à des sections du mapping.
- Le nombre d'occurrence d'une section est spécifiée par les caractères `?` et `*`. `?` signifie que cette section peut apparaître zéro ou une fois, et `*` zéro ou plusieurs fois.
- L'ajout de la section décrite dans l'arbre abstrait est spécifié par la directive `#Nom_du_noeud:` soit ici `#CopsStatement:` (ligne **??**) qui entraine la création d'un objet de la classe `CopsStatement` générée par JJTree.
- Il est possible d'inclure du code Java dans une méthode (ligne **??**) mais aussi à l'appel d'une méthode. Par exemple, `jjtThis.setLcIdentifier();` où `jjtThis` est l'instance de `CopsStatement` courante (ligne **??**).
- Les éléments de grammaire figés (appelés tokens) sont décrits entre crochets (`<copsprKeyword>`, `<LBRACE>`, ...) et spécifiés précédemment dans la grammaire.

Listing 6.1 – Extrait de la grammaire JavaCC du *mapping* de SMIng vers COPS-PR

```
void copsStatement() #CopsStatement:
{
        Token t;
}
{
        <copsprKeyword>
        (
        lcIdentifier()
        {jjtThis.setLcIdentifier();}
        )?
        <LBRACE>
        (
        oidStatement()
        {jjtThis.setOidStatement();}
        )?
        // ...
        (
        prcStatement()
        {jjtThis.setPrcStatement();}
        )*
        // ...
        <RBRACE> <SEMI>
}
```

L'extrait **??** de description de grammaire JavaCC s'interprète ainsi de la manière suivante : Une section de *mapping* de SMIng vers COPS-PR commence par le token `<copsprKeyword>`, c'est à dire le mot clé "`cops-pr`" dans le fichier source (l'association entre `<copsprKeyword>` et

"`cops-pr`" étant définie au préalable). Ce mot clé est suivi d'un identifier commençant par une lettre minuscule (ligne **??** ; le retour de la fonction `lcIdentifier()` entraine l'exécution de code java de la ligne **??**). Une section de type `oidStatement()` peut être spécifiée zéro ou une fois et si elle est présente, le code de la ligne **??** sera exécuté autant de fois.

Génération de l'arbre abstrait

De manière très générale, durant sa phase d'analyse, un compilateur utilise une structure hierarchique pour représenter les opérations spécifiées dans le programme source que l'on appelle arbre abstrait [**?**]. C'est à partir de celui-ci que le compilateur va pouvoir effectuer son travail de traduction du programme source vers un programme cible.

Dans le cadre de notre travail, le *parser* génère l'arbre abstrait issu d'une spécification SMIng à l'aide du préprocesseur de JavaCC, JJtree, et le stocke dans un fichier nommé *repository*. Cet outil va permettre la génération d'un tel arbre sous forme de classes Java. D'un point de vue technique, la spécification d'un noeud de l'arbre s'effectue par l'ajout de l'instruction `#<nom_du_noeud>:` dans la grammaire JavaCC, comme le montre la ligne **??** de l'exemple précédent. La figure **??** illustre une partie de l'arbre abstrait ainsi généré.

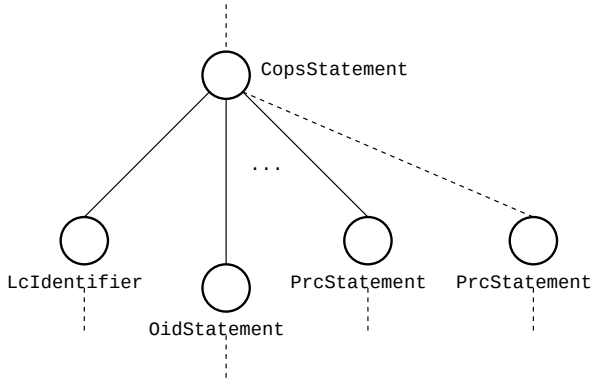


FIG. 6.4 – Exemple d'arbre abstrait généré par le mapping de SMIng vers COPS-PR.

6.2.2 L'agent *toolkit*

L'agent *toolkit* est un outil de génération de code dont le rôle est de lire l'arbre abstrait stocké dans le *repository* et de générer les classes Java correspondant à la spécification SMIng originale.

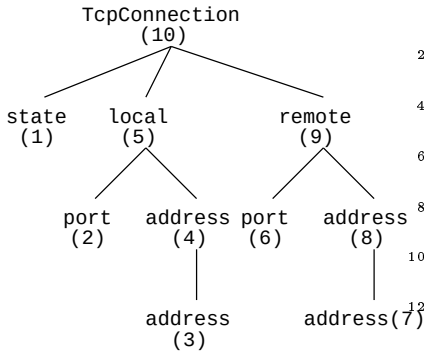
La génération d'objets

Comme l'illustre la figure **??**, deux sortes de classes Java sont générées :

- les classes Java correspondant aux classes SMIng. Celles-ci sont indépendantes du protocole sous-jacent. Seules, les classes correspondant à des définitions de types (`typedef`) SMIng sont véritablement associées à un protocole de communication (SNMP ou COPS-PR).
- les classes Java inhérentes au *mapping*. Celles-ci permettent de lier les objets gérés par les classes Java précédentes au objets stockées dans une PIB ou MIB, par le biais de l'OId défini pour chaque objet résultant du *mapping*.

La figure **??** décrit la manière dont les objets sont automatiquement générés en prenant l'exemple de la classe SMIng de connexion TCP . La partie gauche représente l'arbre abstrait

de cette classe. La racine correspond au nom de la classe, les noeuds intermédiaires, à des types simples ou des classes, et les feuilles à des types simples. Le code généré,illustré sur la partie droite, débute par la création des feuilles (lignes ?? à ??) et remonte jusqu'à la racine de l'arbre. Chaque noeud crée est lié à son père par le biais de la méthode `setSming`*attribute* (ligne ??). En outre, chaque feuille, correspondant à un type simple, recoit comme paramètre lors de sa création l'adresse de l'agent SNMP ou COPS-PR qui contient l'objet géré ainsi que son Oid.




```
  state = new TcpConnectionState(...);
2 localport = new InetPortNumber(...);
  localaddressaddress =  new InetAddress(...);
4 localaddress =  new InetNetworkEndpoint();
  localaddress.smingSetlocaladdress(localaddressaddress);
6 local = new InetTransportEndpoint();
  local.smingSetlocalport(localport);
8 local.smingSetlocaladdress(localaddress);
  remoteport = new InetPortNumber(...);
10  ...
  tcpconnection = new TcpConnection(...);
12  ...
```

FIG. 6.5 – Schema d'implantation du processus de génération

Intégration des objets dans un agent JMX

En plus des classes Java précédemment générées, l'agent *toolkit* génère un agent JMX, qui va servir de conteneur à l'ensemble de ces classes. C'est lui qui va déclencher l'envoi de requetes SNMP/COPS-PR pour que chaque objet Java accède à la valeur de l'objet géré qui lui correspond.

