République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur
Et de la Recherche Scientifique

Université de Constantine
Faculté des Sciences de l'Ingénieur
Département Informatique

N° d'ordre :
Série :

Thèse

Présenté Par
ZEMMOUCHI Fares Mounir

Pour l'Obtention du Diplôme de
Magister en Informatique

Conception d'un serveur de politiques (QoS) via le système MAUDE

Devant le jury composé de :

M. CHAOUI Allaoua	**Maître de Conférence**	**Président**
M. ZAROUR Nacereddine	**Maître de Conférence**	**Examinateur**
M. CHIKHI Salim	**Maître de Conférence**	**Examinateur**
Mme BELALA Faiza	**Maître de Conférence**	**Rapporteur**

Laboratoire: **LIRE, Université de Constantine.**
Équipe: **Génie Logiciel et Systèmes Distribués**

Remerciements

Je tiens à remercier toutes les personnes qui ont permis que cette année se passe de la meilleure façon qui soit. Plus précisément tous mes remerciements à Faiza BELALA responsable du bon déroulement de mon magister pour son encadrement efficace, ses idées inépuisables, sa disponibilité…. Et la patience dont elle a du faire preuve.

Je remercie aussi toutes les personnes qui de près ou de loin m'ont suivi pendant mon travail, notamment toute l'équipe LIRE, dans laquelle l'ambiance a toujours été au beau fixe, à ma famille.

TABLE DES MATIERES

CHAPITRE 1

1. INTRODUCTION GENERALE

1.1. Introduction

Pendant longtemps, les réseaux de communication étaient spécifiques à un seul type d'information et un seul type de service : les réseaux à commutation de circuit pour la téléphonie, les réseaux hertziens pour la télévision, les réseaux IP pour les données... Ces différents types de réseaux ont aujourd'hui tendance à converger, chaque opérateur voulant offrir de multiples services à ses clients : accès à Internet, téléphonie, télévision interactive, vidéo à la demande, visioconférence, télémédecine... Nous observons aussi l'émergence de nouvelles applications complètement distribuées permettant d'utiliser des ressources réparties sur un réseau ou sur internet : la téléphonie *peer-to-peer* (P2P) [Yu, 2003], le calcul distribué (*grid computing, P2P computing*), ou même la diffusion *peer-to-peer* de flux vidéo continus (voir par exemple [Cui et al., 2004]).

1.2. Contexte de travail

Les réseaux actuels sont désormais multiservices et hétérogènes. Outre cette multiplication des services dans les réseaux, ceux-ci sont de plus en plus exigeants quant à la qualité des transmissions. La notion de qualité de service permet de formaliser ces exigences en terme de critères de performance : bande passante, délai de transmission de bout en bout, taux de perte des paquets, gigue... et chaque service peut avoir des exigences différentes en terme de qualité de service. L'architecture *best-effort* établi par les réseaux IP dans les années 70 ne permet pas de garantir une quelconque qualité de service et il a été nécessaire de définir de nouvelles architectures de réseaux pour répondre à ces nouveaux besoins. De cette nécessité sont nées les architectures à intégration de service (réseaux ATM, modèle IntServ pour les réseaux IP) qui s'appuient sur une réservation préalable des ressources et un multiplexage statistique des trafics, et plus récemment les architectures à différentiation de services (modèle DiffServ, réseaux MPLS) qui effectuent un traitement différencié des trafics, regroupés en quelques classes de services, pour garantir la qualité de service. Toutefois ces architectures ne permettent pas de résoudre tous les problèmes et la maîtrise de la qualité de service reste aujourd'hui un défi majeur pour la recherche sur les réseaux multiservices.

L'introduction de mécanismes «*intelligents*» dans les réseaux multiservices permet de surmonter la difficulté de mettre en place des méthodes plus traditionnelles pour prendre en compte toute la complexité générée par la multiplication des services. Cela permet aussi de répondre à un autre besoin de plus en plus pressant : les réseaux doivent être de plus en plus autonomes et doivent s'adapter automatiquement aux conditions de trafic, à l'ajout de nouveaux services, aux congestions, aux pannes, etc. Ils doivent devenir «intelligents» (*smart networks*) ou «conscients d'eux-mêmes» « *self-aware networks »* . [Gelenbe et Núñez, 2003]). Les méthodes d'apprentissage

notamment permettent de répondre en partie à ces nouveaux besoins et la littérature scientifique en proposent de nombreuses applications.

C'est dans ce contexte général que se situent ces travaux de thèse. Nous nous intéressons au problème de l'évaluation de performance dans les réseaux, et plus spécifiquement l'évaluation des critères de qualité de service. Il s'agit d'un problème transversal aux différents mécanismes de gestion d'un réseau : pour être capable de garantir la qualité de service de chaque client, il faut en effet être capable d'en évaluer les différents critères. Traditionnellement, ce problème est résolu de deux manières : soit de façon analytique à l'aide de la théorie des files d'attente, soit de façon expérimentale par simulation (simulation par événements discrets ou simulation fluide). Ces deux approches sont complémentaires : la théorie des files d'attente permet d'obtenir des expressions analytiques (parfois exactes) de critères de performances sous des hypothèses relativement fortes et pour certains systèmes d'attente, tandis que la simulation a un pouvoir d'expression beaucoup plus important (*a priori* sans limite) mais se heurte à des temps de calcul prohibitifs.

Dans un travail antérieur [ZEMMOUCHI, 2003], nous avons abordé la QoS (qualité de service) par un chemin algorithmique pour établir de manière ad-hoc un lien entre les politiques de qualité de service et les annuaires LDAP comme structure de stockage des règles..

Nous proposons dans cette thèse un chemin formel qui conduit à préciser la nature contractuelle de la QoS dans les serveurs de politique.

1.3. Problématique et objectifs

La qualité de service au sein d'un réseau IP revient à regrouper les paquets d'un flux d'applications dans une certaine classe de trafic. Cette classification a pour objet d'accorder à ces paquets un traitement plus ou moins prioritaire par rapport à d'autres paquets. Ainsi, certains utilisateurs recevront un meilleur service que d'autres. Il est alors inévitable que des utilisateurs rechercheront à obtenir le meilleur service sans en payer le prix ou sans y être autorisé. On conçoit qu'il est nécessaire de disposer de certains mécanismes sur le réseau, afin de mettre en œuvre des règles de gestion de trafic, faisant appel à une architecture, à un certain nombre de protocoles et à des modèles de représentation des données sous forme d'objet. La complexité croissante d'un tel problème et le manque de la science proportionnée et technologie pour soutenir le développement robuste mène typiquement à des solutions qui sont fragiles, incertaines, et extrêmement difficiles d'évoluer. Tout ça impose la recherche des méthodes de spécification et vérification puissantes.

Dans ce contexte, l'introduction d'un cadre formel pour développer un serveur de politiques est d'un grand intérêt, il permet d'avoir une expression claire des propriétés du système à modéliser, ainsi qu'une vision cohérente et transparente de ce que le système fait et ne fait pas. Cette

formalisation est ensuite utilisée, pour donner une sémantique formelle à ce type de système en termes de logique de réécriture [Mes, 2002], un formalisme rigoureux de spécification, de prototypage et de vérification des systèmes.

L'objectif principal de ce travail est d'attribuer une sémantique formelle et opérationnelle, basée logique de réécriture, à un serveur de politiques gérant les qualités de service, permettant ainsi une compréhension approfondie du serveur par la mise en évidence des ambiguïtés laissées dans l'ombre par les spécifications informelles, ainsi que l'utilisation d'outils de preuve et d'évaluation symbolique existants autour de cette logique.

La logique en général est une méthode de raisonnement correct pour quelques classes de systèmes. Pour la logique équationnelle, les modèles des systèmes sont les ensembles, les fonctions entre ces ensembles et la relation d'identité entre les éléments. Pour la logique de réécriture, les entités sont les systèmes concurrents ayant des états, et qui évoluent à travers des transitions. Elle est considérée également comme une logique de changement concurrent qui traite l'état et les calculs concurrents.

Une politique de QoS est un ensemble de règles associées à certains services. Les règles définissent les critères à satisfaire pour obtenir ces services. Elles sont définies en fonction de conditions et d'actions, les actions découlant du respect des conditions. Dans la pratique, une règle pourra elle-même contenir d'autres règles. Ce principe de hiérarchie de règle permet de construire des politiques complexes à partir de règles simples. Le mode de définition de ces conditions et de ces actions doit être établi d'une façon globale, de sorte à maintenir une cohérence entre les différents domaines de gestion de politiques de QoS. En d'autres termes, un flux défini comme prioritaire dans un domaine de gestion (le réseau interne de l'entreprise) doit rester prioritaire s'il traverse un réseau d'opérateurs qui correspond à un autre domaine de gestion. Pour assurer cette portabilité des règles, il est nécessaire que la définition de celle-ci soit réalisée dans un langage de description de haut niveau, indépendant des équipements. Dans ce cas, nous exprimons alors ces règles (politiques) dans la logique de réécriture qui représente un cadre sémantique unificateur de plusieurs modèles de spécification.

Une politique peut se définir à différents niveaux. Le niveau le plus haut correspond à celui de l'utilisateur. La détermination d'une politique s'effectue par un dialogue entre l'utilisateur et l'opérateur. Ils utilisent pour cette négociation soit le langage naturel soit des règles déjà préparées par l'opérateur du réseau. Dans ce dernier cas, l'utilisateur ne peut sélectionner que des règles proposées par le réseau. Dans les deux cas, cette politique doit être traduite dans un langage de niveau réseau permettant de déterminer le protocole réseau de gestion de la qualité de service et son paramétrage. Ensuite, il faut traduire ce langage de niveau réseau en langage de bas niveau

correspondant à la programmation des nœuds du réseau, ce que l'on peut appeler la configuration du nœud.

Nous enrichissons, à travers l'approche proposée, le langage de niveau réseau par quelques structures basées logique de réécriture afin de décrire de façon non ambiguë les modèles de représentation des données considérés (sous forme d'objet) et les protocoles qui y sont autour.

1.4. Plan

Ce document est organisé en cinq chapitres. Le premier chapitre, porte sur les concepts clés de la qualité de service en général et celle qui se rapporte au réseau IP en particulier. Un intérêt particulier est donné à la présentation des modèles IntServ et DiffServ et leurs limites.

Dans le deuxième chapitre, des notions précises et claires concernant les politiques de services et les serveurs qui les gèrent sont données. On commence par donner la structure et l'architecture générique d'une politique de service, ensuite nous présentant un aperçu général sur les langages de spécification de politiques (ponder en particulier) et leurs intérêts en pratique. Enfin, on montre la solution souvent préconisée dans la spécification et la gestion des politiques de service et qui se base sur l'annuaire LDAP.

Le troisième chapitre détaille les concepts clés de LDAP : protocole, modèle de données, modèle fonctionnel et comment déployer un service d'annuaire LDAP, après avoir motivé convenablement cette approche dans la gestion des politiques de QoS.

Le quatrième chapitre est consacré à la présentation du formalisme logique de réécriture ainsi que la syntaxe du langage Maude basé sur cette logique. Un intérêt particulier est donné à la déclaration des modules orientés objets en Maude qui seront utilisés dans cette modélisation.

Dans le dernier chapitre, un nouveau support à base de logique de réécriture est proposé pour concevoir un serveur de politique QoS. L'approche de conception est inspirée de celle basé LDAP. Dans la première étape, un modèle d'information pour les politiques de QoS de type QPIM est défini en utilisant principalement les modèles orienté objet de Maude. Dans une deuxième étape, nous proposant un schéma fonctionnel formel d'un serveur de gestion de politiques de QoS en exploitant les caractéristiques syntaxiques et sémantique de Maude.

Une conséquence théorique importante se découle de cette étude concernant :

- Le prototypage et éventuellement la vérification de certaines propriétés dans Maude.
- La formulation correcte des métas politiques.
- La structuration des politiques de QoS selon leur sémantique et régler ainsi le problème de conflit sémantique qui peut se poser dans certaine situations.

Le manuscrit se termine par une conclusion générale permettant de situer la présente contribution par rapport aux travaux réalisés dans le même contexte et d'évoquer les différentes continuations jugées possible pour ce travail.

CHAPITRE 2

2. LA QUALITE DE SERVICE DANS L'INTERNET

2.1. Introduction

La "qualité de service" est une notion relativement controversée, même si les polémiques se sont calmées au cours de ces dernières années. Ce chapitre décrit les approches et les axes discutés pour introduire de nouveaux types de données comme les flux multimédia, la téléphonie, les simulations distribuées.

L'Internet a été conçu comme un réseau d'interconnexion entre des universités et des instituts de recherche aux Etats Unis. Le protocole IP "*Internet Protocol*" a permis un développement très rapide du réseau grâce à sa flexibilité et sa nature ouverte. IP s'est également imposé comme technologie réseau dans la plupart des réseaux privés. Pourtant, les paradigmes qui ont fait le succès de l'Internet imposent de fortes limitations aujourd'hui. Quand le protocole IP a été conçu, l'architecture réseau comptait peu de terminaux connectés, et les applications usuelles étaient telnet, l'email et le FTP. Avec la naissance du Web, l'âge de l'Internet commercial fut inauguré, donnant naissance à de nouveaux types d'utilisateurs, toujours plus avides de services et donc de bande passante. La consommation des utilisateurs a donc évolué aussi rapidement que l'Internet, l'augmentation des débits étant très vite rattrapée par une consommation accrue de nouvelles applications. L'apparition d'ordinateurs personnels plus puissants à des prix plus abordables a conduit à un formidable essor des développements d'applicatifs. Le paradigme "*IP over everything*" des premières années (focalisation sur le transport) a donc changé pour un paradigme de "*everything over IP*" d'aujourd'hui (focalisation sur les services au dessus d'IP). L'Internet est présent presque partout, générant des économies d'échelle et favorisant le développement d'un grand nombre d'applications, ainsi que l'adaptation des applications existantes sur IP. Toutes les applications imaginables, de la navigation sur le Web à la diffusion vidéo ou la téléphonie seront transportées par le même réseau.

L'apparition d'un tel réseau multiservice permettra de sur croie une utilisation optimale des ressources avec des coûts d'opération très réduits. Pour atteindre cet objectif, il s'avère néanmoins nécessaire de pouvoir différentier les applications entre elles de façon à leur offrir un traitement spécifique, conforme à leurs besoins et contraintes (de temps réel par exemple). Sous sa forme originelle, le protocole IP traite tous les éléments transportés de façon égale. La conséquence est qu'à certains moments, des applications n'auront pas les ressources minimales dont elles ont besoin, en revanche d'autres applications disposeront de ressources qu'elles n'utiliseront pas pleinement. Il est important de mentionner que les flux téléphonies requièrent de petites quantités de bande passante, mais nécessitent des délais de transport faibles et déterministes. De tels flux ne pourront donc pas être efficacement transportés dans un réseau IP sans mécanismes de QoS (*Quality of Service*) supplémentaires.

2.2. Gestion des flux

2.2.1. Intégration de service

Les travaux sur l'intégration de service dans l'Internet ont été pris en compte par trois groupes de travail :

• le groupe de travail RSVP "*ReSerVation Protocol*" [RFC 2205] porte mal son nom, car il a défini un protocole conçu pour transporter des messages de caractérisation (appelé aussi signalisation) de flux, il n'effectue aucune réservation ;

• le groupe IntServ "*Integrated Services*" définit les services qui peuvent être offerts. Cela passe par la description des caractéristiques des flux de données et les paramètres à indiquer lors d'une demande de réservation. Ces caractéristiques sont prévues pour être transportées dans les données des paquets RSVP ;

• le groupe de travail issll "*Integrated Services over a Specific Link Layer*" définit les méthodes pour traduire les caractéristiques de flux spécifiés par l'IntServ vers un niveau 2 particulier

Quand une source produit un flux de données, elle peut également émettre des messages de signalisation RSVP, décrivant les caractéristiques de celui-ci. Cette signalisation ayant la même destination que le flux (il peut aussi s'agir d'une adresse de multicast), traversera les mêmes routeurs intermédiaires qui y ajouteront leurs caractéristiques, principalement leur temps de traversée.
Les flux restent traités par les routeurs en"*Best-Effort*".Le destinataire recevant les messages de signalisation, en plus des données du flux, peut décider d'améliorer la qualité de celui-ci en envoyant un message de réservation vers la source pour demander aux routeurs d'améliorer le traitement du flux. Mais le routage dans l'Internet n'est pas symétrique, il faut donc que les routeurs se souviennent pour chaque flux quel était le précédent routeur. Un contexte doit être établi dans chaque routeur pour chaque flux signalé par la source. A la réception d'un message de réservation, un contrôle d'admission est fait par le routeur pour déterminer si l'ajout d'un nouveau flux ne perturbera pas ceux pour qui une réservation a déjà été faite.
Le groupe de travail IntServ a actuellement défini deux types de services lors de la réservation :

• Le service garanti [RFC 2212] est basé sur les résultats du "Network Calculus" [Le Boudec, 1997] qui permet de trouver une borne maximale pour le temps de transmission d'un paquet et la taille maximale des mémoires tampons nécessaires dans les équipements pour éviter les pertes de paquets. Le destinataire en fixant le débit minimal que doit offrir le réseau à ce flux influe sur le

temps maximal de transmission des paquets. Ainsi, pour réduire les temps de traversée, il est possible de réserver à un débit nettement supérieur à celui de la source.