Chaque objet Java est ainsi distribué et accessible à distance par le biais de son nom d'enregistrement dans l'agent.

Développement d'un service GET dans COPS-PR

La génération d'objets dont l'implantation sous-jacente est COPS-PR pose un problème protocolaire. En effet, aucun service de type GET n'est décrit dans la spécification du protocole. Ceci s'explique simplement par le fait suivant : bien que COPS-PR permette de manipuler des objets (PRI) très similaires à ceux définis dans le cadre d'une gestion SNMP, ceux-ci ont une utilisation et une sémantique différents. En effet, les objets gérés par SNMP ont pour rôle principal d'être lu et peuvent être découverts (par le biais du service GET-NEXT), alors que ceux définis par COPS-PR doivent être stockés et sont normalement connus par tous les éléments succeptibles d'y accéder.

C'est pourquoi, dans le cadre de notre travail, il a fallu envisager une manière de lire les PRI stockées dans une PIB. Deux solutions ont été envisagées :

1. Mettre les éléments stockés dans la PIB à disposition d'un agent SNMP. Leur lecture pourrait alors s'effectuer à l'aide du service GET SNMP.
2. Implanter un service GET dans COPS-PR.

La solution retenue actuellement est la seconde. Neanmoins, celle-ci n'exclue nullement le développement de la première. En effet, les objets générés par notre agent *toolkit* qui dépendent d'un protocole sous-jacent sont paramétrés par son type. Ainsi, on peut parfaitement envisager que les objets implantés avec vers COPS-PR fassent des appels SNMP pour connaitre leur valeur.

Le développement du service GET dans COPS-PR (figure **??**) a nécessité la création d'un nouveau type de client (`clienttype`) : *SMIng management* (0x03). Celui-ci respecte toutes les spécifications décrites dans le standard du protocole et utilise les messages GET et REQ définis dans COPS-PR pour effectuer son opération.

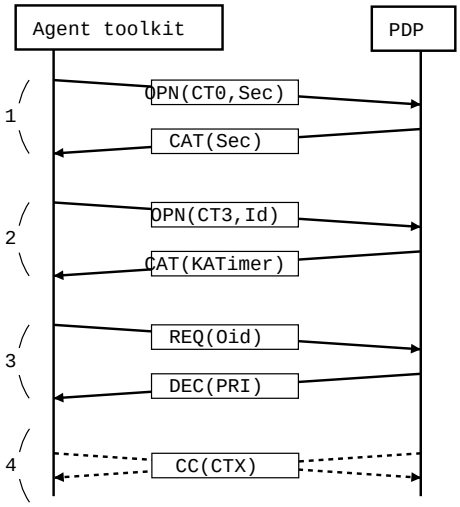


FIG. 6.6 – Le service GET COPS-PR.

6.3 Etat actuel et Développement futur

La majorité du travail de développement est terminée. L'analyseur syntaxique de la spécification du *mapping* de SMIng vers COPS-PR est achevé, l'agent *toolkit* génère correctement l'ensemble des classes Java et un mini-PDP qui permette l'accès à sa PIB via le service GET de COPS-PR est opérationnel.

Les deux principales taches qu'il me reste à effectuer sont les suivantes :

- effectuer le lien entre les objets correspondant à des définitions de types et la couche COPS-PR. Ceci nécessite de résoudre le problème d'accès à des données par des sources diverses, par le biais d'un protocole qui fonctionne en mode connecté.
- implanter un service de notification, en accord avec celui défini dans COPS-PR, qui puisse automatiquement mettre à jour la valeur stockée par une classe Java, si la PRI qui lui correspond change.

Chapitre 7

Gestion par politique de l'environnement FLAME

Une forte part de l'évolution des réseaux et télécommunications semble tournée vers les réseaux actifs et programmables. Un réseau actif diffère d'une architecture classique par le fait que les paquets qui transitent par un noeud peuvent contenir du code exécuté lors de son traitement. Un réseau programmable, quant à lui, présente des noeuds munis d'une interface de programmation qui permet une modification du code s'exécutant dans leur environnement.

Les services déployés sur un réseau sont toujours plus dynamiques et volatiles, et par conséquent, de plus en plus difficiles à administrer par une approche classique, du type SNMP. La supervision de réseau semble donc s'orienter vers une implantation active, permettant un contrôle plus proche du fonctionnement en cours.

FLAME (*Full-Fledged Loria-Alcatel Active Management Environnement*) [**?**] est une architecture munie d'un environnement d'exécution dédié à la supervision des noeuds actifs de l'Internet. Développé en collaboration entre l'INRIA et Alcatel, elle a pour objectif d'apporter une alternative à la gestion classique SNMP en proposant un déploiement actif des fonctions de gestion de réseau. Le travail que j'effectue dans ce projet consiste à gérer l'environnement d'exécution FLAME par une architecture de gestion par politiques.

L'objectif de ce chapitre est de présenter une implantation concrète de gestion par politique dans le cadre d'un projet de supervision. Dans un premier temps, nous présenterons l'environnement FLAME. Ensuite, nous détaillerons le concept de politique retenu. Enfin nous conclurons sur le développement futur des éléments à implanter.

7.1 FLAME

FLAME est un environnement d'exécution implanté dans un noeud de réseau actif (figure **??**). Son élément central est le gestionnaire du noeud que l'on appelle environnement d'exécution (EE). Il a pour mission de télécharger les applications actives (AA), les exécuter, et les stopper.

Le serveur d'application (*Code Server*) est un serveur web dédié à la distribution de code au sein des noeuds actifs. Le chargeur de code (*Code Loader*) effectue les demandes de téléchargement auprès d'autre serveurs de code et vérifie le code reçu.

Du côté des applications actives, l'API (*Application Programing Interface*) *Core* fournit les fonctions élémentaires pour dialoguer entre EE et AA (réception de paquets, chargement d'API supplémentaires, ...).

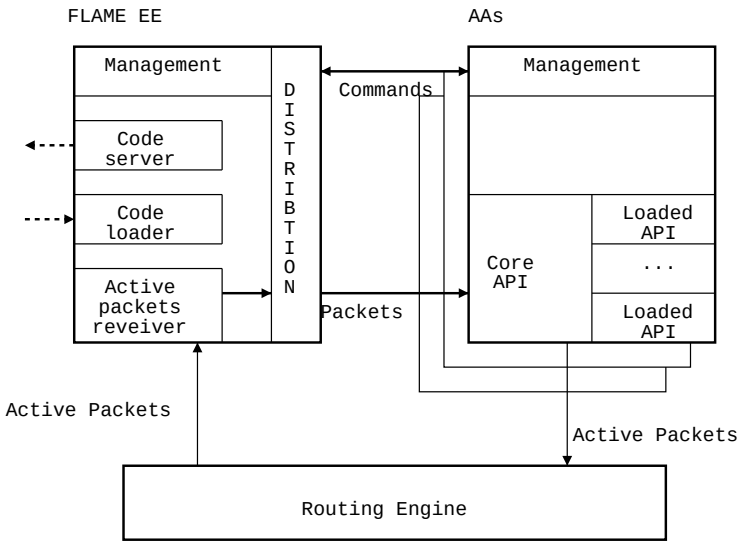


FIG. 7.1 – L'architecture FLAME.

Le fonctionnement d'un noeud FLAME est le suivant : lors de la réception de paquets actifs, l'environnement d'exécution va vérifier que le code à exécuter est présent, sinon il va le télécharger depuis un serveur de code distant. Ensuite, chaque paquet actif sera traité par l'application active qui lui correspond, avant d'être restitué au processus de routage.

7.2 Intégration d'une gestion par politique

L'intégration d'une gestion par politique dans l'environnement FLAME a pour but de contrôler les différentes opérations effectuées par l'environnement d'exécution et de parametrer l'exécution des applications actives. Deux modèles de fonctionnement de politiques sont prévus ; un modèle d'*outsourcing* qui gère le contrôle des opérations, et un modèle de *provisioning* qui permet de stocker au sein de l'environnement d'exécution les paramètres des applications actives.

L'architecture que nous avons développée s'apparente à un seul niveau technologique : elle consiste à ajouter un PDP qui régit des PEP s'exécutant dans leur environnement d'exécution. Nous n'avons pas développé de niveau *business* car la seule règle de gestion politique de l'environnement qui soit définie est la suivante :

Une opération est exécutée par l'environnement d'exécution si et seulement si tous les paramètres véhiculés dans sa requête et leur valeur sont autorisés par le PDP.

7.2.1 Gestion politique des opérations de l'EE

Le contrôle des opérations de l'EE est effectué par une approche d'*outsourcing*. Pour chaque requête reçue, correspondant à une opération, le PDP va répondre par une décision de type acceptation ou refus.

Les opérations

Les 6 opérations suivantes ont été recensées comme nécessitant une autorisation du serveur politique avant d'être exécutées :

- *Code downloading* : lorsque le code qui implémente une AA n'est pas présent dans l'EE, il est téléchargé depuis un site distant. Le contrôle politique de cette opération a pour but de vérifier que le code n'est effectivement pas installé, et que le site de téléchargement est connu.
- *AA Instantiation* : lorsque le code d'une AA est présent dans l'EE et qu'une requête arrive pour cette AA, le serveur de politiques doit autoriser son instantiation en fonction de son nom et du nombre de fois que cette AA s'est arrêtée en dysfonctionnement.
- *AA running* : la décision de continuer l'exécution d'une AA doit être régulièrement vérifiée. Ceci qui fournit un moyen indirect de stopper l'exécution d'une AA sur tous les noeuds ou elle est déployée.
- *API Function* : les AA peuvent déployer des API en plus de l'API *Core* (API de base de l'EE). L'autorisation permet un contrôle de l'API qu'une AA peut télécharger, de même que les méthodes qu'elle invoque.
- *User Agent or System Function* : un super-utilisateur (User Agent) permet d'effectuer des opérations de gestion de l'EE ou des AA. Le contrôle consiste à autoriser ou non l'exécution de ces opérations.
- Packets : il est possible d'interroger le PDP pour savoir si un paquet doit être traité par son AA ou non.

7.2.2 La PIB FLAME

Le modèle de gestion par politique des opérations FLAME est effectué par une approche d'*outsourcing*. Néanmoins, une architecture de stockage des différents critères de décision est nécessaire pour assurer la gestion politique des opérations. Pour cela, nous avons choisi d'implanter une PIB au sein du PDP.

Expression des besoins

La PIB que nous avons spécifiée a été conçue dans le souci de respect des critères suivants :

- Intelligibilité : l'architecture de la PIB et le nommage des PRC sont choisis afin d'être compréhensibles par un utilisateur. Ainsi, celle-ci est découpée en trois groupes principaux :
  1. Le groupe `flamePolicyBaseClasses` : stocke toutes les classes inhérentes à la gestion et configuration de l'environnement FLAME ainsi que la PIB.
  2. Le groupe `flamePolicyFilterClasses` : définit l'ensemble des filtres qui seront utilisés dans le cadre de la gestion des opérations FLAME.
  3. le groupe `flamePolicyActionClasses` : définit les actions à effectuer lors de la réception d'une requête de décision.
- Extensibilité : La structure de PIB est extensible, en ce sens que les PRC actuellement définies sont stockées dans les groupes précédents. Ceci permet ainsi d'ajouter une nouvelle PRC à un groupe sans pour autant modifier l'architecture générale du modèle.
- Modularité : La PIB Flame n'est pas une simple base de données de décisions politiques. Un système de pointeur permet d'effectuer une association entre les PRI. Ceci a pour intérêt de permettre la construction de règles politiques complexes (expressions logiques de conditions, enchaînement d'actions).