• Le service contrôlé [RFC 2211] a une définition volontairement relativement vague : un service contrôlé offre un service proche de l'Internet peu chargé. Ce type de service a un intérêt si l'on considère que des outils de téléconférence comme "vat" pour la vidéo ou "vic" pour l'audio offrent une bonne qualité quand le réseau est peu utilisé, mais la qualité se dégrade très vite si le réseau est saturé.

Depuis que les travaux ont été finalisés, l'impact de ce protocole a surtout été politique. Il a conduit à prendre en compte le protocole IP dans ces travaux, en particulier ceux concernant la téléphonie sur l'Internet et à recommander son utilisation dans la norme H.323. Par contre aucune réalisation à grande échelle n'a été faite de ce protocole. Ceci pour plusieurs raisons :

• Le protocole ne résiste pas au facteur d'échelle, il faut mettre un état par flux signalé dans chaque routeur. Ceci n'est pas réalisable dans un réseau d'opérateur ou sur une liaison transatlantique.

Des propositions pour agréger les différentes réservations au sein d'un réseau d'opérateur ont été faites, mais elle implique qu'il faille aussi "désagréger" dans le réseau [Baker,Davie, 2000]. La vision qu'a un routeur est limitée au prochain saut, c'est-à-dire l'information contenue dans la table de routage. Ce n'est pas parce que plusieurs flux ont le même prochain routeur qu'il vont avoir le même routeur de sortie du réseau de l'opérateur. Par conséquent quelque part dans le réseau, ces flux seront séparés, mais l'agrégation de la signalisation aura fait perdre leurs caractéristiques individuelles.

• Les bornes offertes par la classe de service garanti sont trop pessimistes, ce qui limite le nombre de flux pouvant obtenir une réservation.

• Il n'existe pas de modèle économique et technique pour facturer les ressources réservées dans le réseau et les utilisateurs doivent être capable de comprendre la relation entre le coût et la qualité des réservations.

Il s'en suit que si la réservation de ressource est utilisée dans l'Internet, elle le sera uniquement au niveau d'un système autonome et pas de bout-en-bout. Les problèmes de facturation sont grandement simplifiés. Puisqu'il sera possible d'agréger des flux ayant un même routeur de sortie chez un opérateur, de limiter le nombre de flux à signaler (donc de réduire le nombre de contextes dans les routeurs) et d'offrir une plus grande pérennité au flux. Dans les sites terminaux, l'usage de la réservation est plus contestable car il est beaucoup plus facile d'augmenter la capacité du réseau pour se trouver dans une situation de "gaspillage" de bande passante ou d'introduire des priorités sur les flux que d'administrer un réseau mettant en œuvre de la réservation de ressources. Les seuls cas

où cette augmentation de capacité est difficilement réalisable sont les réseaux sans fils et les réseaux Intranet utilisant des liens longues distances.

2.2.2. Différenciation de services

La différenciation de services [RFC 2475] consiste dans une situation de congestion à reporter les pertes de paquets sur certaines classes de trafic, pour en protéger d'autres. Elle n'offre donc aucune garantie sur les flux, car il n'existe à aucun moment un contrôle d'admission dynamique permettant d'éviter une congestion. Le contrôle d'admission est fait a priori par la définition d'un contrat pour chaque classe de trafic et par le dimensionnement des ressources pour pouvoir garantir ce contrat. La différenciation de service présente les avantages suivants :

• La signalisation est faite dans chaque paquet (IPv4 ou IPv6) en attribuant une signification différente aux bits du champ type de service [RFC 2474]. Il n'est plus besoin de garder dans le routeur un contexte liant le flux de signalisation au flux de données. Cela permet aussi une agrégation naturelle des flux, ainsi pour un opérateur, les paquets qu'il reçoit marqués pour une certaine classe peuvent appartenir à plusieurs sources.

• La complexité du traitement est concentrée dans les routeurs aux frontières du réseau. Ils effectuent les opérations " complexes " de contrôle de la validité du contrat pour les différentes classes de trafic. Dans le cœur du réseau, le traitement est plus simple, ce qui autorise un relayage rapide des paquets.

• La tarification du service est plus simple, il suffit, comme avec "Frame Relay", de définir les ¨Paramètres de contrôle de classes de services. Les travaux sur l'architecture à différenciation de services ont surtout porté sur la définition d'une architecture la plus générale possible, permettant aux opérateurs de développer facilement des classes de services adaptées aux besoins de leurs clients. Néanmoins, deux classes de services sont en cours de définition :

• La classe "*Expedited Forwarding*" [RFC 2598] offre un service de liaison virtuelle à travers le réseau d'un opérateur. Le contrat porte sur un débit constant. Les paquets excédentaires sont lissés ou rejeté à l'entrée pour toujours rester conforme au contrat. L'opérateur s'engage à traiter ce trafic prioritairement. Van Jacobson propose d'utiliser une priorité stricte pour traiter ces flux. Pour qu'un tel service soit performant, il faut qu'il ne représente qu'une faible partie du trafic total pour qu'aucun paquet marqué EF ne soit rejeté dans le cœur du réseau. Le service pouvant être l'interconnexion de sites via un service de réseaux privés virtuels mis en place par un opérateur.

• Les classes "*Assured Fowarding*" [RFC 2597] définissent 3 priorités (notées par une couleur vert, orange, rouge) définissant l'ordre de rejet dans un routeur en cas de congestions. Les paquets rouges ont une probabilité de rejet plus important que les paquets oranges. La couleur dépend de la

conformité de la source avec son contrat. Le marqueur actuellement préconisé pour déterminer la couleur du paquet est basé sur "tokens buckets" [RFC 2698]: si le trafic est conforme aux deux, les paquets sont marqué en vert, s'il n'est conforme qu'à un des "tokens buckets", les paquets seront marqués en orange et s'il n'est conforme à aucun des "tokens buckets" il est marqué en rouge. Dans le cœur du réseau, les mécanismes de rejet sont basés sur RIO [Clark, Fang, 1998], une extension à plusieurs priorités de RED. Chaque couleur dispose d'un seuil et d'une probabilité de rejet différente. Quatre classes AF (notées AF1, AF2, AF3 et AF4) sont disponibles. Toujours pour être le plus général possible, l'IETF ne définit pas de priorité parmi ces classes. On peut donc imaginer attribuer les classes en fonction par exemple d'un ratio temps de traversée des paquets taux de perte qui pourrait être utilisé pour différencier, par exemple, les services multimédias interactifs, de streaming ainsi que les données. Une autre solution pourrait être d'affecter une classe pour le trafic UDP et une autre pour le trafic TCP ce qui permet de ne pas détruire l'équité entre les flux TCP.

Contrairement aux travaux du groupe IntServ qui partaient de la base théorique du "Network Calculus", ceux du groupe DiffServ, sont surtout basés sur une approche d'ingénierie portant sur la définition de l'architecture et sur la gestion de l'octet DS, mais les travaux effectués sur les classes de service d'ATM devraient pouvoir être repris et facilement applicables dans le contexte Internet. Il faut pouvoir arriver à déterminer les paramètres de configuration des équipements.

Les classes de service DiffServ ont été présentées dans l'optique d'un opérateur unique. Or le trafic peut traverser plusieurs systèmes autonomes. Il faut donc un critère de comparaison entre les classes de services. L'exemple le plus marquant est le concept de "*Bandwidth Broker*", vaguement introduit par Van Jacobson pour la classe de service EF. Le "*Bandwidth Broker*" a pour fonction de négocier les paramètres de la classe EF avec les autres systèmes autonomes. La difficulté réside, comme pour l'agrégation des flux de réservation dans IntServ, en la prise en compte des routes prises par ceux-ci lors des négociations. Or l'exemple pris par Van Jacobson ne prend en compte qu'une succession linéaire de domaine. Une des possibilités pour intégrer la route prise par les paquets serait de travailler dans les bases de données d'état des liens du protocole de routage interne (OSPF, IS-IS) qui contiennent une vision beaucoup plus précise (quoique toujours incomplète) de l'état du réseau.

Enfin, une méthode importante aussi bien dans la phase de validation des services que dans celle d'exploitation, sera l'utilisation de critères objectifs pour caractériser les performances des classes de services. Le groupe de travail IPPM"*IP Performance Measurement*" [RFC 2330] de l'IETF a défini un certain nombre de critères pour mesurer en particulier le temps d'aller simple d'un paquet dans le réseau. Pour augmenter la précision des mesures, des horloges synchronisées avec les systèmes GPS permettent une synchronisation précise de tous les sites. Quand le service de différenciation sera mis en place les clients pourront aussi se servir de ces mécanismes pour valider le réseau de leur opérateur.

2.2.3. Intégration IntServ / DiffServ

L'intégration de ces deux mécanismes est à l'étude. Plusieurs propositions ont été soumises. La première solution consiste à ne mettre l'intégration de service que dans les sites terminaux. Le cœur du réseau ne traite pas les messages de signalisation, mais les transmet comme des paquets normaux qui sont à nouveau interprétés dans le site destinataire. Un contrôle d'admission en bordure du réseau DiffServ permet de déterminer si le flux peut entrer dans la classe de service.
L'autre possibilité consiste à considérer le réseau DiffServ avec la classe EF comme un élément de réseau [Zhang, Yavatkar, 2000] et le caractériser pour permettre de construire un service garanti.

2.2.4. Ingénierie de trafic

L'ingénierie de trafic peut être un autre moyen d'augmenter la qualité du service, en dimensionnant plus finement les ressources dans le réseau de l'opérateur. Elle peut être liée au routage par qualité de service ou à l'utilisation de la différenciation de services, par exemple pour protéger certains flux d'un trop grand nombre de pertes. Il s'agit alors d'adapter le comportement du réseau à chaque classe de service en modifiant le routage et/ou le traitement dans les équipements intermédiaires.
L'adaptation du routage à chaque qualité de service peut se faire de différentes manières, la plus évidente étant sans doute d'établir dans tous les routeurs du réseau une table de routage différente pour chaque qualité de service. Bien qu'OSPF en prévoyait la possibilité, cette solution n'a pas été employée parce qu'elle est coûteuse en mémoire et ne permet pas aux opérateurs de maîtriser les routes prises par les paquets. En effet, celles-ci sont construites automatiquement par le protocole de routage qui se base sur une métrique associée à chaque lien. Si la manipulation des métriques permet de favoriser ou de défavoriser tel lien et donc d'influencer le chemin parcouru par les paquets, elle reste d'une utilisation complexe. Les opérateurs préfèrent la possibilité d'établir des chemins protégés (fixés à l'avance) dans le réseau et d'utiliser un chemin donné pour un trafic donné. Ce trafic peut correspondre au trafic d'un client ou d'un réseau privé virtuel (VPN : Virtual Private Network) particulier, ou appartenir à une classe de service donnée.
Il est possible d'utiliser de tels tunnels avec des technologies orientées circuits comme ATM ou Frame Relay, celles-ci permettant aussi d'attribuer des ressources réservées aux chemins (circuits). Malheureusement, non seulement l'administration de ces réseaux est complexe et peu automatisée,

mais l'utilisation d'un cœur de réseau ATM ou "Frame Relay"directement au-dessous d'IP suppose un maillage logique (à base de circuits) complet des routeurs de bordure pour chaque qualité de service. Chaque routeur d'entrée choisit en fonction du paquet et de la qualité de service qui lui est associée le circuit conduisant au routeur de sortie avec la qualité de traitement correspondante.

Aujourd'hui MPLS "*Multi Protocol Label Switching*" [Callon, Swallow, 1999] permet de simplifier l'administration d'un tel coeur de réseau en ajoutant de nouvelles fonctionnalités particulièrement intéressantes pour la gestion de la qualité de service. Dans le même esprit que l'architecture DiffServ, MPLS permet de réduire le coût des traitements associés au relayage des paquets en les reportant à la périphérie du réseau et en en réduisant la fréquence. Il apporte aussi un mécanisme de routage hiérarchique efficace, c'est-à-dire des tunnels permettant de gérer les réseaux privés virtuels (VPN).

Le principe de MPLS est d'attribuer un label (une étiquette) à chaque paquet lorsqu'il entre dans le réseau. Ce label est attribué en fonction de la classe de relayage (FEC : Forwarding Equivalent Classe) à laquelle appartient le paquet. La définition de ces classes dépend de l'opérateur du réseau, généralement une classe correspond à une entrée de la table de routage ou à un routeur de sortie du réseau, mais elle peut aussi prendre en compte la classe de service DiffServ. Le routeur décide de la FEC à laquelle appartient un paquet en fonction des informations contenues dans son en-tête (adresse destination, classe de service DiffServ, appartenance à un VPN, ...) et éventuellement de la connaissance qu'il a de la topologie du réseau. Une fois à l'intérieur du réseau les paquets ne sont plus traités qu'en fonction du label qui leur a été associé et l'en-tête IP n'est plus consulté. Ce label décide donc dans chaque routeur : du prochain routeur, du comportement DiffServ et de l'utilisation éventuelle des ressources réservées.

2.3. Conclusion

On peut dire que ni IntServ ni DiffServ ne sont des modèles parfaits répondant à toutes les exigences d'une bonne qualité de service. En effet, IntServ est très performant et bien adapté aux réseaux de petite envergure, alors que DiffServ s'adapte mieux à des réseaux de grande échelle. Cependant, la mise en œuvre de l'ensemble des mécanismes décris est une tache très lourde, il est difficile de configurer manuellement l'ensemble des équipements réseau pour deux raisons essentielles :

- l'abondance des informations de QoS,
- la nature dynamique des configurations de QoS.

L'IETF a ouvert plusieurs directions pour l'amélioration de la qualité de service. Aucune de ces techniques n'est universelle, mais on assiste en ce moment à une convergence et une interaction entre ces différentes technologies. Elles donnent aux opérateurs des outils permettant une meilleure gestion des flux. Les techniques de réservation de ressources impliquent une certaine stabilité et une pérennité. Elles ne peuvent être déployées que dans le cœur du réseau.

Nous aborderons dans le chapitre suivant une approche de gestion du réseau pour l'amélioration de la qualité de service en utilisent la notion de politique. Cette approche devient une solution idéal pour les réseaux de grandes dimensions.

CHAPITRE 3

3. Les serveurs de politiques

3.1. Introduction

La qualité de service au sein d'un réseau IP revient à regrouper les paquets d'un flux d'applications dans une certaine classe de trafic. Cette classification a pour objet d'accorder à ces paquets un traitement plus ou moins prioritaire par rapport à d'autres paquets. Ainsi, certains utilisateurs recevront un meilleur service que d'autres. Il est alors inévitable que des utilisateurs rechercheront à obtenir le meilleur service sans en payer le prix ou sans y être autorisé. On conçoit qu'il est nécessaire de disposer de certains mécanismes sur le réseau, afin de mettre en œuvre des règles de gestion de trafic, faisant appel à une architecture, à un certain nombre de protocoles et à des modèles de représentation des données sous forme d'objet. Pour cela l'architecture de serveur de politique et la notion de politique deviens une approche innovatrice.

3.2. Définition d'une politique de QoS

Une politique de QoS est un ensemble de règles associées à certains services. Les règles définissent les critères à satisfaire pour obtenir ces services. Elles sont définies en fonction de conditions et d'actions, les actions découlant du respect des conditions. Dans la pratique, une règle pourra elle-même contenir d'autres règles. Ce principe de hiérarchie de règle permet de construire des politiques complexes à partir de règles simples. Le mode de définition de ces conditions et de ces actions doit être établi d'une façon globale, de sorte à maintenir une cohérence entre les différents domaines de gestion de politiques de QoS. En d'autres termes, un flux défini comme prioritaire dans un domaine de gestion (le réseau interne de l'entreprise) doit rester prioritaire s'il traverse un réseau d'opérateurs qui correspond à un autre domaine de gestion. Pour assurer cette portabilité des règles, il est nécessaire que la définition de celle-ci soit réalisée dans un langage de description de haut niveau, indépendant des équipements [Mélin 2001].

Une politique peut se définir à différents niveaux. Le niveau le plus haut correspond à celui de l'utilisateur. La détermination d'une politique s'effectue par un dialogue entre l'utilisateur et l'opérateur. Ils utilisent pour cette négociation soit le langage naturel soit des règles déjà préparées par l'opérateur du réseau. Dans ce dernier cas, l'utilisateur ne peut sélectionner que des règles proposées par le réseau. Dans les deux cas, cette politique doit être traduite dans un langage de niveau réseau permettant de déterminer le protocole réseau de gestion de la qualité de service et son paramétrage. Ensuite, il faut traduire ce langage de niveau réseau en langage de bas niveau correspondant à la programmation des nœuds du réseau, ce que l'on peut appeler la configuration du nœud [RFC 2753].

3.3. Structure et architecture d'une politique de QoS

Une architecture générique a été définie par l'IETF qui permet la mise en œuvre d'un contrôle d'admission des requêtes d'allocation de ressources fondé sur la mise en œuvre de politique (*policy-based admission control*). Le contrôle ou la gestion par politique implique plusieurs composants : les nœuds du réseau prennent le nom de PEP (*Policy Enforcement Point*). Les politiques y sont appliquées pour gérer le flux des utilisateurs. Le PDP (*Policy Decision Point*) est l'élément qui prend les décisions et choisit les politiques à appliquer aux PEP. Le système comporte également une console utilisateur qui regroupe des outils de gestion des politiques. Elle permet notamment de stocker, de mémoriser les politiques dans une base de données nommée *Policy Repository*, qui entrepose les règles de politique que le PDP pourra rechercher pour les appliquer aux nœuds du réseau [RFC 2753]. La figure 1 présente l'architecture proposée par l'IETF pour la prise en charge des politiques de QoS.

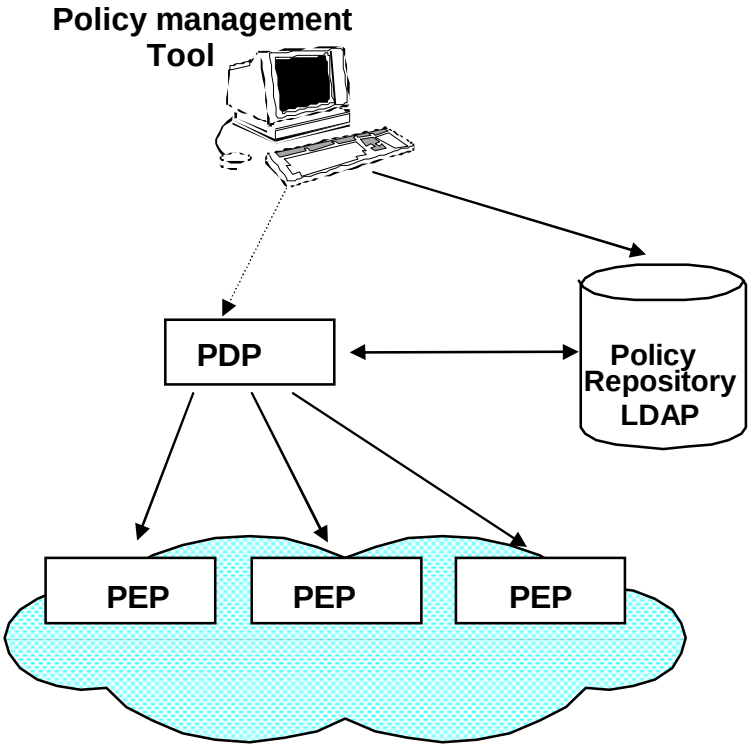


Figure 1 : Architecture générique

3.3.1. Le PDP (policy décision point)

Le PDP est défini comme une entité logique prenant des décisions politiques pour lui-même ou pour d'autres éléments du réseau qui demandent ses décisions. Le PDP ou serveur de politique, est donc le point central qui doit décider des politiques à appliquer dans le réseau. Le PDP est en quelque sorte un organe de décision qui recherche les informations dont il a besoin dans de nombreux serveurs qui communiquent directement avec lui de façon à prendre une décision. Ces serveurs peuvent être locaux ce qui est le cas le plus général, mais ils peuvent être distants.

Le principal serveur est le *policy repository* dans lequel le PDP vient rechercher les règles de politique, cette mémoire communique avec le PDP par le protocole LDAP (*Lightweight Directory Access Protocol*) ce qui explique son nom de serveur LDAP.

Il peut exister des serveurs complémentaire comme la PIB (*Policy Information Base*) qui garde en mémoire les modèles informationnelles qui peuvent être utilisés pour représenter une politique sous une syntaxe particulière. Le rôle du PDP consiste à identifier les règles de politiques qui sont applicable aux différents PEP et à déterminer les règles à appliquer stratégiquement à ces PEP.

Le PDP doit également se préoccuper de la traduction des règles de politique dans des formats compréhensibles des nœuds comme le format PIB. De plus il doit pouvoir communiquer avec d'autres serveurs internes pour ses décisions. Enfin, le PDP assure la distribution des configurations. En résumé, le PDP décide des règles à appliquer et envoie les ordres de configuration aux nœuds du réseau [RFC 2753].

3.3.2. Les PEP (Policy Enforcement Point)

Les PEP sont des entités logiques qui appliquent les décisions en terme de politique prises par le PDP dont elles dépendent. Les PEP sont généralement les nœuds du réseau, qui peuvent être de différents types (routeurs, commutateurs, etc.). Un PEP peut également être un firewall ou un équipement intermédiaire entre le client et le réseau. Un PEP doit être facilement accessible par PDP de sorte qu'il soit possible de le configurer sans problème [RFC 2753].

Il faut noter que la séparation du PEP, du PDP et de l'annuaire (*policy repository*) est logique et ne correspond pas forcément à une mise en œuvre physique. Ainsi, il est parfois possible que l'ensemble des composants soient situés sur un seul nœud du réseau. Dans d'autre cas, les nœuds du réseau peuvent comprendre un PEP et un PDP local, devant être complété par un PDP global. Dans ce cas, le PDP associé au PEP sera appelé LPDP (*local policy décision point*) et permettra d'appliquer une politique locale à l'équipement. Cependant, l'architecture prévoit, pour éviter des trous de sécurité, que LPDP se réfère à un PDP pour la décision finale quant à la politique à appliquer. L'architecture précise également qu'il est possible d'en avoir plusieurs (PDP/base de

données des politiques associées) pour fiabiliser la gestion du réseau et introduire un niveau de redondance. Ce nombre doit être limité afin de simplifier l'administration [Mélin 2001].

On considère que les trois fonctions suivantes doivent être associées à une politique :

- La fonction de décision : cette fonction réalisée par le PDP suppose la recherche d'une politique, son interprétation, la détection de conflits entre politiques, la réception de la description des interfaces des équipements, la réception des requêtes de PEP, la détermination de la politique à appliquer.
- La fonction d'application (*enforcement*) : cette fonction implique un PEP agissant selon les décisions du PDP, en fonction des politiques à appliquer et des conditions du réseau.
- La fonction de mesure (*metering*) : cette fonction permet de vérifier la mise en place de la politique et son respect.

Outre les composants décrits ci-dessus, cette architecture recourt à deux protocoles majeurs :

- COPS (*Common Open Policy Service*) : c'est le protocole sécurisé permettant les échanges entre les PEP et les PDP [RFC 2748].
- LDAP (*Lightweight Directory Access Protocol)* : c'est le protocole utilisé pour accéder à la base de données des règles [RFC 2251].

3.4. Les langages de politique (*Policy Language*)

La politique est quelque chose qui contrôle l'information, c'est-à-dire le trafic des utilisateurs. Mais pourquoi faut-il le contrôler ? Parce qu'il existe de contraintes à imposer dans le réseau, telles que :

- l'intégration des différents médias
- les ressources qui ne sont pas infinies
- les applications ont des besoins divers (délai, bande passante, sécurité, etc.)

Les politiques réseau ont pour but de gérer les éléments du réseau pour satisfaire les différentes contraintes de l'utilisateur. Différentes vues abstraites de la politique réseau ont été définis. Au niveau le plus haut (niveau réseau), se trouvent les SLA (*Service Level Agreement*) qui correspond à un contrat entre l'utilisateur et le fournisseur réseau. Un SLA contient une partie technique liée aux paramètres réseau appelée SLS (*Service Level Specification*) et SLO (*Service Level Objectives*) qui présente les objectifs qui se fixe l'opérateur. [RFC 3198].

Dans le niveau suivant (niveau routeur), se trouvent les règles de politiques. La gestion par politiques va automatiser la configuration du réseau. Différents types de politiques existent :

- de configuration,
- d'installation,
- de contrôle d'erreurs et événements, de sécurité.

Dans l'approche de gestion de la QoS à base de politique, la qualité de service est négociée sous forme de SLA. La gestion des ces SLA et de la QoS représente une nouvelle question qui doit être abordée. Il faut regarder la façon dont les SLA sont décrits ainsi que la gestion et la présentation des paramètres de QoS. Plus spécifiquement, des langages de politique de gestion de service ont été définis dans lesquels un administrateur peut publier et commander les ressources fondamentales du réseau. L'architecture proposée soutient également la conception d'applications de gestion à base de politique à l'aide d'un langage d'interprétation de politique, d'une interface graphique, et d'un dépôt de politique.

La vérification et la validation des politiques ont suscité beaucoup d'intérêt dans la communauté de l'informatique. Cette problématique à motivé le développement d'une panoplie de langages, nous passeront en revue les langage de politique qui existe et nous mettront l'accent sur un en particulier qui et le langage de spécification de politique Ponder.

.

3.4.1. Langage PFDL (Policy Framework Definition Language)

Le PFDL est un langage de l'outil de l'administrateur de réseau pour exprimer divers genres de politiques de réseau. Ceci peut inclure des politiques de sécurité, politiques de qualité de service, ce langage est issu du modèle PCIM. PFDL voit une politique comme ensemble de règles pour un domaine donné. Alternativement, une règle définit un ordre des actions qui peuvent être déclenchées en raison de quelques conditions. Des conditions sont faites d'un type et d'une valeur. Leur exécution peut être séquentielle, conditionnelle, aléatoire (mode de défaut) ou basée sur des résultats des règles semblables [Strassner 1998].

3.4.2. Langage ProNet

C'est un langage qui à une syntaxe simple et qui a été adopté afin de représenter des politiques et des règles. Il est basé sur deux concepts principaux, à savoir, les politiques et les événements. Un programme écrit dans ProNet se compose d'un ensemble constructions d'où chacune de ces derniers représente une définition, un instanciation, une liste et/ou un déplacement d'une politique ou d'un événement, ou une invocation de service [Fidalgo 2002].

3.4.3. Langage Ponder

Ponder [Damianou, Dulay, 2000] [Damianou, 2002] est un langage de spécification de politiques de contrôle d'accès pour les systèmes distribués. Ce langage se base sur les notions suivantes

- Le Sujet (Subject) fait référence aux utilisateurs, programmes ou toute autre entité qui peut accéder à une ressource ou un service.
- La Cible (Target) fait référence aux ressources et aux services partagés dans un réseaux de communication ou dans un système. Les cibles sont accessibles par les sujets selon les politiques de Ponder.
- Les sujets et les cibles sont regroupés en Domaines (Domains).

Le langage Ponder considère plusieurs types de politiques de contrôle d'accès. Nous allons introduire ces types dans les paragraphes qui suivent.

Politiques d'autorisation

Les politiques d'autorisation définissent l'ensemble des actions permises et défendues pour un ensemble de sujets et sur un ensemble de cibles. Elles peuvent également être validées par un ensemble de contraintes. Ces contraintes vont être présentées dans la sous-section suivante.

Politiques de filtrage de l'information

Les politiques de filtrage servent à sélectionner l'information accessible aux sujets.

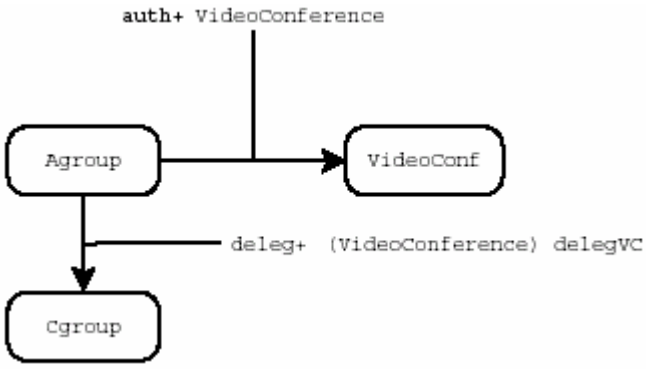
Ces politiques sont associées aux actions dans les politiques d'autorisations. Comme exemple, nous pouvons considérer la politique d'autorisation suivante :

```
inst auth+ VideoConference {
subject   /Agroup + /Bgroup;
target    /USAgroup -NYgroup;
action    VideoConf(BW,Priority)
          { in BW=2; in Priority=3;}
```

Une politique de filtrage de l'information est associée à la politique d'autorisation nommée *VideoConference*. Cette politique autorise aux membres des groupes */Agroup* et */Bgroup* d'initialiser une vidéoconférence avec les membres du groupe USA*group* excepté ceux de NY*group*. L'initialisation d'une vidéoconférence prend comme paramètres la bande passante (BW) et la priorité (Priority). Une politique de filtrage est associée à cette action, elle limite la bande passante à 2MB/s et établit sa priorité à 3.

Politiques de délégation

Les politiques de délégation autorisent un ensemble d'utilisateurs à transférer les autorisations d'accès, en entier ou en partie, qu'ils possèdent à un autre ensemble d'utilisateurs. Un sujet peut donc déléguer à un autre utilisateur les droits qu'il possède.



Dans cet exemple le groupe *Agroup* délègue à *Cgroup* la politique d'autorisation *VideoConference*. La politique de délégation est appelée *DelegVC*.

Politique de retenue

Les politiques de retenue (*Restrain Policies*) sont des politiques qui empêchent un groupe de sujets d'exécuter un ensemble d'actions sur un ensemble de cibles. Les politiques de retenue sont similaires aux politiques d'autorisation négatives, mais ces dernières sont gérées par les cibles alors que les politiques de retenues sont gérées par les sujets.

Politiques d'obligation

Les politiques d'obligation spécifient les actions à entreprendre à la suite d'un événement.

```
inst oblig LoginFailure {
on          3*loginfail(userid)
subject     s = /NRegion/SecAdmin;
target      t = /NRegion/users;
do          t.disable() -> s.log(userid);
```

La politique de l'exemple ci dessus est déclenchée à la suite de trois échecs d'authentification successifs par les sujets du domaine */NRegion /SecAdmin* accédant au domaine */NRegion/users*. La politique désactive le sujet et écrit son identificateur dans le journal.

a) Composition des politiques

Le langage Ponder offre la possibilité de structurer les politiques de contrôle d'accès
en des groupes de politiques. Un groupe peut contenir des politiques ou des sous-groupes.
Le regroupement des politiques peut avoir deux objectifs :

- structurer et organiser les politiques pour faciliter l'accès et la réutilisation et là on parle de groupe (group)
- structurer les politiques selon leur sémantique et les associer à un domaine de sujets et là on parle de *rôle*

b) Contraintes dans Ponder

Les contraintes des politiques de contrôle d'accès servent à spécifier les conditions de validité et d'applicabilité des politiques. Dans Ponder, il y a deux types de contraintes les contraintes relatives à une seule politique et les contraintes relatives à un groupe de politiques : *métapolitiques*.