Architecture générale

Cette section présente le détail des différentes PRC définies dans la PIB FLAME. Elle est illustrée par la figure **??**.

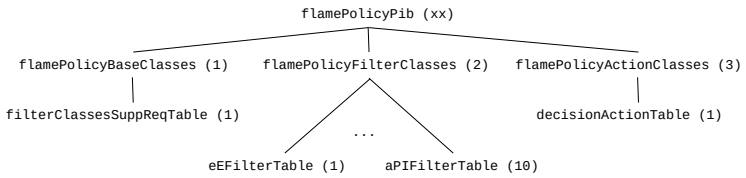


FIG. 7.2 – Architecture de la PIB FLAME.

1. Le Groupe des classes de base : Le groupe `flamePolicyBaseClasses` contient la PRC suivante :
   - `filterClassesSuppReqTable` : Cette PRC permet de spécifier la présence de paramètres dans les requêtes d'opérations reçues. En effet, les requêtes sont caractérisées par un certain nombre de paramètres qui sont obligatoires ou facultatifs. Cette PRC permet donc de vérifier que la requête reçue est conforme au modèle défini.
2. Le groupe des filtres : Le groupe `flamePolicyFilterClasses` contient les filtres qui vont déterminer les décisions à prendre. Chaque filtre contient trois colonnes (Index, Masque, Pointeur vers une action) ; il existe une PRC de filtre par paramètre recevable dans les opérations, soient les dix PRC suivantes :
   - `eEFilterTable`
   - `aAFilterTable`
   - `userAuthFilterTable`
   - `cTypeFilterTable`
   - `iPSourceFilterTable`
   - `uRLFilterTable`
   - `failureFilterTable`
   - `aAtypeFilterTable`
   - `functionFilterTable`
   - `aPIFilterTable`
3. Le groupe des actions : le groupe `flamePolicyActionClasses` décrit les actions à effectuer pour chaque objet reçu dans chaque requête. Il contient la table de décision suivante :
   - `decisionActionTable` : Cette table contient la décision (accepter ou refuser) que le PDP va appliquer lors de la réception d'une requête. Elle contient une colonne par opération. Chaque ligne (PRI) permettra donc de spécifier la décision à prendre pour chaque opération.

Exemple de gestion politique

Afin d'illustrer la description générale de la PIB FLAME précédente, considérons l'exemple de la figure **??**. L'arrivée de paquets actifs nécessitent l'instanciation d'une nouvelle application active AA1 au sein de l'environnement en cours d'exécution EE1. L'environnement sait que l'application va traiter des paquets (d'où CT1) et qu'elle s'est précédemment mal déroulée F1 fois. Connaissant tous ces paramètres, l'environnement d'exécution va effectuer une requête d'autorisation d'instantiation (REQ) auprès du PDP.

A la réception de cette requête, le PDP va vérifier la consistance du message en consultant la table des actions supportées : la lecture de la colonne AAI (*Active Application Instantiation*) lui indique quels sont les objets qui doivent être présents dans le message reçu. De plus, la dernière colonne de la table contient une série de pointeurs vers les tables de filtres des paramètres.

Les tables de filtres permettent de définir les masques de paramètres qui nécessitent une

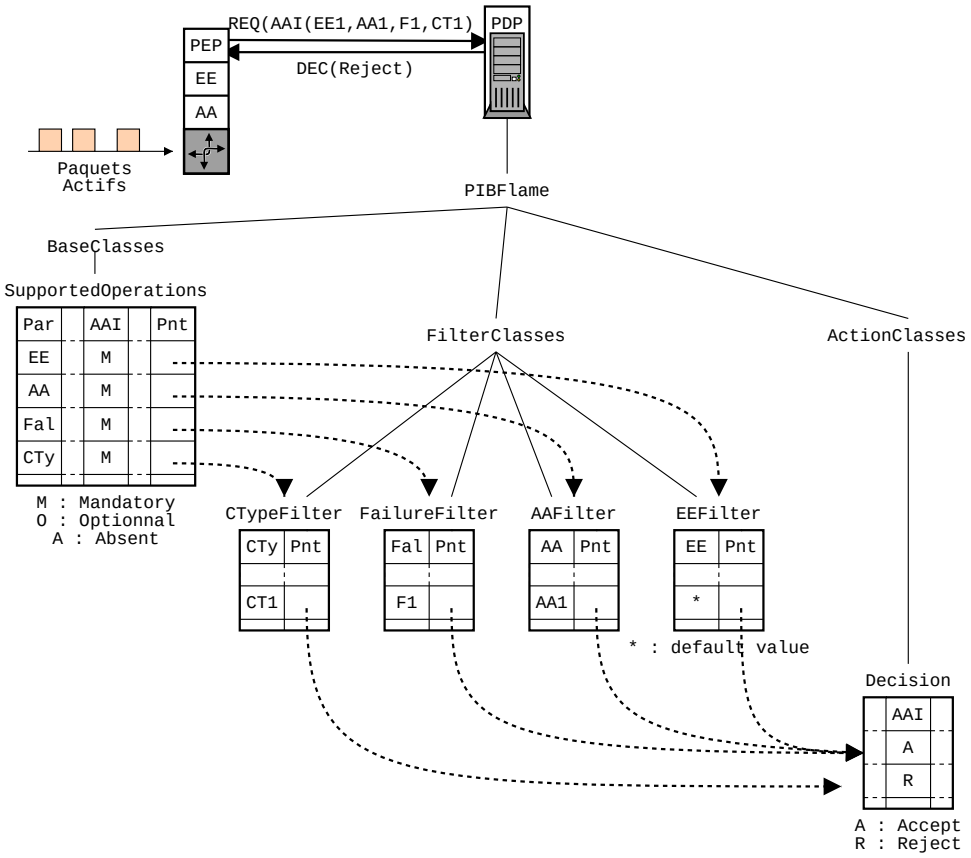


FIG. 7.3 – Exemple de traitement de requête FLAME.

décision politique. Par exemple, la table `AAFilter` contient le masque AA1, ce qui signifie que si un message contient cette valeur, elle va induire une décision, précisée, via un pointeur, dans la table des décisions politiques. La valeur `*` (inscrite dans la table `EEFilter`) permet de définir la décision par défaut à adopter si un paramètre n'est pas mentionné dans sa table de filtres correspondante.

Enfin, la table des décisions permet de spécifier les décisions à appliquer en fonction de la requête reçue. En effet, cette table comporte une colonne par opération, ce qui permet de distinguer les actions à effectuer à la reception d'un paramètre en fonction de l'opération dans laquelle il s'inscrit. Ici, on lit donc la colonne AAI qui permet d'établir l'acceptation des paramètres AA1, EE1 et F1 mais le refus du paramètre CT1. La règle générale de la gestion politique de FLAME étant d'accepter une requête si et seulement si tous ses paramètres sont acceptés, la requête AAI sera refusée.

7.3 Implantation du PDP FLAME

Le développement du PDP FLAME a été réalisé en Java. Comme le montre la figure **??**, ce dernier se scinde en trois parties : le noyau du PDP, la communication avec ses PEP via COPS-PR et l'interface graphique qui permet sa gestion.

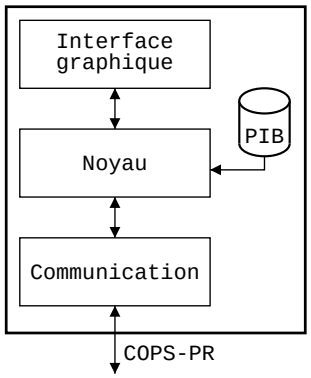


FIG. 7.4 – L'architecture du PDP FLAME.

7.3.1 Le noyau du PDP FLAME

Le noyau du PDP FLAME a pour tâche de traiter les requêtes reçues par la couche de communication. Pour cela, il consulte la PIB qui lui indique, en fonction des paramètres reçus, la décision à prendre. La PIB est constituée de classes Java correspondant aux classes SMIng décrites dans le fichier de spécifications de la PIB FLAME. Un des objectifs de ce travail de développement était d'intégrer des classes générées automatiquement par l'agent *toolkit* dans le PDP FLAME. Les classes Java consistuant la PIB FLAME sont fortement similaires à celles qui pourraient être générées par l'agent *toolkit*.

7.3.2 La couche de communication

La couche de communication permet au PDP d'effectuer toutes les opérations de codage/décodage des PDU qui transitent sur le réseau. Elle s'appuie sur une API Java COPS-PR

développée au sein de notre équipe de recherche. L'ensemble des spécifications protocolaires de l'environnement FLAME sont données en annexe.

7.3.3 L'interface graphique

L'interface graphique du PDP FLAME permet à un utilisateur de manipuler les objets stockés dans la PIB FLAME. Il propose en effet une vue où filtres et décisions sont directement associés.

FIG. 7.5 – La fenêtre principale de l'interface graphique FLAME.

Les différentes fonctionnalités de celui-ci sont les suivantes :
- Menu de selection de filtre : permet de selectionner le filtre courant parmi ceux présents dans la PIB.
- Boutons **Add, Modify, Remove** : permet d'ajouter/modifier/supprimer une PRI de type du filtre courant et la décision qui lui est associée.
- Tableau de PRI : Affiche les PRI du filtre courant.
- Bouton **View Decision** : Affiche l'ensemble des décisions saisies.

7.4 Etat actuel et Développement futur

L'état actuel du PDP FLAME est en cours de développement. Le noyau est opérationnel, il est capable de gérer une PIB modélisée par des classes Java. Il reste simplement à intégrer effectivement les classes Java générées par l'agent *toolkit* dans son architecture afin d'obtenir une application concrète de notre agent.

Au niveau des couches de communication et d'interface graphique, une implantation complète a été réalisée pour deux types de filtres, et deux requêtes sur l'ensemble de la spécification FLAME. Le reste du travail pourrait être effectué afin de finaliser le PDP. De même, d'autre fonctionnalités pourraient être envisagées (comme l'intégration d'un accès LDAP pour la gestion des politiques saisies) afin d'obtenir un logiciel complet.

Troisième partie

Conclusion

Conclusion

Le stage que j'effectue au sein de l'équipe RESEDAS du LORIA s'inscrit dans le cadre de ma formation d'ingénieur Réseaux et Télécommunications, mais aussi dans celui du DEA du même intitulé.

Les recherches que j'ai effectuées afin de cerner le cadre de mon travail m'ont permi de prendre un certain recul par rapport l'ensemble du domaine de la gestion de réseaux. Tout d'abord, son évolution semble conséquente : d'une approche SNMP qui permet principalement d'effectuer la surveillance des équipements, la gestion de réseau s'est tournée vers la gestion par politiques, approche qui permet un contrôle accru des éléments régis, une simplification de la gestion et une forte adéquation avec les services déployés. De même, les modèles de l'informations ont évolués, SPPI est une extension de SMI qui entrevoit la notion d'objets et dernièrement, SMIng offre enfin un langage lisible et véritablement orienté objet.

Néanmoins, si le domaine de la gestion de réseau semble actif, il n'en est pas moins incertain. En effet, les longues discussions qui opposaient COPS à SNMP lors des débuts de la gestion de réseaux par politique sont à nouveau évoquées. SNMPv3 pourrait offrir à la gestion SNMP les avantages de COPS. De plus, cette incertitude se traduit à différents niveaux. En effet, l'IETF a clot le groupe de travail *sming* et SMI-DS (*SMI Data Structure*), un nouveau langage de spécifications, est déjà en cours d'étude. Néanmoins, c'est dans cette période de confusion que la réalisation d'une interface de distribution d'objets de gestion de réseaux, faisant abstraction de leur implémentation, prend toute son importance.

Le travail que j'ai effectué durant mon stage de DEA s'est porté tout d'abord sur la gestion de réseaux par politique, au travers de l'interface donnée par SMIng. Pour cela, les modèles utilisés dans ce domaine ont du être intégrés à une architecture de type client/serveur indépendante. Cette nouvelle approche permet d'accroître les possiblités offertes par les modèles actuels de gestion de réseaux pour plusieurs raisons : tout d'abord, elle apporte une vue plus générale et équivalente des objets rentrant en considération dans l'élaboration d'une architecture de gestion de réseau, comme par exemple, la construction de règles de politiques de gestion. Ensuite, la distribution des objets par l'interface JMX permet d'envisager une architecture générale de gestion non plus systèmatiquement hierarchique où l'accès aux objets s'effectue par un équipement de niveau supérieur, mais aussi de manière distribuée, par des équipements du niveaux égaux.