Contraintes relatives à une seule politique

Les contraintes relatives à une seule politique sont spécifiées au niveau des politiques d'autorisation avec des prédicats. Ces contraintes peuvent être basées sur :

- les sujets et les cibles
- les paramètres des actions et des événements
- le temps (date et heure)

```
inst auth+ VideoConference {
subject   /Agroup + /Bgroup;
target    /USAgroup -NYgroup;
action    VideoConf(BW,Priority)
when      time.between("1600","1800");
```

L'exemple ci-dessus montre la même politique d'autorisation de la sous-section précédente mais qui possède une contrainte sur le temps. Ainsi, cette politique n'est appliquée qu'entre 16h00 et 18h00.

Métapolitiques

Les métapolitiques ont pour objectif de définir des contraintes sur un ensemble de politiques. Une méta-politique utilise un sous-ensemble de OCL (*Object Constraint Language* [Group, O.M, 2003]). Une contrainte est exprimée avec des d'expressions OCL qui peuvent déclencher l'exécution d'actions.

c) Conflits dans Ponder

Dans la mesure ou plusieurs politiques peuvent s'appliquer à un contexte particulier, des conflits peuvent exister entre elles. Dans Ponder deux types de conflits peuvent être détectés et analysés [Lupu et Sloman, 1997] [Lupu et Sloman, 1999] : les conflits de modalité et les conflits sémantiques.

Conflits de modalité.

Si des politiques positives et négatives s'appliquent pour les mêmes sujets, cibles et actions, alors il y aura un conflit de modalité. Ce type de conflits survient lorsque les domaines des politiques chevauchent (domaines des sujets, cibles et actions).

Pour résoudre ce genre de conflits, des règles de priorité et de préséance peuvent être établies afin d'aboutir à une seule décision. Comme relations de préséance nous pouvons citer :

Les politiques négatives ont toujours la priorité

Des priorités explicites sont assignées aux politiques

Considérer les politiques les plus spécifiques

Ce genre de conflits est détecté par l'outil Ponder Toolkit .

Conflits sémantiques

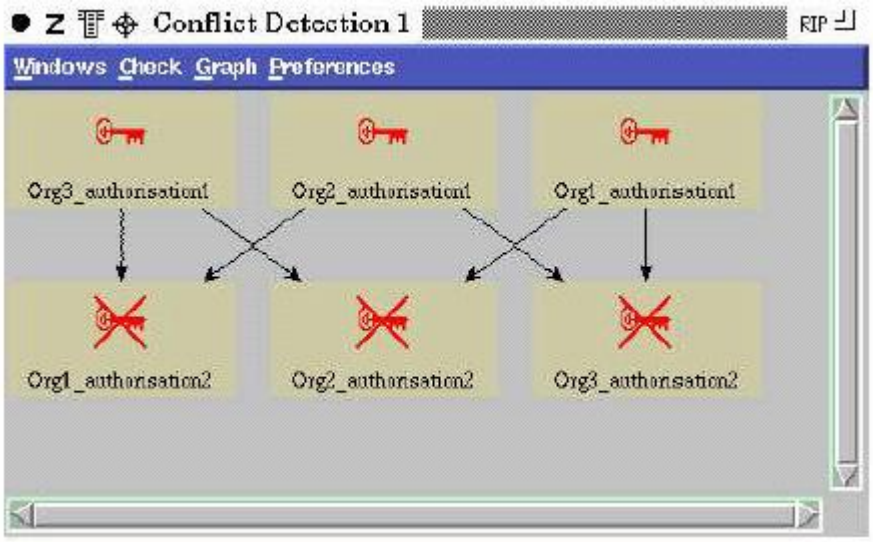
Les conflits sémantiques sont des situations non conformes aux spécifications d'un système. Ces conflits sont spécifiques aux applications et aux contextes associés. Ces conflits dépendent des facteurs externes tels que la disponibilité des ressources ou bien des politiques externes telles que les politiques globales dans une entreprise. Nous pouvons citer les exemples suivants :

- Séparation des taches : une même personne ne peut pas autoriser et initier le même paiement.
- Conflits des ressources : une ressource n'est disponible qu'en quantité limitée.
- Un administrateur a toujours la priorité d'accès.

Ces situations peuvent générer des conflits sémantiques. En effet, les conflits sémantiques sont le résultat de la contradiction entre les politiques et les propriétés du système. Pour traiter ce genre de conflits les métapolitiques sont utilisées pour caractériser ce genre de situation (OCL) et effectuer les actions nécessaires.

d) Outils pour la détection de conflits

Ponder *Toolkit* est une suite d'outils [Damianou, 2002] permettant de gérer et d'analyser les politiques de contrôle d'accès dans un environnement distribué. Cette suite intègre un outil d'analyse des conflits de modalité (*Conflict Analyzer*). Cet outil est implémenté en Java et détecte le chevauchement des domaines et les présente sous un format graphique.



La figure ci-dessous montre le résultat obtenu pour la détection de conflits de modalité pour six politiques d'autorisations, trois politiques positives (les clés) et trois politiques négatives (les clés barrées). La relation de chevauchement est représentée par un arc orienté de la politique la plus spécifique vers la politique la plus générale. Nous pouvons bien remarquer que les politiques positives chevauchent avec les politiques négatives, ce qui peut etre une source de conflit. Par exemple la politique positive *Org1_authorisation1* annule la politique négative *Org3_authorisation2* car leurs domaines chevauchent.

La figure ci-dessus montre un exemple de conflit. Les sujets des domaines *O2_m1* et *O2_m2* peuvent effectuer les opérations retract*(), delete() et disable()* sur les cibles du domaine *SharedPolicies* puisque la politiques *Org2_authorisation1* l'autorise. Mais, les politiques *Org1_authorisation2 et Org3_authorisation2* ne l'autorisent pas. L'outil permet également de transformer les politiques et les métapolitiques en des prédicats en Prolog afin de détecter les conflits sémantiques.

Les travaux réalisés sur Ponder restent spécifiques à ce langage. La vérification et l'analyse se font avec deux techniques différentes pour les conflits de modalité et les conflits sémantiques. Ces deux types de conflits sont étroitement liés mais il n'y a pas un seul outil qui les traite. De plus, malgré que les métapolitiques représentent une technique efficace pour traiter les conflits, il existe encore un risque que les métapolitiques soient elles mêmes incohérentes

3.5. Conclusion

.

Dans ce chapitre, nous avons proposé une approche de gestion autonome, basée sur des politiques , adaptée aux réseaux IP multimédia pour le support de la QoS de bout en bout. Cette approche permet d'automatiser la gestion des ressources à l'intérieur de chaque équipement afin de satisfaire les objectifs de l'administrateur.

Afin de réaliser cette proposition, nous allons étudier la manière dans le politiques sont stoker, pour cela l'annuaire LDAP reste la solution préconisée, le chapitre suivant présente en détail les principaux concepts des annuaire LDAP

CHAPITRE 4

4. LES ANNUAIRES LDAP

4.1. Introduction

Un annuaire est un recueil de données dont le but est de pouvoir retrouver facilement des ressources (généralement des personnes, des ressources informatiques ou des organisations) à l'aide d'un nombre limité de critères. Les annuaires électroniques sont un type de base de données spécialisée permettant de stocker des informations de manière hiérarchique et offrant des mécanismes simples pour rechercher l'information, la trier, l'organiser selon un ou plusieurs critères. L'utilisation d'annuaire ne se limite pas à la recherche de personnes ou de ressources. En effet, un annuaire peut servir à :

- **constituer un carnet d'adresse,**
- **authentifier des utilisateurs grâce à des mots de passe,**
- **définir les droits sur des utilisateurs de l'annuaire,**
- **recenser des informations sur un parc matériel (ordinateurs, serveurs, leurs adresses IP, etc.),**
- **décrire les applications disponibles et gérer leurs droits.**

Etant donné le grand nombre d'enregistrements que peut comporter un annuaire, plusieurs milliers voire un million pour les mises en œuvre les plus importantes, il n'est parfois pas concevable de créer un annuaire centralisé contenant les informations de toutes ces entrées. C'est la raison pour laquelle les serveurs d'annuaires doivent être capables de communiquer entres eux afin de partager l'information. C'est la grande force et l'avantage du protocole LDAP. Pour des grandes entreprises, un annuaire doit pouvoir être interrogé sur les adresses de personnes de différentes filiales localisées dans des régions géographiquement éloignées. Les serveurs d'annuaires possèdent la faculté de se répliquer, c'est-à-dire d'offrir des fonctions permettant d'importer et d'exporter des enregistrements (on dit généralement synchroniser) avec d'autres annuaires. On parle ainsi de réplication ou bien de synchronisation. La synchronisation est peut être assurée de deux manières différentes :

- **Grâce à des mécanismes de synchronisation intégrés par les produits du marché dans le système d'administration de l'annuaire**
- **Grâce à des outils appelés méta-annuaire (on parle aussi parfois de proxy LDAP). On distingue habituellement deux types de méta-annuaire. Les méta-annuaires avec réplication créant une base de donnée issue de l'agrégation des données provenant des différents annuaires et les méta-annuaires virtuels créant des références centrales vers les données contenues dans les différents annuaires, mais ne dupliquant pas l'information**.

4.2. Les concepts de LDAP

Dans le cadre de l'utilisation en mode client-serveur de la couche applications du modèle OSI, la fonctionnalité des annuaires (administration, authentification et contrôle de d'accès) a été élaborée à l'origine pour traiter la gestion des adresses électroniques. A partir de ces concepts, il a été reconnu qu'il y avait une utilisation possible avec plusieurs autres applications. Par conséquent, elle a été définie comme une norme ou un module distinct dans la recommandation X.500 de l'UIT-T [X.500]. Bien que la couverture ait été complète, les personnes chargées de la mise en œuvre à cette période, l'ont critiquée comme étant trop complexe et, par conséquent, trop difficile à mettre en œuvre.

Pour aborder ce problème du service annuaire DAP qui était trop complexe à mettre en œuvre, l'université du Michigan a élaboré une version plus simple du DAP basée sur TCP/IP en vue d'une utilisation sur Internet. Le LDAP offre de nombreuses fonctionnalités du DAP d'origine et on peut l'utiliser pour interroger des données d'annuaires exclusifs aussi bien que d'un service ouvert X.500. Au fur et à mesure des années, la plupart des fournisseurs importants de logiciels de courrier électronique et de services d'annuaires se sont montrés intéressés par le LDAP, celui-ci est maintenant le véritable protocole d'annuaire utilisé pour Internet.

4.2.1. Le protocole LDAP

Bien que le LDAP ait commencé comme un simple composant de l'annuaire X.500, il évolue en un service d'annuaire complet. En se servant des services de noms d'Internet, les réalisateurs construisent des couches de sécurité et ajoutent des capacités qui existent dans les composants de l'annuaire X.500, autres que le DAP. Par exemple, le contrôle de sécurité, duplication des données entre des sites multiples et utilisation de caractères plus complexes que le simple ASCII.

Les principaux éléments du protocole LDAP sont :

- **un protocole permettant d'accéder à l'information contenue dans l'annuaire et définit comment s'établit la communication client-serveur,**
- **un modèle de données qui définit le type de données contenues dans l'annuaire,**
- **un modèle de nommage qui définit comment l'information est organisée et référencée,**
- **un modèle fonctionnel qui définit comment on accède à l'information,**
- **un modèle de sécurité qui définit comment les données et l'accès sont protégés,**
- **un modèle de duplication qui définit comment la base est répartie entre serveurs.**
- **des APIs pour développer des applications clientes,**
- **LDIF, et plus récemment DSML, des formats d'échange de données.**

Un annuaire LDAP est basé sur un modèle de collection d'objets représentant des entrées d'annuaires. Chaque type d'entrées (d'objets) est composé de plusieurs attributs qui servent à décrire l'objet en question. Pour illustrer ce modèle, prenons un objet "*personne*". L'objet personne est défini par une liste d'attributs facultatifs ou obligatoires.

Des mécanismes de chiffrement comme SSL ou TLS, et d'authentification comme SASL, couplés à des mécanismes de règles d'accès (ou A.C.L. pour Access Control List) gérées par l'annuaire proprement dit, permettent de protéger les transactions et l'accès aux données.

La plupart des logiciels serveurs LDAP proposent également un protocole de communication serveur-serveur. Il permet à plusieurs serveurs d'échanger leur contenu et de le synchroniser ou bien de créer entre eux des liens permettant ainsi de relier des annuaires les uns aux autres.

Aujourd'hui dans la version actuelle du protocole LDAP est la version 3, celle-ci est définit par les standards délivrés par l'IETF [RFC 2251].

Concernant la communication serveur-serveur, le « referral service » est définit par LDAPv3, par contre le « replication service » est encore en cours de normalisation. Des produits du marché l'ont cependant intégré avec leur propre mécanisme en participant et anticipant la parution de la normalisation.

Contrairement à d'autres protocoles d'Internet, comme HTTP, SMTP ou NNTP, le dialogue LDAP ne se fait pas en ASCII mais utilise le format de codage Basic Encoding Rule (BER).

4.2.2. Modèles de données LDAP

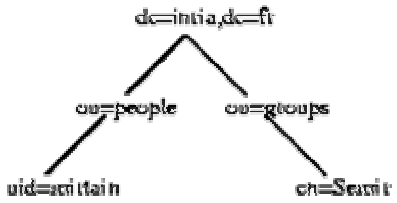
LDAP était à l'origine une passerelle d'accès à des annuaires X500. En 95, l'Université du Michigan créa le premier serveur LDAP autonome (*standalone LDAP*) utilisant sa propre base de données au format de type *dbm*. Les serveurs LDAP sont conçus pour stocker une grande quantité de données mais de faible volume et pour accéder en lecture très rapidement à celles-ci grâce au modèle hiérarchique.

Le Directory Information Tree

Les données LDAP sont structurées dans une arborescence hiérarchique qu'on peut comparer au système de fichier Unix. Chaque noeud de l'arbre correspond à une **entrée** de l'annuaire ou *directory service entry* (DSE) et au sommet de cette arbre, appelé *Directory Information Tree* (DIT), se trouve la *racine* ou *suffixe* (fig1). Ce modèle est en fait repris de X500, mais contrairement à ce dernier, conçu pour rendre un service d'annuaire mondial (ce que le DNS fait par exemple pour les noms de machines de l'Internet), l'espace de nommage d'un annuaire LDAP n'est pas inscrit dans un contexte global.

Les entrées correspondent à des *objets* abstraits ou issus du monde réel, comme une personne, une imprimante, ou des paramètres de configuration. Elles contiennent un certain nombre de champs appelés *attributs* dans lesquelles sont stockées des valeurs. Chaque serveur possède une entrée spéciale, appelée *root directory specific entry* (rootDSE) qui contient la description de l'arbre et de son contenu.

figure 1. exemple de DIT



Le schéma

L'ensemble des définitions relatives aux *objets* que sait gérer un serveur LDAP s'appelle le *schéma*. Le schéma décrit les *classes d'objets,* leurs types d'*attributs* et leur syntaxe.

Les attributs

Une entrée de l'annuaire contient une suite de couples types d'attributs - valeurs d'attributs. Les attributs sont caractérisés par :

- Un nom qui l'identifie
- Un Object Identifier (OID) qui l'identifie également
- S'il est mono ou multi-valué
- Une syntaxe et des règles de comparaison
- Un indicateur d'usage
- Un format ou une limite de taille de valeur qui lui est associée

Les attributs décrivent généralement des caractéristiques de l'objet (tableau 1), ce sont des attributs dits *normaux* qui sont accessibles aux utilisateurs (ex : attribut *givenname*). Certains attributs sont dits *opérationnels* car ils ne servent qu'au serveur pour administrer les données (ex : attribut *modifytimestamp*).

La syntaxe indique le type de données associées à l'attribut et la manière dont l'annuaire doit comparer les valeurs lors d'une recherche (tableau 2).

Certains serveurs LDAP respectent les standards X500 de hiérarchisation des attributs, qui permettent de décrire un attribut comme étant un *sous-type* d'un attribut *super-type* et d'hériter ainsi de ses caractéristiques. Par exemple, les attributs *cn*, *sn*, *givenname* sont des *sous-types* de l'attribut *super-type name*. Ces attributs *super-types* peuvent être utilisés comme critère de recherche générique qui porte sur tous ses *sous* attributs.

Les classes d'objets

Les *classes d'objets* modélisent des objets réels ou abstraits en les caractérisant par une liste d'attributs optionnels ou obligatoires. Une classe d'objet est définie par :

- Un nom qui l'identifie
- Un OID qui l'identifie également
- Des attributs obligatoires
- Des attributs optionnels
- Un type (structurel, auxiliaire ou abstrait)

Le type d'une classe est lié à la nature des attributs qu'elle utilise.

- Une *classe structurelle* correspond à la description d'objets basiques de l'annuaire : les *personnes*, les *groupes*, les *unités organisationnelles*... Une entrée appartient toujours au moins à une classe d'objet structurelle.
- Une *classe auxiliaire* désigne des objets qui permettent de rajouter des informations complémentaires à des objets structurels. Par exemple l'objet *mailRecipient* rajoute les attributs concernant la messagerie électronique d'une personne. L'objet *labeledURIObject* fait de même concernant les infos Web.
- Une *classe abstraite* désigne des objets basiques de LDAP comme les objets *top* ou *alias*.

Les classes d'objets forment une hiérarchie, au sommet de laquelle se trouve l'objet *top*. Chaque objet hérite des propriétés (*attributs*) de l'objet dont il est le fils. On peut donc enrichir un objet en créant un objet fils qui lui rajoute des attributs supplémentaires.
On précise la classe d'objet d'une entrée à l'aide de l'attribut *objectClass*. Il faut obligatoirement indiquer la parenté de la classe d'objet en partant de l'objet *top* et en passant par chaque ancêtre de l'objet., l'objet *inetOrgPerson* a la filiation suivante :

Exemple :

objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson

L'objet *person* a comme attributs : *commonName, surname, description, seeAlso, telephoneNumber,userPassword*

L'objet fils *organizationalPerson* ajoute des attributs comme : *organizationUnitName, title, postalAddress...*

L'objet petit-fils *inetOrgPerson* lui rajoute des attributs comme : *mail, labeledURI, uid (userID), photo ...*

Une entrée peut appartenir à un nombre non limité de classes d'objets. Les attributs obligatoires sont dans ce cas la réunion des attributs obligatoires de chaque classe.

ableau 1 : Exemples de classes d'objets

Entry Type	Required Attributes	Optional Attributes
inetOrgPerson (defines entries for a person)	commonName (cn) surname (sn) objectClass	businessCategory carLicense departmentNumber description employeeNumber facsimileTelephone Number givenName mail manager mobile organizationalUnit (ou) pager postalAddress roomNumber secretary seeAlso telephoneNumber title labeledURI uid
organizationalUnit (defines entries for organizational units)	ou objectClass	businessCategory description facsimileTelephoneNumber location (l) postalAddress seeAlso telephoneNumber
organization (defines entries for organizations)	o objectClass	businessCategory description facsimileTelephoneNumber location (l)

		postalAddress seeAlso telephoneNumber

Tableau 2. types de formats des attributs

Type	Description
bin	binary information
ces	case exact string (case of text is significant during comparison).
cis	case ignore string (case of text is ignored during comparison).
tel	telephone number (numbers are treated as text, but all blanks and dashes (-) are ignored).
dn	distinguished name.

Les OIDs

Les objets et leurs attributs sont normalisés par le RFC 2256 de sorte à assurer l'interopérabilité entre les logiciels. Ils sont issus du schéma de X500, plus des ajouts du standard LDAP ou d'autres consortiums industriels. Ils sont tous référencés par un *object identifier* (OID) unique dont la liste est tenue à jour par l'*Internet Assigned Numbers Authority* (IANA).
Il est possible de modifier le schéma en rajoutant des attributs à un objet (déconseillé) ou en créant un nouvel objet (mieux) et d'obtenir un OID pour cet objet auprès de l'IANA (encore mieux).

Un OID est une séquence de nombres entiers séparés par des points. Les OID sont alloués de manière hiérarchique de telle manière que seule l'autorité qui a délégation sur la hiérachie "1.2.3" peut définir la signification de l'objet "1.2.3.4". Par exemple :

2.5 - fait référence au service X500
2.5.4 - est la définition des types d'attributs
2.5.6 - est la définition des classes d'objets
1.3.6.1 - the Internet OID
1.3.6.1.4.1 - IANA-assigned company OIDs, used for private MIBs
1.3.6.1.4.1.4203 - OpenLDAP

La définition du schéma

Il existe plusieurs formats pour décrire un schéma LDAP :

- *slapd.conf* : fichier de configuration utilisé par U-M slapd, OpenLDAP et Netscape Directory
- *ASN.1* : utilisé dans les documents décrivant les standards LDAP et X500
- *LDAPv3* : La version 3 du protocole LDAP introduit l'obligation pour un serveur de publier son schéma via LDAP, pour permettre aux applications clientes d'en connaître le contenu. Le *schéma* est localisé par l'attribut opérationnel *subschemaSubentry* de l'entrée rootDSE. La valeur de cet attribut est une liste de DNs qui pointent vers des entrées, dont la classe d'objet est *subschema*, dans lesquelles sont stockées les descriptions des objets et des attributs.

Le tableau 3 montre les différences de syntaxe pour l'attribut *cn* et l'objet *person*

Tableau 3. formats de description du schéma

	slapd.conf	ASN1	LDAPv3
attribut	attribut cn commonName 2.5.4.3 cis	ub-common- name INTEGER ::= 64 commonName ATTRIBUTE WITH ATTRIBUTE-SYNTAX caseIgnoreStringSyntax (SIZE (1..ub-common-name)) ::= {attributeType 3}	attributetypes: (2.5.4.3 NAME 'cn' DESC 'commonName Standard' Attribute' SYNTAX 1.3.5.1.4.1.1466.115.121.1.15)
objet	objectclass person oid 2.5.6.6 superior top requires sn, cn allows description, seeAlso, telephoneNumber, userPassword	person OBJECT-CLASS ::= { SUBCLASS OF top MUST CONTAIN { commonName, surname} MAY CONTAIN { description, seeAlso, telephoneNumber, userPassword} ::= {objectClass 6}	objectclass: (2.5.6.6 NAME 'person' DESC 'standard person' Object Class' SUP 'top' MUST (objectclass $ sn $ cn) MAY (description $ seealso $ telephonenumber $ userpassword))

Quand une entrée est créée, le serveur vérifie si sa syntaxe est conforme à sa classe ou ses classes d'appartenance : c'est le processus de *Schema Checking*.

Il existe deux objets abstraits particuliers : les *aliases* et les *referrals* qui permettent à une entrée de l'annuaire de pointer vers une autre entrée du même ou d'un autre annuaire. L'attribut *aliasObjectName* de l'objet *alias* a pour valeur le DN de l'entrée pointée. L'attribut *ref* de l'objet *referral* a pour valeur l'URL LDAP de l'entrée désignée.

Le Distinguish Name

Chaque entrée est référencée de manière unique dans le DIT par son *distinguished name* (DN). Le DN représente le nom de l'entrée sous la forme du chemin d'accès à celle-ci depuis le sommet de l'arbre. On peut comparer le DN au path d'un fichier Unix. Par exemple, mon DN correspondant à l'arbre de la figure 1 est :

uid=mirtain,ou=people,dc=inria,dc=fr

Le DN représente le chemin absolu d'accès à l'entrée. Comme pour le système de fichier Unix, on peut utiliser une *relative distinguished names* (RDNs) pour désigner l'entrée depuis une position déterminée de l'arbre. Par exemple, à partir de la position dc=inria, dc=fr de la figure 1, on peut employer les RDNs suivants :

ou=people
ou=groups
cn=Semir,ou=groups
uid=mirtain, ou=people

4.2.3. LDIF

LDAP Data Interchange Format (LDIF) permet de représenter les données LDAP sous format texte standardisé, il est utilisé pour afficher ou modifier les données de la base. Il a vocation à donner une lisibilité des données pour le commun des mortels.

LDIF est utilisé dans deux optiques :

- faire des imports/exports de base
- faire des modifications sur des entrées.

La syntaxe est un nom d'attribut suivi de : suivi de la valeur (uid: mirtain), le premier attribut d'une entrée étant le DN (dn: uid=mirtain,ou=people,dc=inria,dc=fr). Le format utilisé est l'ASCII, les données binaires étant codés en base 64.

La forme générale est :

dn: <distinguished name
objectClass: <object class
objectClass: <object class
...
<attribute type:<attribute value
<attribute type:<attribute value

Une entrée de type personne se représente de la manière suivante :

dn: cn= June Rossi, ou= accounting, o= Ace Industry, c= US
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: June Rossi
sn: Rossi
givenName: June
mail: rossi@aceindustry.com
userPassword: {sha}KDIE3AL9DK
uid: rossi
telephoneNumber: 2616
roomNumber: 220

Les commandes de modification ont la syntaxe suivante :

dn: distinguished name
changetype <identifier
change operation identifier
list of attributes...
...
-
change operation identifier
list of attributes
...
<identifier :
add (ajout d'une entrée),
delete (suppression),
modrdn (modification du RDN),
modify (modification : add, replace, delete d'un attribut)

Le caractère - spécifie le séparateur entre 2 instructions

Par exemple :

Ajouter le numéro de téléphone et le nom du manager pour la personne Lisa Jangles :

dn: cn= Lisa Jangles, ou= Sales, o= Ace Industry, c= US
changetype: modify
add: telephonenumber
telephonenumber: (408) 555- 2468
-
add: manager
manager: cn= Harry Cruise, ou= Manufacturing, o= Ace Industry, c= US

Pour détruire l'entrée

dn: cn= Lisa Jangles, ou= Sales, o= Ace Industry, c= US

changetype: delete

LDAP utilise le jeu de caractères *Unicode Transformation Format- 8* (UTF-8) pour le stockage des valeurs d'attributs de type texte et celui des DNs. UTF- 8 englobant tous les jeux de caractères (isoLatin, Shift- JLS...), on peut employer différentes langues pour les valeurs d'attribut grâce à l'option *language code* de l'attribut (extension proposée par l_IETF). On peut donc ainsi créer des annuaires multilingues.

Par exemple, on peut avoir pour un objet personne, un attribut *description* en français et un autre en japonais :

description, lang-fr : le texte en français

description, lang-ja : le même en japonais

Le code suit le standard ISO 639.

4.2.4. Le modèle fonctionnel

Les opérations de base sont résumées dans le tableau 4. Elles permettent d'accéder au serveur ou de modifier la structure de l'arbre et/ou les entrées de l'annuaire. Elles jouent un rôle analogue aux commandes de manipulation de fichiers d'Unix (*cp*, *mv*, *rm*...).

Tableau 4. opérations de base

Opération LDAP	Description
Search	recherche dans l'annuaire d'objets à partir de critères
Compare	comparaison du contenu de deux objets
Add	ajout d'une entrée
Modify	modification du contenu d'une entrée
Delete	suppression d'un objet
Rename (Modify DN)	modification du DN d'une entrée
Bind	connexion au serveur
Unbind	deconnexion
Abandon	abandon d'une opération en cours
Extended	opérations étendues (v3)

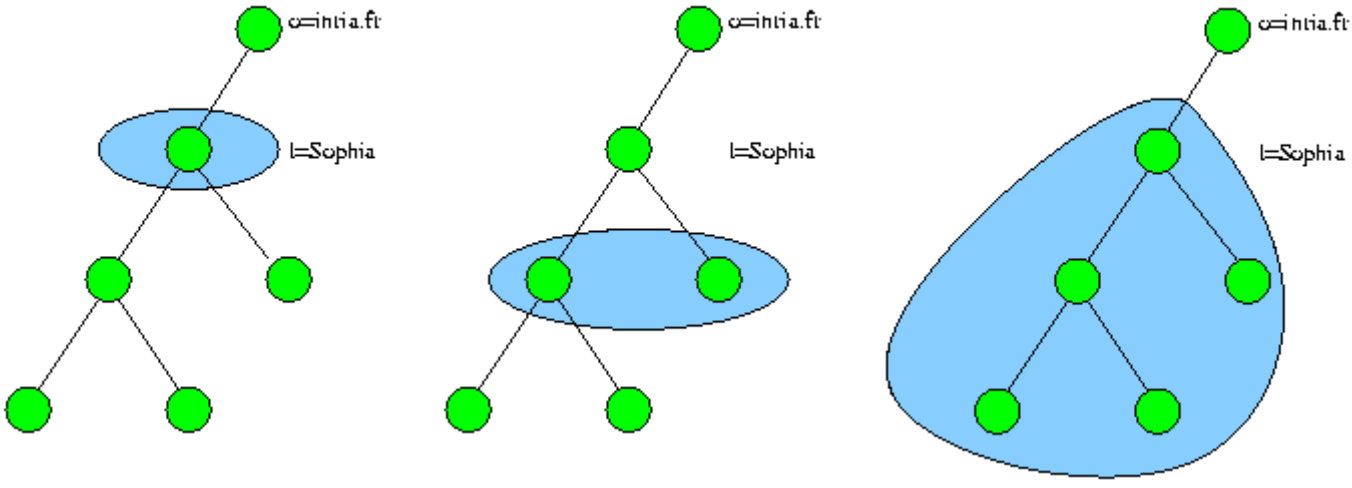
Les commandes *search* et *compare* se font sous la forme d'une requête composée de 8 paramètres (tableau5)

:

Tableau 5. paramètres d'une requête	
Paramètre	**Description**
base object	l'endroit de l'arbre où doit commencer la recherche
scope	la profondeur de la recherche
derefAliases	si on suit les liens ou pas
size limit	nombre de réponses limite
time limit	temps maxi alloué pour la recherche
attrOnly	renvoie ou pas la valeur des attributs en plus de leur type
search filter	le filtre de recherche
list of attributes	la liste des attributs que l'on souhaite connaître

Le scope définit la profondeur de la recherche dans le DIT. La figure 2 met en relief la portée d'une recherche ou d'une comparaison en fonction du paramètre *scope* :

figure 2. le scope



search scope = base | search scope = onelevel search | searchscope = subtree

Il n'existe pas de fonction *read* dans LDAP. Cette fonction est simulée par la fonction *search* avec un *search scope* égal à *base*.

Le filtre de recherche s'exprime suivant une syntaxe spécifique dont la forme générale est :

(< operator(< search operation)(< search operation)...))

Ce filtre décrit une ou plusieurs conditions exprimées sous forme d'expressions régulières sensées désigner un ou plusieurs objets de l'annuaire, sur lesquels on veut appliquer

L'opération voulue. Le tableau 6 récapitule les opérateurs de recherche disponibles :

Tableau 6. opérateurs de recherche

Filtre	Syntaxe	Interprétation
Approximation	(sn~=Mirtain)	nom dont l'orthographe est voisine de Mirtain
Egalité	(sn=Mirtain)	vaut exactement Mirtain
Comparaison	(sn>Mirtain) , <= , >= , <	noms situés alphabétiquement après Mirtain
Présence	(sn=*)	toutes les entrées ayant un attribut sn
Sous-chaîne	(sn=Mir*), (sn=*irtai*), (sn=Mirt*i*)	expressions régulières sur les chaînes
ET	(&(sn=Mirtain) (ou=Semir))	toutes les entrées dont le nom est Mirtain et du service Semir
OU	(|(ou=Direction) (ou=Semir))	toutes les entrées dont le service est le Semir ou la Direction
Négation	(!(tel=*))	toutes les entrées sans attribut téléphone

Lors de la connexion au serveur (*bind*), ce dernier demande une authentification. Le client doit alors fournir un *DN* et le *mot de passe* correspondant, celui-ci transitant en clair. Pour sécuriser les transactions,. LDAPv3 fournit la possibilité d'utiliser du chiffrement (SSL ou TLS) et le mécanisme *Simple Authentification and Security Layer* (SASL) procurant des outils d'authentification plus élaborés à base de clés (OTP...). Une fois connecté, le client peut envoyer autant de commandes qu'il souhaite jusqu'à ce qu'il ferme la session (*unbind*).