Enfin, nous avons montré que notre approche d'intégration était valide et permettait notamment de gérer un cas concret d'implantation de gestion : FLAME. Cet environnement de supervision de réseau IP est entièrement construit sur des objets issus de notre architecture de gestion.

D'un point de vue plus personnel, je ne peux que constater que de l'ampleur des connaissances nécessaires au développement d'un projet de recherche. Ayant effectué mon précédent stage chez Schlumberger dans le cadre d'une plate-forme de développement, l'expérience actuelle de travail en laboratoire m'apporte une vision autre du travail apréhendable par la formation d'ingénieur que

nous avons reçu. En outre, je ressens un fort intérêt pour le domaine de la recherche scientifique. En effet, ce travail nécessite une mise à jour continuelle de ses connaissances, et permet de travailler sur des domaines encore inexplorés. Ainsi, l'ensemble de ces intérêts me font envisager une poursuite d'études en thèse de doctorat, au sein du LORIA, dans le cadre d'une étude sur la gestion par politique des réseaux *Peer to Peer*.

Quatrième partie

Annexes

L'administration de réseaux

De part leur continuelle évolution, les réseaux se complexifient, avec des domaines d'applications et services toujours plus nombreux (téléphonie, Internet, réseaux locaux, ...), des supports qui se diversifient (Ethernet, ATM, Wireless, X25, ...), et une forte hétérogéneïté des éléments à connecter. De plus, l'exigence de l'utilisation se fait croissante, en terme de sécurité, fiabilité, performance, ...

Pour répondre à ces besoins, en 1989, SNMP [?] fut standardisé pour l'administration des réseaux TCP/IP.

Ce chapitre a pour objectif de présenter l'approche de gestion de réseau SNMP. Dans un premier temps, nous présenterons le modèle d'une gestion de réseau. Ensuite nous en étudierons les différents composants.

7.5 Modélisation de la gestion de réseaux

Afin d'établir une architecture complète d'administration de réseau, quatre aspects doivent être modélisés [?] :

1. Modélisation de l'information : c'est le choix d'un langage commun de description des ressources administrées, exprimées sous la forme d'objets gérés. L'approche SNMP utilise SMIv2 (*Structure of Management Information version 2*) [?].
2. Modélisation de la communication : c'est le choix d'un protocole de communication entre les différents éléments qui puisse véhiculer les valeurs des objets gérés.
3. Modélisation de l'organisation : détermine le rôle de chaque élément entrant dans le cadre de la gestion. Ici, c'est la définition des agents et gestionnaires. Un exemple d'organisation est présenté sur la figure **??**. Chaque ressource à administrer (équipement matériel ou logiciel) éxécute un agent SNMP qui maintient une base de données, la MIB (*Management Information Base*), relative à son hote, l'état du réseau, ... Un gestionnaire SNMP peut lire et/ou écrire dans la MIB maintenue via le protocole SNMP.
4. Modélisation des fonctions : en termes de critères de fonctionnalité de la gestion de réseaux, les besoins recensés par les utilisateurs sont assez variés. Certains accordent un fort intérêt à l'automatisation de la gestion alors que d'autres se concentrent sur l'aspect sécurité. L'ISO (*International Organizaztion for Standardization*) a répertorié et défini cinq aires fonctionnelles de gestion de réseau [?] :
   - Gestion des fautes : la capacité à détecter, isoler et corriger une opération anormale dans l'environnement réseau.
   - Gestion de la facturation : la capacité à comptabiliser et estimer le coût de l'utilisation d'objets gérés.
   - Gestion de la configuration et du nommage : la capacité d'identification et de gestion des données relatives aux ressources présentes au sein d'un réseau.

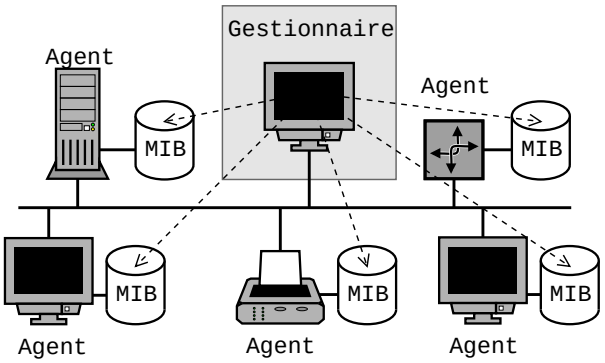


FIG. 7.6 – Exemple d'une architecture SNMP.

- Gestion de la performance : la capacité à évaluer et répertorier l'activité d'un réseau.
- Gestion de la sécurité : la capacité à protéger les objets gérés.

7.6 Le modèle de l'information : SMI

L'administration réseau a pour caractéristique l'hétérogénéité de ses ressources. Par exemple, un réseau local peut compter parmi ses machines des stations de travail SUN mais aussi des PC, qui représentent différemment l'information (comme par exemple emplacement du poids fort sur le code d'un entier de deux octets). Afin de conserver la cohérence des informations véhiculées par les requêtes administratives, il est important d'adopter un codage et une représentation de l'information qui soit unique, quel que soit l'élément où elle est stockée. C'est pourquoi le modèle d'information mis en oeuvre dans l'administration de réseaux est SMI, un langage dérivé d'ASN.1.