Chaque commande se voit attribuer un *numéro de séquence,* qui permet au client de reconnaître les réponses lorsque celles-ci sont multiples - ce qui peut parfois arriver lors d'une recherche simple qui peut renvoyer jusqu'à plusieurs milliers d'entrées. A chaque opération, le serveur renvoie également un *acquittement* pour indiquer que sa tâche est terminée ou qu'il y a une erreur.

4.2.5. Les URLs LDAP

Les URLs LDAP [RFC2255] permettent aux clients Web d'avoir un accès direct au protocole LDAP. La syntaxe est de la forme :

ldap[s]://<hostname>:<port>/<base_dn>?<attributes>?<scope>?<filter>

<base_dn> : DN de l'entrée qui est le point de départ de la recherche

<attributes>: les attributs que l'on veut consulter

<scope> : la profondeur de recherche dans le DIT à partir du

<base_dn> :"base" | "one" | "sub"

<filter> : filtre de recherche, par défaut (objectClass=*)

exemples :

ldap://ldap.netscape.com/ou=Sales,o=Netscape,c=US?cn,tel,mail?scope=sub?(objetclass=person)

ldap://ldap.loria.fr/cn=Laurent%20Mirtain,ou=Moyens%20Informatiques,o=loria.fr

ldap://ldap.loria.fr/o=loria.fr?mail,uid,sub?(sn=Mirtain)

4.3. Exemples d'application de LDAP

LDAP peut fournir différents services. On peut l'utiliser comme :

- Un protocole de diffusion de données : annuaire LDAP de type pages blanches pour contacter les personnes.
- Un protocole de service pour des applications : annuaire LDAP des adresses mail qui permet aux outils ou serveurs de mail de composer ou vérifier les adresses, utilisé par des gestionnaires de liste (sympa), base permettant de stocker les préférences de configuration d'une application (profiles communicator).
- Une passerelle entre applications : permettant l'échange de données entre applications incompatibles, par exemples les carnets d'adresses Netscape Communicator et Microsoft Outlook.
- Un protocole de service pour les systèmes d'exploitation ou les services Internet : base LDAP des utilisateurs utilisée par les systèmes pour assurer l'authentification (certificats d'authentification) et gérer les droits d'accès aux ressources réseau - Des projets sont en cours pour remplacer NIS, NIS+ par LDAP.

LDAP fournit un protocole standardisé d'accès à de l'information. Cette information peut être de toute nature et servir à différents types d'applications, allant d'un système d'annuaire classique jusqu'aux appels systèmes du système d'exploitation.

LDAP est disponible sur de nombreux types de plates-formes, ce qui lui confère un statut de service fédérateur, pouvant centraliser des informations issues de différentes sources - et en simplifier ainsi la gestion - et destinées à différentes applications et différents utilisateurs.

De ce fait, LDAP peut devenir la clef du système d'information de l'entreprise. Cela en rend la conception et la maintenance d'autant plus cruciales. C'est pourquoi la mise en place d'un tel service, à une telle échelle, relève d'une mission longue et délicate pouvant impliquer de nombreux intervenants et sources d'informations et pouvant parfois remettre en cause certains fonctionnements existants.

Apres avoir aborder tous les principaux concepts des annuaires LDAP, nous allons déployé un annuaire LDAP on nous basson sur une politique de qualité de service

4.4. Déployer un service d'annuaire LDAP

Déployer un service d'annuaire LDAP nécessite en premier lieu une réflexion sur la nature des données que l'on y met, sur la manière dont on les récupère, sur l'utilisation que l'on compte en faire et sur la façon de gérer le tout. Dans notre cas, il s'agie de règle de politique. La mise en place d'un annuaire LDAP met donc en jeu plusieurs phases de conception que l'on va passer en revue.

4.4.1. Déterminer les besoins en service d'annuaire et ses applications

Déployer un système d'annuaire se fait généralement sous la contrainte de la mise en place ou du remplacement d'une application. C'est alors que se pose la question d'élargir le service à d'autres types d'applications, la première venant à l'esprit étant un annuaire de régle. Cette phase consiste donc à prévoir toutes les applications possibles, actuelles ou futures, d'un annuaire LDAP.

4.4.2. Déterminer quelles données sont nécessaires

Il s'agit d'inventorier la liste exhaustive des données que l'on souhaite inclure dans le système d'information et de déterminer ensuite par quelle source les obtenir et les maintenir à jour. Des aspects comme le format, la taille des données, leur confidentialité, leur pertinence, leur source (statique, dynamique...), leur pérennité, les personnes susceptibles de les fournir, de les maintenir et d'y accéder doivent être pris en compte lors de cette phase. De ce point de vue, c'est la plus délicate à franchir car elle implique d'autres intervenants. Il faut également se faire une idée précise sur la manière dont les données vont être maintenues à jour : synchronisation avec un SGBD, intervention manuelle, scripts automatiques...

4.4.3. Choisir son schéma

Dans cette phase, on désigne le schéma, il s'agit de choisir, en fonction des données que l'on a retenues, quelles sont les *classes d'objets* et les *types d'attributs* qui s'en rapprochent le plus pour construire son annuaire LDAP.

La plupart du temps, les schémas standards, issus de X500 et de LDAP conviennent aux besoins de modélisation. De plus, en fonction du logiciel que l'on choisira, des objets supplémentaires seront fournis. Il reste, au final, la possibilité de créer ses propres objets, spécifiques à ses besoins. En règle générale, il faut éviter de modifier le schéma existant car l'on risque de rendre son annuaire inutilisable par les applications clients ou les autres serveurs.
Il est préférable de créer une *sous classe* d'une classe d'objet existante et exploiter le mécanisme d'héritage d'*attributs* des *classes objets*. Par exemple Supposant que l'on doit fournir un service vidéo Svideo. Cette opération doit être mise en application comme politique : Fournissez le service Svideo visuel pour les utilisateurs autorisés entre les points autorisés, mais seulement à des heures convenues.

On peut créer la classe d'objet *ApprovedUsers* fille de *VideoServices* dans laquelle on définira les attributs nécessaires à ses besoins :

```
objectclass ApprovedUsers
superior VideoServices
requires
    sn,
    cn
allows
    user1,
    user2,
    user3,
    user4,
```

Dans tous les cas, il faut prévoir de documenter son schéma pour en faciliter la maintenance et l'évolution. Il faut proscrire également la désactivation de l'option de *schema checking* implantée dans la plupart des serveurs, qui permet de vérifier que les attributs saisis sont bien conformes au schéma que l'on a choisi.

4.4.4. Concevoir son espace (modèle) de nommage

Cette étape consiste à définir comment les entrées de l'annuaire vont être organisées, nommées et accédées. L'objectif est de faciliter leur consultation et leur mise à jour mais aussi de prévoir leur duplication, leur répartition entre plusieurs serveurs ou leur gestion par plusieurs personnes. En fonction de ces priorités, on privilégiera tel ou tel espace de nommage.

Les paramètres qu'il faut prendre en compte lors de cette étude sont les suivants :

- Le nombre d'entrées prévu et son évolution ?
- La nature (type d'objet) des entrées actuelles et futures ?
- Vaut-il mieux centraliser les données ou les distribuer ?
- Seront-elles administrées de manière centrale ou faudra-t-il déléguer une partie de la gestion
- La duplication est-elle prévue ?
- Quelles applications utiliseront l'annuaire et imposent-elles des contraintes particulières ?
- Quel attribut utiliser pour nommer les entrées et comment garantir son unicité ?

Durant cette phase, nous allons choisir le modèle d'organisation des données, leur mode de désignation et le suffixe de notre organisation.

4.4.5. Le Directory Tree

Le modèle de nommage structuré en arbre hiérarchique de LDAP est repris du standard X500. Ce dernier a été conçu dans l'optique d'un service global
Le modèle LDAP, lui, n'impose pas une racine universelle du DIT car il renonce à être un service d'annuaire mondial et se limite à une petite communauté. Dans ce cadre, le modèle LDAP peut être plat (fig 1),, *branché* par type d'objet (fig 2).

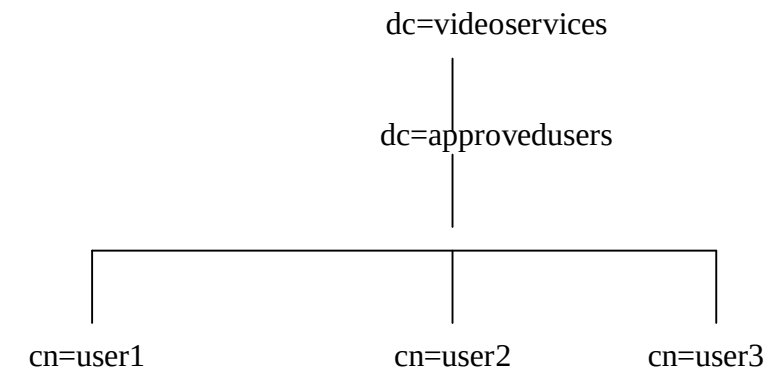


figure 1 espace de nommage plat

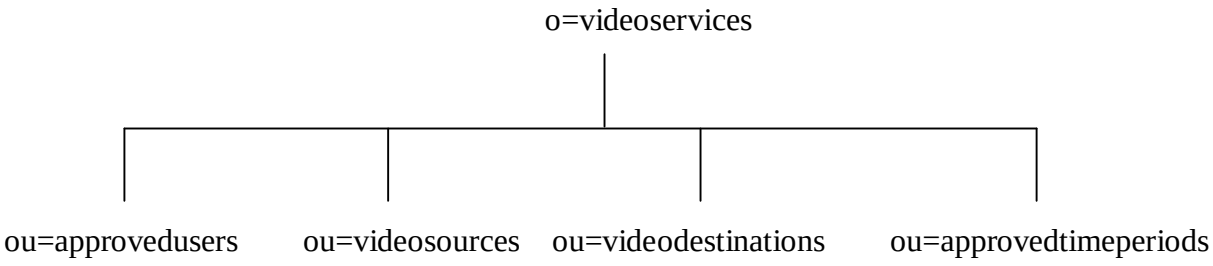


Figure 2 espace de nommage basé sur les objets

En règle générale, en terme de performance, il vaut mieux avoir un arbre le plus plat possible. Par contre, en terme de facilité d'administration, il vaut mieux introduire du branchage par type d'objet ou par organisation. Cela facilite les mises à jour de données ou la mise en place de règles d'accès spécifiques à une partie de l'annuaire ou la répartition de la gestion de l'arbre entre plusieurs serveurs. Si votre organisation change souvent ou bien que le personnel est très mobile, le branchage par organisation est alors à proscrire.

4.4.6. Nommage des entrées

La deuxième étape consiste à choisir l'attribut utilisé pour le DN de l'entrée, dans la partie RDN. Ce choix peut dépendre des applications clientes, mais il doit se faire surtout de manière à garantir l'unicité de la valeur utilisée.

4.4.7. Choix du suffixe

La dernière étape consiste à choisir le suffixe. C'est en quelque sorte l'identifiant de l'annuaire. Son choix est important car, même si la base n'a qu'une vocation interne, elle peut à terme devenir en partie un maillon d'un annuaire global ou d'un système d'information. Le suffixe peut à l'extrême être une chaîne vide, mais dans l'optique d'une diffusion de son annuaire, on choisira, en général, un suffixe unique au monde. Pour cela, il est recommandé d'utiliser comme suffixe d'annuaire, le nom de domaine DNS de l'organisation.

4.4.8. Définir la topologie de son service

Dans cette phase, il faut réfléchir sur la manière dont le service d'annuaire LDAP va être rendu en termes de performance, de fiabilité et de facilité de gestion. Il faut prendre en compte :

- Les applications qui vont utiliser l'annuaire et le nombre d'utilisateurs.
- Les capacités du logiciel serveur qui va être choisi.
- La topologie de son réseau.
- Le design de son espace de nommage.

La conception de la topologie du service d'annuaire LDAP est étroitement liée à celle de l'espace de nommage. Il est donc possible de devoir revenir sur l'une ou l'autre durant la phase de conception ou même celle d'exploitation, dans le cas d'un changement d'organisation interne ou de marque de logiciel serveur.

La question principale de cette phase est de déterminer si la base et sa gestion seront centralisées sur un seul serveur ou si elles devront être éclatées sur plusieurs serveurs. La deuxième étude porte sur le nombre de serveurs redondants à déployer et leur emplacement sur le réseau physique.

4.4.9. Le partitionnement

Il consiste à séparer les données de l'annuaire sur plusieurs serveurs. Il peut être imposé par le volume d'entrées à gérer, leur gestion répartie sur plusieurs sites, les types d'accès au réseau physique ou le mode d'organisation de la société. Séparer les données ne veut pas dire forcément les dissocier : les standards LDAP et X500 définissent des moyens de les relier (recoller). Ce sont les services *referral* et *replication.*

4.5. Conclusion

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

Comme pour toutes les bases de données, son fonctionnement et son organisation interne dépendent fortement de son constructeur. Pour cette raison, dans un souci d'interopérabilité, tous les annuaires utilisent un protocole d'accès standardisé : LDAP, L'annuaire LDAP reste la principale technique pour réaliser le Policy Repository.

Nous aborderont dans le chapitre suivant le chemin formel qui conduit à préciser la nature contractuelle de la QoS dans les serveurs de politique.

CHAPITRE 5

5. LA LOGIQUE DE REECRITURE & MAUDE

5.1. Introduction

La logique de réécriture ("rewriting logic" ou RWL), telle que présentée dans [Clavel et al. 1999], [Martí-Oliet et Meseguer 1993], [Martí-Oliet et Meseguer 1996] et [Meseguer 1998], est une logique de changement concurrent, qui peut traiter de façon naturelle l'état du système, et des calculs concurrents non déterministes de haut niveau. Elle a de bonnes propriétés comme cadre de travail sémantique flexible et général, qui apporte une sémantique à un large ensemble de langages et de modèles à concurrence. En particulier, elle supporte très bien des calculs concurrents orientés-objets.

Une spécification en logique de réécriture peut être interprétée en termes de calcul ou en termes de logique. Les mêmes raisons qui en font un bon cadre de travail sémantique au niveau calculatoire, en font aussi un bon cadre de travail logique, c'est-à-dire une métalogique dans laquelle bien d'autres logiques peuvent être représentées et implémentées de façon naturelle. Par conséquent, certaines des applications les plus intéressantes sont celles de métalangages, dans lesquelles sont créés des environnements exécutables pour des prouveurs de théorèmes, des langages de programmation ou des modèles de calcul.
Un outil basé sur la logique de réécriture peut constituer à la fois un langage et un système de haute performance pour des calculs de logique équationnelle comme de logique de réécriture, avec un éventail d'applications important.
La suite de cette partie est organisée en commençant par la présentation des différents éléments de base de la logique de réécriture. Nous nous attarderons ensuite sur les théories de la logique de réécriture. Finalement, nous présentons les caractéristiques du langage Maude, défini à base de cette logique, que nous utiliserons dans le cadre de notre travail.
Pour d'avantages de précisions sur la réécriture de termes (dans un cadre plus général que la seule logique de réécriture), [Baader et Nipkow 1998] ou [Ohlebusch 2002].

5.2. Eléments de base

A l'origine la *réécriture* est une mise en oeuvre informatique de la preuve de théorèmes équationnels à partir d'axiomes (équations). Celle-ci consiste à utiliser ces équations pour remplacer les termes du théorème par des égaux. Le théorème est vrai si l'on peut obtenir à partir d'un des membres de l'équation, l'autre membre.
Devant l'indéterminisme de ce procédé, lié au double sens de l'équation et au choix des bons axiomes, on oriente les équations pour obtenir un système de règles de réécriture. Réécrire un terme t consiste à choisir un sous-terme t' de t et une règle de réécriture g → d, tels que g filtre t' (on dit que t' est instance de g), et à remplacer t' dans t par l'instance correspondante de *d*.