ASN.1 est un langage standardisé par l'ISO qui est utilisé dans de nombreux domaines : bancaire, télécommunications, aéronautique, ...Il se présente sous deux aspects : la spécification de données à représenter et leur codage. SMI n'utilise que quelques éléments de ce langage afin de pouvoir représenter les données à maintenir mais aussi leur architecture de stockage [**?**] :

- les types simples : Ils sont définis à partir de types de base prédéfinis (entier, chaine d'octets, booléen, ...). Ils permettent de mieux représenter l'information à stocker. Par exemple, le type `IpAdress` est défini comme une chaîne de 4 octets :

Listing 7.1 – Définition SMI d'un type simple

```
IpAdress ::= OCTET STRING(4)
```

- les types complexes : Ils permettent la construction de nouveaux types. Par exemple le type `SEQUENCE` est un type composé de plusieurs types :

Listing 7.2 – Définition SMI d'un nouveau type

```
IpRouteEntry ::= SEQUENCE {
2                  ipRouteDest      IpAdress
                   ipRouteNextHop   IpAdress
4 }
```

- Les macros : Elles permettent de définir de nouveaux types mais aussi un ensemble de propriétés à appliquer à celui-ci. Dans le cas de l'administration de réseaux, on a défini plusieurs macros (`COMPLIANCE`, `MODULE-IDENTITY`, ...) et notamment la macro `OBJECT-TYPE`

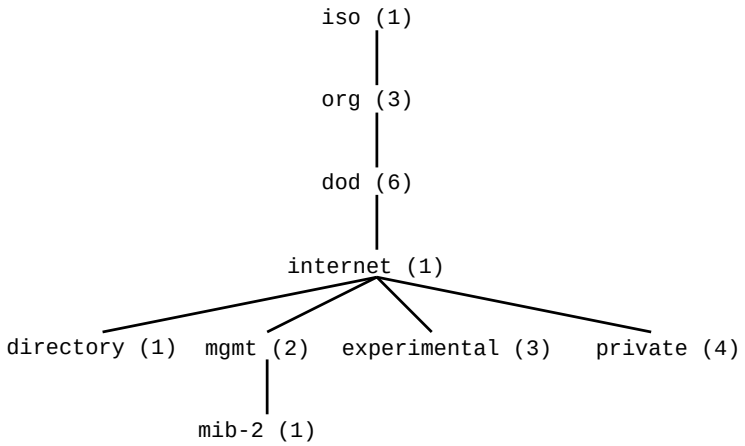


FIG. 7.7 – L'arbre d'enregistrement ISO.

qui munit les objets gérés d'un statut, de droits d'accès et d'une description :

Listing 7.3 – Définition SMI d'une macro

```
tcpMaxConn  OBJECT-TYPE
                SYNTAX  INTEGER          -- le type ASN.1 de l'objet éégr
                ACCESS  read-only        -- le mode d'èaccs
                STATUS  mandatory        -- la éprsence dans la MIB
                DESCRIPTION
                        ``The limit on the total number of TCP connections
                        the entity con support. On entities where the
                        maximum number of connections is dynamic, this
                        object should contain the value -1.''
                ::= {tcp 4}              -- identificateur global
```

7.7 La MIB

La MIB (*Management Information Base*) est une base de donnée qui permet de stocker les objets gérés usités dans le cadre de la gestion de réseau. L'architecture mise en oeuvre pour identifier chaque élément est une structure arborescente nommée arbre d'enregistrement. Ainsi, les objets gérés sont représentés par les feuilles de l'arbre et sont décrits par la macro `OBJECT-TYPE` en ASN.1. Les noeuds (ou groupes) permettent d'établir la hierarchie de l'arbre et sont décrit par le type `OBJECT-IDENTIFIER`.

Chaque élément (noeud et feuille) de la MIB est muni d'un nombre entier qui constitue son identifiant. On peut ainsi référencer de manière unique chaque objet par la concaténation des identifiants de ses ancètres et du sien. Comme le montre la figure **??**, la MIB II **??** est référencée par l'identifiant `1.3.6.1.2.1` .

La MIB est une base de donnée très vaste qui permet de stocker divers catégories d'informations. On y trouve notamment le groupe `mib-2`, utile à l'administration réseau standard, mais aussi le groupe `private` qui permet aux constructeurs d'éléments administrables de stocker des informations propriétaires.

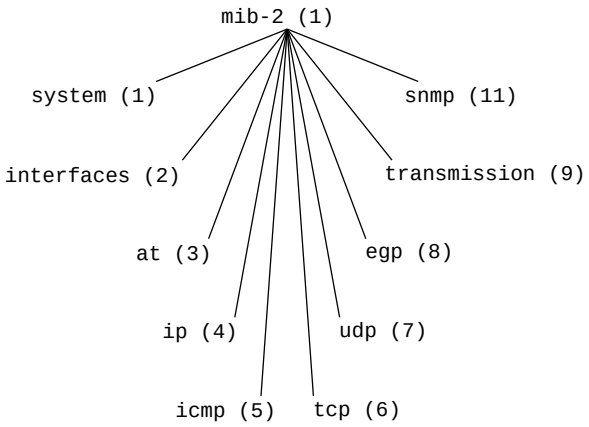


FIG. 7.8 – Les principales branches de la MIB II.

7.7.1 La MIB II

Elle stocke toutes les informations génériques inhérentes à la gestion de réseau [?]. Ses principaux groupes sont les suivants :

1. le groupe `system` : fournit des informations générales relatives à l'élément administré.
2. le groupe `interfaces` : fournit des informations sur les interfaces physiques implantées dans l'équipement.
3. le groupe `at` : contient une simple table qui associe à chaque adresse physique d'une interface implantée une adresse réseau IP.
4. le groupe `ip` : fournit des informations concernant le protocole IP au noeud considéré.
5. le groupe `icmp` : fournit des informations sur ICMP, un protocole de messages concernant le niveau IP.
6. le groupe `tcp` : contient des informations concernant l'implémentation et le fonctionnement de TCP.
7. le groupe `udp` : contient des informations concernant l'implémentation et le fonctionnement de UDP.
8. le groupe `egp` : contient des informations concernant l'implémentation et le fonctionnement de EGP (*External Gateway Protocol*).
9. le groupe `transmission` : fournit des informations relatives au médium de transmission.
10. le groupe `snmp` : contient des informations sur le protocole d'administration.

7.8 Le protocole SNMP

Le protocole SNMP (*Simple Network Management Protocol*) [?] permet aux gestionnaires d'accéder aux objets gérés dans la MIB des différents agents. Il utilise UDP (port n°161) comme protocole de transport. Son PDU comporte trois champs :

- `version` : identifie la version du protocole.
- `communauté` : identifiant du gestionnaire qui permet de spécifier les droits d'accès sur les objets gérés.

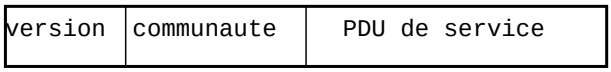


FIG. 7.9 – Le PDU SNMPv1.