La réécriture a rapidement trouvé des applications directes dans le développement de systèmes de calculs formels. Plus récemment, elle a été appliquée dans la conception des systèmes d'aide à la preuve de programmes, un programme étant représenté par des spécifications formelles, la preuve automatique de théorème en logique du premier ordre, l'écriture d'interprètes de langages logiques. Les systèmes de réécriture ont été par ailleurs utilisés pour supporter des langages de contraintes symboliques, dans le contexte d'applications graphiques.

Dans le contexte de la logique de réécriture qui est basée sur la théorie de réécriture, réécrire un terme consiste à trouver une *relation de réécrivabilité* entre les classes d'équivalences de ce terme.

En logique de réécriture chaque système est représenté par une *théorie de réécriture* T = (*Σ, E, L, R*). Sa structure statique est décrite par la *signature* (*Σ, E*), alors que sa structure dynamique est décrite par les *règles de réécriture* R.
Une *signature* est formée par un ensemble de sortes *S* et d'opérateurs *F*, et un ensemble équations *E*. Cette signature (*Σ, E*) détermine une relation de congruence engendrée par *E* et un ensemble quotient des termes modulo cette relation. Les objets principaux de la logique de réécriture sont les *classes d'équivalence* des termes $T_{\Sigma,E}$.
Les *formules* de la logique de réécriture sont construites en utilisant des classes d'équivalence de cet ensemble. Elles ont la forme [*t*] → [*t*'] représentant intuitivement la proposition :
"*La classe d'équivalence contenant le terme t peut se réécrire en la classe d'équivalence contenant le terme t'* ".

Les symboles de fonctions Σ et leurs propriétés structurelles *E* sont entièrement définis par l'utilisateur. Ceci procure à la logique de réécriture une grande flexibilité dans la définition de la structure des états, offrant ainsi la possibilité de présenter des structures différentes et variés.

Exemple :

Soient la signature *(Σ, E)* , les règles de réécriture R et - *t*1, *t*2, *t3* $\in \mathbf{T}_{\Sigma,E}(X)$ des termes (avec ou sans variable).

Sorte *bool*

F (opns) *true :* $\rightarrow$ *bool*

false : $\rightarrow$ *bool*

not(_) : bool $\rightarrow$ *bool*

_ and _ : bool bool $\rightarrow$ *bool*

_ or _ : bool bool $\rightarrow$ *bool*

Vars x,y : bool

E (Eqns) *(not(true) = false)*

(not(false) = true)

(true and x = x)

(false and x = false)

(x or y = not(not(x) and not(y)))

R *x and y* $\rightarrow$ *x* (1)

x and y $\rightarrow$ *y* (2)

t1= *false,* **t2** = *true and X,* **t3** = *not ((true and false) or not(t2)).*

5.3. Théorie de réécriture

Une *théorie* de la logique de réécriture est donnée par une signature et par un ensemble de règles de réécriture représentant des axiomes. Les règles de réécriture agissent sur les classes d'équivalence de termes, de plus elles sont étiquetées, nous ajoutons donc à une théorie de réécriture un ensemble d'étiquettes.

Définition :

Une théorie de la logique de réécriture peut être décrite comme un quadruplet : *R = (Σ , E, L, R),*

où :

- Σ est un alphabet rangé de symboles de fonctions,
- *E* est un ensemble de Σ-équations,
- *L* est un ensemble de labels,
- *R* est un ensemble de paires.

Chaque paire de *R* est composée d'un label (une étiquette) et d'un couple de classes d'*E*-équivalence

de termes (c'est-à-dire les classes d'équivalence des termes modulo *E*) : $R \subseteq L \times T_{\Sigma, E}(X)^2$.

$X = \{x_1, \ldots, x_n, \ldots\}$ représente un ensemble de variables de cardinalité infinie.

Les éléments de *R* sont appelés des règles de réécriture : un couple labellisé *(r ([t], [t'])* est noté $r : [t] \longrightarrow [t']$.

Si cet ensemble de règles de réécriture est vide, on retrouve la théorie équationnelle classique de la logique équationnelle qui est à la base des spécifications algébriques.

Etant donné une théorie de réécriture, l'ensemble des formules impliquées par conséquences logiques cette théorie est défini par les *règles de déduction* de la logique de réécriture. Ces règles formalisent la notion de la réécriture des termes en définissant comment les preuves des théorèmes sont formées.

Dans un système concurrent, la séquence des transitions exécutées à partir d'un état initial donné, constitue ce que nous appelons le calcul qui correspond à une *preuve* ou à une *déduction* dans la logique de réécriture.

Pour une théorie de réécriture *R*, on dit que $[t] \rightarrow [t']$ est prouvable dans *R* (où *R* implique une formule $[t] \rightarrow [t']$) et on écrit $R \vdash [t] \rightarrow [t']$ ssi $[t] \rightarrow [t']$ est obtenue par une application finie des règles de déduction suivantes:

1. Réflexivité : pour chaque $[t] \in T_{\Sigma,E}(X)$,

$$[t] \rightarrow [t']$$

2. Congruence : pour chaque $f \in \Sigma n$, $n \in \boldsymbol{N}$

$$\frac{[t1] \rightarrow [t'1] \ldots [tn] \rightarrow [t'n]}{[f(t1, \ldots, tn)] \rightarrow [f(t'1, \ldots, t'n)]}$$

3. Remplacement : pour chaque règle de réécriture

$r : [t(x1, \ldots, xn)] \rightarrow [t'(x1, \ldots, xn)]$ **if** $[u1(x)] \rightarrow [v1(x)] \wedge \ldots \wedge [uk(x)] \rightarrow [vk(x)]$ dans *R*,

$$\frac{[w1] \rightarrow [w1'] \ldots [wn] \rightarrow [w'n] \quad [u1(w/x)] \rightarrow [v1(w/x)] \ldots [uk(w/x)] \rightarrow [vk(w/x)]}{[t(w/x)] \rightarrow [t'(w'/x)]}$$

Cette règle , signifie que par une substitution (w / x) de x_i par w_i ,$1 \leq i \leq n$. on peut déduire les séquences $[u_j(w / x)] \rightarrow [v_j(w / x)]$, $1 \leq j \leq k$. et si en plus nous pouvons déduire $[w_i] \rightarrow [w'_i]$, pour $1 \leq i \leq n$, nous pouvons alors déduire que : $[t(w / x)] \rightarrow [t'(w' / x)]$.

4. Transitivité :

$$\frac{[t1] \rightarrow [t2] \quad [t2] \rightarrow [t3]}{[t1] \rightarrow [t3]}$$

Après avoir présenté les 4 règles de base, la déduction dans la logique de réécriture est une itération des étapes suivantes :

1. La règle de remplacement identifie toutes les règles de réécriture applicables à l'état global courant. Et, comme la logique de réécriture est une logique de changement, la règle de réflexivité, sera appliquée aux sous termes non identifiés, les transforme mais en eux-mêmes.

2. Les règles de réécriture, identifiées par la règle de remplacement ainsi que la règle de réflexivité, sont exécutées en concurrence et indépendamment les unes des autres. La règle de congruence compose les effets, membres droits, de ces règles pour construire le nouveau terme global.

Les étapes (1) et (2) sont réitérées jusqu'à ce qu'il n'y ait plus de règle applicable.

3. Enfin, la règle de transitivité construit la séquence des réécritures faites du terme initiale jusqu'au terme final. La séquence ainsi construite correspond à un calcul possible dans le système concurrent.

5.4. Applications de la logique de réécriture

Toutes les applications typiques de la programmation équationnelle et des spécifications algébriques peuvent être supportées de façon adéquate et efficace par la logique de réécriture (Théories fonctionnelles). En fait, la logique équationnelle a un pouvoir d'expression suffisant pour offrir d'aussi bons avantages qu'un cadre de travail logique, pour un très large éventail de langages de spécification algébrique, et elle peut être facilement implémentée.

Néanmoins, beaucoup d'autres applications portent au-delà de la logique équationnelle. En effet, les théories de réécriture systèmes supportent les applications générales de la logique de réécriture. De plus, le domaine important de la spécification et du prototypage de systèmes concurrents et distribués basés sur les objets peut être supporté par ce type de théories.

Par ailleurs, la réflexivité rend possibles de nombreuses nouvelles applications de méta-programmation et de métalangage. En particulier, elle est très appréciable dans toutes celles qui utilisent la logique de réécriture comme cadre de travail logique et sémantique.

En outre, les langages de conception orientés-objets, les langages de description d'architecture et les langages pour composants distribués trouvent une sémantique naturelle dans la logique de réécriture.

En général, un système concurrent est défini par trois concepts de base : états, transitions et calcul. L'état décrit l'évolution du système et sa structure détermine si le système est concurrent ou non: si l'état est atomique, le système est séquentiel mais s'il est distribué, le système a le potentiel d'être concurrent. Le système passe d'un état à un autre en exécutant un ensemble de transitions locales, indépendantes et distribuées. La trace ou l'histoire de toutes les transitions exécutées de l'état initial jusqu'à l'obtention de l'état courant représente un calcul dans un système concurrent. Etant donné que la logique de réécriture est une logique de raisonnement correct sur les systèmes concurrents ayant des états et évoluant en terme de transitions, elle offre les outils nécessaires pour les décrire [Mes94].

La réécriture dans cette logique a lieu modulo les axiomes équationnels *E* (Par exemple, la réécriture peut se faire modulo la plupart des différentes combinaisons d'axiomes d'associativité, de commutativité, d'identité et d'idempotence), les règles de réécriture *R* et les quatre règles de déduction. Les règles de *R* n'ont pas besoin d'avoir la propriété de Church-Rosser ou de terminer. Plusieurs chemins de réécriture différents sont alors possibles; il en résulte que le choix de *stratégies* appropriées est crucial pour "exécuter" des théories de réécriture.

De telles stratégies ne forment pas pour autant une partie en dehors de la logique de ce langage. Au contraire, ce sont des stratégies internes définies par des théories de réécriture au *méta-niveau*. En effet, cela est rendu possible parce que la logique de réécriture est *réflexive* : il existe une théorie universelle *U* qui peut représenter n'importe quelle théorie de réécriture *T* présentée de façon finie (incluant *U* elle-même) et tous termes *t* et *t'* de *T*, comme des termes *T*, *t* et *t'* dans *U*, de telle sorte à avoir l'équivalence suivante :

$$T \vdash t \rightarrow t' \Leftrightarrow U \vdash \langle \underline{T}, \underline{t} \rangle \rightarrow \langle \underline{T}, \underline{t'} \rangle .$$

Puisque *U* peut se représenter elle-même, une "tour réflexive" peut alors être obtenue, avec un nombre arbitraire de niveaux de réflexivité. Celle-ci rend possibles, non seulement la définition déclarative et l'exécution de stratégies de réécriture définies par l'utilisateur, mais aussi bien d'autres applications, dont par exemple une algèbre de modules extensible, d'opérations sur les modules paramétrées, qui serait définie et exécutée à l'intérieur de la logique.

Cette extension par réflexivité permet d'étendre les fonctionnalités de base de la logique pour considérer naturellement les applications orientées-objet.

5.5. Maude : un Langage à base de Logique de Réécriture

La programmation déclarative a eu le mérite de faciliter considérablement les aspects conceptuels de la tâche de programmation. Cependant, la logique sur laquelle se base généralement les langages déclaratifs ne prend pas en charge les caractéristiques inhérentes aux systèmes réels, notamment l'action et le changement.

Pour mettre en évidence ce constat, Meseguer a défini un langage, appelé MAUDE, basé sur une logique d'action qui est la logique de réécriture. Ce langage implémente et concrétise les différents concepts de la logique de réécriture. Dans la section suivante, nous présentons en détails ce langage.

Il existe bien d'autres langages à base de la logique de réécriture, les plus connus sont :

- **ELAN** [*BKK 97*] (France) fournit un environnement pour les systèmes de spécification et prototypage des déductions dans un langage basée sur des règles commandées par des stratégies. Son but est de soutenir la conception des preuves de théorèmes.
- **CAFEOBJ** est une nouvelle génération des langages de spécifications algébriques qui a été développée au Japon. L'équipe de CAFEOBJ est guidée par le prof. Kokichi Futatsugi, à JAIST (Japon), qui est un des fondateurs originaux d'OBJ2. CAFEOBJ est un langage de spécifications algébrique *exécutable*, et un successeur moderne d'OBJ, il est prévu pour la spécifications des systèmes réels, la vérification formelle des caractéristiques, prototypage rapide, ou même de programmation.

5.6. Le Système Maude

Maude est un langage déclaratif pur et un système performant, et aussi un environnement de développement. Les buts du projet de Maude est de supporter la spécification formelle exécutable, et d'élargir le spectre d'utilisation de la programmation déclarative et des méthodes formelles pour spécifier et réaliser des systèmes de haute qualité dans des secteurs comme : le génie logiciel, réseaux de communication, l'informatique répartie, bioinformatique, et le développement formel d'outils.

En outre, Maude a une collection d'outils formels supportant différentes formes de raisonnement logique pour vérifier des propriétés de programme comprenant :

• Un *Model-Checker* pour vérifier les propriétés temporelles linéaires de la logique (LTL) des modules d'états finis ;
• Un *Prouveur de Théorème* ("Theorem Prover") (ITP) : pour vérifier des propriétés des modules fonctionnels;
• Un *Analyseur Church-Rosser* pour assurer une telle propriété : pour les modules fonctionnels;
• Un *Analyseur de terminaison et un outil de Complétion* ("Knuth-Bendix") : pour les modules fonctionnels
• Un *Analyseur de Cohérence* : pour les modules systèmes.

Maude a été influencé d'une manière importante par OBJ3 [*JTJ 92*]. En particulier, le sous langage de la logique équationnelle de Maude contient essentiellement OBJ3 comme sous langage. La logique équationnelle de Maude est plus riche.

• *Maude est puissant :* Il peut modéliser presque n'importe quelle chose, c-à-d qu'il peut modéliser tout ce que nous pouvons écrire, parler, ou décrire avec la langue naturelle, autrement dit, Maude possède un large spectre d'application et peut être utiliser non seulement pour les spécifications exécutables et le prototypage, mais comme un vrai langage de programmation et de développement.

• *Maude est simple :* Sa sémantique est basée sur les principes fondamentaux de la théorie de catégorie, que l'utilisateur n'a pas besoin de connaître à l'initiation. Maude possède un ensemble de mots clés et de symboles, et quelques orientations et conventions à maintenir, et donc, les programmes Maude sont aussi simples et aussi faciles à comprendre.

• *Maude est concrêt :* Il est extrêmement abstrait, et sa conception permet une grande flexibilité dans la syntaxe des systèmes à spécifier.

• *Multi-paradigme :* Il combine la programmation déclarative et orientée objet.

Dans Maude, les spécifications non exécutables sont appelées "les théories", et leurs axiomes sont utilisés pour imposer des exigences formelles sur les modules programmes et sur les interfaces des modules paramétriques.

5.6.1. Core Maude

On appelle Core Maude l'interpréteur de Maude implémenté en C++ offrant toutes les fonctionnalités de base Maude, intégrant les modules systèmes et fonctionnels, et n'accepte que des hiérarchies des modules fonctionnels et systèmes non paramétré (voir Figure 1).
Core Maude intègre d'autres modules tel que : les modules prédéfinis des types de données et le module Model-Checker , la réflexion et la fonction méta-langage.

5.6.2. Full MAUDE

Full Maude est une extension de Core Maude écrite en langage Maude (notion de méta-level), et qui dote Maude avec un module algébrique puissant et extensible dans lequel les modules Maude peuvent être combinés pour donner d'autres modules complexes (voir Figure 1).
Aussi, les modules peuvent être paramétrés et instantiés avec des traces appelés vues, où les paramètres sont des théories définissent les conditions sémantiques pour l'instantiation correct, et les théories peuvent être elle même paramétrées. Les modules orientés objet (peuvent être paramétrés) supportent les objets, les messages, les classes et l'héritage. Enfin, on peut dire que les nouvelles notions supportées dans Full Maude sont : les modules orientés objet et la paramétrisation.