- `service` : c'est le corps du PDU. Il contient les données du service dont le PDU SNMP fait l'objet.

7.8.1 Les différents services

Dans sa première version, les services SNMP sont des requêtes de lecture ou d'écriture dans un objet géré, ou la signalisation de l'occurrence d'un évènement. La version 1 de SNMP compte les quatre services suivants :

- GET : permet de lire un ou plusieurs objets gérés.
- GET-NEXT : permet de lire un ou plusieurs objets gérés situés "après" (d'après l'ordre lexicographique établi par la MIB) d'autres objets gérés.
- SET : permet de modifier la valeur d'un ou plusieurs objets gérés.
- TRAP : permet à un agent de signaler l'occurrence d'un évènement particulier ayant survenu dans son environnement.

Bibliographie

[AAU91] A. Aho (R. Sethi) et Ullman (J.). – *Compilateurs : Principes, techniques et outils.* – InterEditions, 1991.

[BBGS01] Bernet (Y.), Blake (S.), Grossman (D.) et Smith (A.), *An Informal Management Model for Diffserv Routers*, draft-ietf-diffserv-model-06, March 2001.

[BCS01] Baker (F.), Chan (K.) et Smith (A.), *Management Information Base for the Differentiated Services Architecture*, draft-ietf-diffserv-mib-16, November 2001.

[BZB+97] Braden (R.), Zhang (L.), Berson (S.), Herzog (S.) et Jamin (S.), *Resource ReSerVation Protocol (RSVP)*, RFC2205, September 1997.

[CFSD90] Case (J.), Fedor (M.), Schoffstall (M.) et Davin (J.), *A Simple Network Management Protocol (SNMP)*, RFC1157, May 1990.

[CO97] Crocker (D.) et Overell (P.), *Augmented BNF for Syntax Specifications : ABNF*, RFC2234, November 1997.

[CSD+01] Chan (K.), Seligon (J.), Durham (D.), Gai (S.), McCloghrie (K.), Herzog (S.), Reichmeyer (F.), Yavatkar (R.) et Smith (A.), *COPS Usage for Policy Provisionning (COPS-PR)*, RFC3084, March 2001.

[DBC+00] Durham (D.), Boyle (J.), Cohen (R.), Herzog (S.), Rajan (R.) et Sastry (A.), *The COPS (Common Open Policy Service) Protocol*, RFC2748, January 2000.

[DF01] D'Alu (S.) et Festor (O.). – FLAME : une plate-forme active dédiée à la supervision des services de l'Internet. *CFIP*, 2001.

[EHJ+01] Elliott (C.), Harrington (D.), Jason (J.), Schoenwaelder (J.), Strauss (F.) et Weiss (W.), *SMIng Objectives*, RFC3216, December 2001.

[FMS+02a] Fine (M.), McCloghrie (K.), Seligson (J.), Chan (K.), Hahn (S.), Bell (C.), Smith (A.) et Reichmeyer (F.), *Differentiated Services Quality of Services Policy Information Base*, draft-ietf-diffserv-pib-06, March 2002.

[FMS+02b] Fine (M.), McCloghrie (K.), Seligson (J.), Chan (K.), Hahn (S.), Sahita (R.), Smith (A.) et Reichmeyer (F.), *Framework Policy Information Base*, draft-ietf-rap-frameworkpib-07, January 2002.

[HS01] Hedge (H.) et Sahita (R.), *SMIng Mappings to COPS-PR*, draft-ietf-sming-copspr-01, July 2001.

[HT00] Horlait (E.) et Touhane (N.). – Qualité de service dans l'architecture TCP/IP. *Support de cours*, 2000.

[Kos01] Kosiur (D.). – *Understanding Policy-Based Networking.* – Wiley, 2001.

[Mar99] Martin (J.C.). – Policy Based Networks. *Sun BluePrints*, 1999.

[MESW00] Moore (B), Ellesson (E.), Strassner (J.) et Westerinen (A.), *Policy Core Information Model – Version 1 Specification*, RFC2753, January 2000.

[MFS+01] McCloghrie (K.), Fine (M.), Seligson (J.), Chan (K.), Hahn (S.), Sahita (R.), Smith (A.) et Reichmeyer (F.), *Structure of Policy Provisionning Information (SPPI)*, RFC3159, August 2001.

[Mic02] Microsystems (Sun). – Java Management Extension White Paper. *Sun Microsystems*, 2002.

[Nic] Nicklisch (J.). – A rule language for network policies. *C and C Network Product Development Laboratories.*

[NNEM00] N. (Damianou), N. (Dulay), E. (Lupu) et M. (Sloman). – Ponder : A language for Specifying Security and Management Policies for Distribued Systems. *Imperial College of Science, Technology and Medecine*, 2000.

[PB02] Polyrakis (A.) et Boutaba (R.). – The Meta-Policy Information Base. *IEEE Network*, vol. 16, n° 1, 2002, pp. 40–47.

[Pro02] Products (Webgain). – JavaCC documentation. *www.webgain.com/products/java_cc/documentation.html*, 2002.

[RM90] Rose (M.) et McCloghrie (K.), *Structure and Identification of Management Information for TCP/IP-based Internets*, RFC1155, May 1990.

[RM91] Rose (M.) et McCloghrie (K.), *Management Information Base for Network Management of TCP/IP-based internets : MIB-II*, RFC1213, March 1991.

[SS01a] Strauss (F.) et Schoenwaelder (J.), *Next Generation Structure of Management Information*, draft-ietf-sming-02, July 2001.

[SS01b] Strauss (F.) et Schoenwaelder (J.), *SMIng Mappings to SNMP*, draft-ietf-sming-snmp-02, July 2001.

[Sta97] Stalling (W.). – *SNMP, SNMPv2 and RMON.* – Addison Wesley, 1997.

[Ver02] Verma (Dinesh C.). – Simplifying Network Administration Using Policy-Based Management. *IEEE Network*, vol. 16, n° 1, 2002, pp. 20–26.

[YPG00] Yavatkar (R.), Pendarakis (D.) et Guerin (R.). – A Framework for Policy-based Admission Control, January 2000. RFC 2753, Informational.