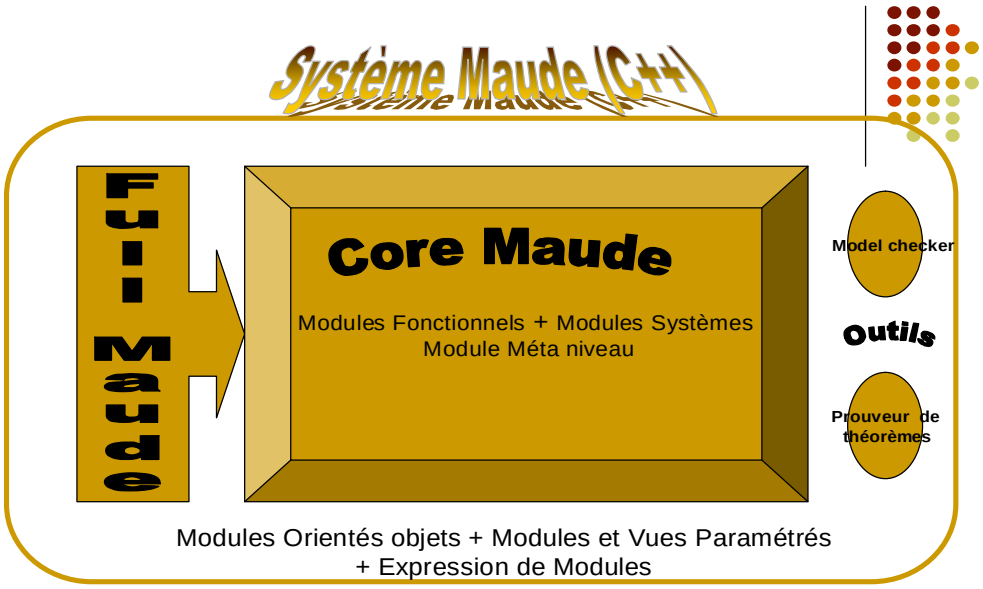


Figure 1 : Schéma général du système Maude

5.7. Les modules Maude

La logique de réécriture permet d'écrire des spécifications orientées propriétés de systèmes logiciels, sous la forme de théories qui peuvent comprendre une partie décrivant les changements d'états (par réécriture de termes). Selon qu'elle comprend ou non ces informations de réécriture, une théorie de la logique de réécriture sera considérée comme "théories système ou fonctionnelle". Maude étant un langage qui implémente et concrétise les différents concepts de la logique de réécriture, ses programmes (qui sont des théories de réécriture) regroupent deux types de modules : des *modules fonctionnels* qui servent à décrire des types abstraits de données et des *modules systèmes* qui servent à décrire des systèmes concurrents.
Les calculs concurrents dans Maude représentent des déductions dans la logique de réécriture. L'objectif principal de Maude est d'élargir le spectre d'utilisation de la programmation déclarative au-delà de ses applications statiques et platoniques.

5.7.1. Modules Fonctionnels

La représentation d'une théorie équationnelle (de la logique de réécriture) peut se faire, simplement sous la forme d'un module fonctionnel. Un module fonctionnel représente essentiellement des types de données, et les fonctions qui s' y appliquent. Comme évoqué précédemment, leurs équations doivent avoir la propriété de Church-Rosser et doivent terminer.
Dans ce cas, chaque étape de réécriture se fait par un remplacement d'égal à égal, jusqu'à ce qu'une valeur équivalente à celle de départ, et complètement évaluée, soit obtenue.

Exemple :
La figure 2 présente un exemple de module fonctionnel délimité par les deux mots clés : "*fmod*" et "*endfm*", il introduit trois types, ou sortes ("*sorts*"), simultanément : *Nat* pourrait représenter l'ensemble des entiers naturels, *NzNat* les entiers naturels non nuls et *Nat3* l'ensemble des entiers modulo 3. Pour cela, *NzNat* est déclaré comme une sous-sorte ("*subsort*") de *Nat*.

Le constructeur *zero* permet d'obtenir l'élément de base des *Nat*, et l'opération ("*op*") *s* servira à construire les autres, tout en ne générant que des *NzNat* ; l'opération *p*, au contraire, est l'opération inverse et pourrait avoir comme résultat un *Nat* n'appartenant pas à *NzNat* (c'est-à-dire *zero*). L'addition + est définie sur l'ensemble *Nat* tout entier, mais il est utile de préciser que le résultat de l'addition de deux éléments de *NzNat* reste dans *NzNat*.

Les trois premières équations ("*eq*") décrivent la façon dont interagissent les opérations, sur des variables ("*vars*") de sorte *Nat* ; nous retrouvons les propriétés usuelles des entiers, comme le prédécesseur du

successeur d'un nombre est ce nombre, l'addition de zéro à un nombre ne le change pas, et l'ajout du successeur d'un nombre à un autre est équivalent à demander le successeur de l'addition de ces nombres. En outre, les trois constantes *0*, *1* et *2* sont les éléments de base de *Nat3* (la spécification de Nat3 aurait pu faire l'objet d'un autre module fonctionnel).

La seule autre opération définie sur cet ensemble est à nouveau l'addition +, mais cette fois-ci elle est déclarée comme étant commutative ("*[comm]*") – d'autres propriétés auraient pu être ajoutées, comme l'associativité ou l'importance relative de l'opérateur par rapport aux autres.

Enfin, quatre équations, requérant une unique variable *N3*, achève la spécification du type de données *Nat3*. Chaque équation participe à la définition de classes d'équivalence de termes. Dans l'exemple de la figure 2, chacune peut être caractérisée par la valeur de l'entier équivalent.

```
fmod NUMBERS is
   sorts Nat NzNat Nat3 .
   subsort NzNat < Nat .
   op zero : -> Nat .
   op s_ : Nat -> NzNat .
   op p_ : NzNat -> Nat .
   op _+_ : Nat Nat -> Nat .
   op _+_ : NzNat NzNat -> NzNat .
   ops 0 1 2 : -> Nat3 .
   op _+_ : Nat3 Nat3 -> Nat3 [comm] .

   vars N M : Nat .
   var N3 : Nat3 .
   eq p s N = N .
   eq N + zero = N .
   eq N + s M = s (N + M) .
   eq (N3 + 0) = N3 .
   eq 1 + 1 = 2 .
   eq 1 + 2 = 0 .
   eq 2 + 2 = 1 .
endfm
```

Figure 2. : Exemple de module fonctionnel

Le calcul dans un module fonctionnel est une simplification équationnelle par application concurrente des équations qui sont orientées de gauche à droite. Ceci constitue la sémantique opérationnelle du module alors, que la sémantique dénotationnelle d'un module fonctionnel est l'algèbre initial parmi la classe des algèbres vérifiant les équations de celui-ci. Elle définit un type abstrait de données.

La façon de réécriture des modules fonctionnels se termine avec une valeur simple comme résultats. Dans de tels modules, chaque étape de la réécriture est une étape du remplacement des termes par des égaux, jusqu'à ce qu'on trouve la valeur équivalente, entièrement évaluée.

5.7.2. Modules Systèmes

Pour ce qui suit, nous n'interprétons plus un terme t comme expression fonctionnelle, mais comme état d'un système, et aussi une règle de réécriture t → t' comme une égalité, mais comme une transition d'état local. Tout changement d'état local peut intervenir en concurrence avec n'importe quel autre changement d'états locaux.

Et comme la logique de réécriture est une logique de changement concurrent d'état. Ses quatre règles de déduction réflexivité, transitivité, congruence, et remplacement, nous permettent de déduire tous les changements d'état concurrents et complexes qui peuvent survenir dans le système.

En Maude la spécification de tels systèmes est réalisée par les modules systèmes. Ils exécutent les calculs de réécriture concurrente, exactement de la même manière comme les modules fonctionnels. Cependant, leur comportement n'est pas fonctionnel.

Exemple :

Un exemple de module système délimité entre les mots clés "*mod*" et "*endm*" est présenté dans la figure 3. Il définit une opération ("?") de choix non déterministe sur les nombres naturels. Ce module système NAT-CHOICE réutilise (avec "*protecting*"), ou étend, la théorie prédéfinie *NAT* en réutilisant la sorte et les opérations définies dans ce module. Une seule opération *?* est ajoutée, qui sera spécifique aux nouveaux éléments ajoutés.

Enfin, les règles de réécriture ("*rl*") définies permettent de réécrire une expression de *Nat*, avec *?*, en choisissant de ne conserver qu'un de ses arguments.

```
mod  NAT-CHOICE is
   extending NAT .
   op _ ? _ : Nat  Nat → Nat .
   vars  N  M : Nat .
   rl N ? M ⇒ N .
   rl N ? M ⇒ M .
endm
```

Figure 3: Exemple de module système

Supposons que la sémantique d'un tel module (Figure 3) soit une algèbre initiale. Dans ce cas là, la déduction équationnelle nous permettra de conclure que M = N, c-à-d., tous les éléments de l'algèbre s'effondrent à un seul point. C'est pour ça, la déclaration "Extending NAT" dont la signification est que deux nombres naturels distincts du sous module NAT ne seront jamais identifié par les nouvelles équations introduites par le super module NAT-CHOICE. La sémantique mathématique d'un tel module ne peut pas être une algèbre initiale.

Il est clair qu'on utilisant l'instruction "Extending", on peut généralement ajouter des nouvelles données ou bien des règles pour enrichir la syntaxe du module importé sans autoriser la confluence. On peut utiliser deux autres instructions pour importer des modules, qui sont "Protecting" et "Including".
Si on utilise l'instruction "Protecting NAT" à la place "Extending NAT" dans le module NAT-CHOICE de la figure 3, le module NAT importé est un sous module de NAT-CHOICE mais aucune nouvelle donnée de type NAT ne sera ajoutée.
L'instruction "Including" peut ajouter des nouvelles données et équations sans aucune limitation

D'autre part, et comme dans OBJ3, la structure de type dans Maude est "order-sorted"; Il est possible aussi d'avoir la surcharge des symboles de fonction "Overloading" pour les opérations qui sont définies à différents niveaux de l'hiérarchie de la sorte.

5.7.3. Les Modules Orientés-objets en Maude

Les modules Orientés-Objets dans Maude sont aussi introduits. Bien qu'ils sont entièrement réductibles à des modules systèmes, ils conviennent pour une large variété d'applications, Ils méritent une syntaxe propre à eux, et ils sont basés sur une théorie logique des objets concurrents, qui a une bonne expression dans des termes de la logique de réécriture.

L'état d'un système orienté-objet appelé souvent "*configuration*", possède la structure d'un multi-ensemble formé d'objets et de messages et déclaré en utilisant la syntaxe du langage Maude grâce à l'opération :

Op _ : Configuration Configuration→ Configuration[ctor assoc comm id:null]

Un "*objet*" est alors noté ainsi:

$<O:C\ |a_1:v_1\ldots,a_n:v_n>$

O est l'identificateur de l'objet,

C est la classe correspondante de l'objet,

a_i , i = 1 à n, sont les attributes de l'objet,

v_i , i = 1 à n, sont les valeurs des attributs.

"*Les classes*" sont déclarées en utilisant la syntaxe suivante:

Class $C \mid a_1:s_1 \ldots, a_n:s_n$.

Où C est le nom de la classe et s_i sont les types ("sortes") respectifs des attributs a_i.

"*Un message*" est déclaré en utilisant le mot-clés: *msg* . Par exemple, une déclaration d'un message de données ("*Data*") envoyé et reçu par deux objets identifiés ("*Oid*") : émetteur et récepteur peut être déclaré comme suit :

Msg to : Oid Oid Data → *Msg*

Les interactions concurrentes entre les différents messages sont axiomatisées par des règles de réécriture conditionnelles dont la forme générale est donnée par:

crl[r] *:* $M_1 \ldots M_n$ $<O_1 : F_1 \mid atts_1> \ldots$
$<O_m : F_m|atts_m> \Rightarrow$
$<O_{i1} : F'_{i1}|atts'_{i1}> \ldots <O_{ik} :$
$F'_{ik}|atts'_{ik}>$
$<Q_1 : D_1|atts''_1> \ldots <Q_p : D_p|atts''_p>$
$M'_1 \ldots M'_q$
If *C*

où r est l'étiquette de la règle de réécriture, *Ms* sont les expressions de messages, $i_1,\ldots,i_k$, sont des nombre pris de l'intervalle des nombres : 1,…,*m*. Ici, la condition de la règle de réécriture est donnée par *C*.

Exemple :

L'exemple ci-dessous (figure 4) est repris de [ ] afin d'illustrer la syntaxe des modules orienté-objet.

On spécifie dans ce module 'accnt' delemité entre les deux mots clés 'omod' et 'endom' un compte bancaire, dans ce lui-ci on peut débiter ou retirer une somme d'argent de sorte 'nat' grase aux deux messages déclarés : 'credit' et 'debit' le transfert d'argent d'un compte donné à un autre peut être réalisé en spécifiant le message 'transfer-from-to-'. Des règles de réécritures sont associées à la descriptions faite sur le compte, la classe à laquelle appartient un objet de type compte bancaire possède un attribut représentant le montant de la somme qui s'y trouve.

```
omod accnt is
        protecting  nat.
        class accnt | bal: nat.
        msg credit debit : oid nat à  message.
        msg transfer- from - to - : nat oid oid à  message.
        vars a, b :oid.
        vars m,n,n': nat.
        rl [crdt] credit(a,m) <a; accnt | bal:n> è <a;accnt | bal : n+m>.
        crl [dbt] debit (a,m) <a; accnt | bal : n>  è  <a; accnt | bal : n-m> if n >=m.
        crl [trsf] transfer(m) from(a) to(b) <a; accnt | bal : N> <b; accnt | bal : n'>  è
        <a; accnt | bal : n-m> <b; accnt | bal : n+m> if n>= m.
endom.
```

Figure 4 : Exemple de module orienté-objets

5.7.4. Exemple récapitulatif

Afin d'illustrer plus en détail les spécifications des systèmes orientés objets nous indiquerons dans Maude un exemple simple, un magasin de location de voiture. Les règlements du système, particulièrement ceux qui règnent les processus de location, sont :

1. Des voitures sont louées pour un nombre spécifique de jours, après quoi elles devraient être retournées.
2. Une voiture peut être louée seulement si elle est disponible.
3. On ne permet aucun crédit ; les clients doivent payer l'argent comptant.
4. Les clients doivent faire un dépôt au temps de ramassage des frais de location estimés.
5. Les frais de location dépendent de la classe de voiture. Il y a trois catégories : économie, mi-taille, et voitures normales.
6. Quand une voiture louée est retournée, le dépôt est employé pour payer les frais de location.
7. Si une voiture est retournée avant la date due, le client paye seulement les nombres de jours où la voiture a été utilisée. Le reste du dépôt est remboursé au client.
8. Des clients qui renvoient une voiture louée après que sa date due soient facturées tous jours ou la voiture a été employés, avec un 20% additionnel pour chaque jour après la date due.
9. ne pas rendre la voiture à l'heure peut avoir comme conséquence la suspension des privilèges.

Commençons par les aspects statiques de ce système, c.-à-d., sa structure. Nous pouvons identifier trois classes principales, à savoir le magasin, les clients, et les voitures. Il y a trois genres spéciaux de clients (personnel, clients occasionnels, et clients réguliers), et trois genres de voitures (économie, mi-taille, et voitures normales).

Les clients peuvent louer des voitures. Ce rapport peut être représenté par une classe `de location` qui, en plus des références aux objets impliqués dans le rapport, a quelques attributs additionnels. Le système exige également un certain contrôle du temps, qui nous obtenons avec une classe représentant des calendriers qui fournit la date du jour et simule le passage du temps.

La classe de `client` a trois attributs, à savoir `argent comptant`, `dette`, et `suspendu` pour garder le disque, respectivement, de la quantité d'argent comptant que le client a actuellement, sa dette avec le magasin, et de s'il est suspendu ou pas. Une telle classe peut être définie par la déclaration suivante de Maude :

```
class Customer | cash : Nat, debt : Nat, suspended : Bool .
```

L'attribut `disponible de la voiture` de classe indique si la voiture est actuellement disponible ou pas, et le `taux` enregistre le taux de location quotidien. Nous modelons les différents types de voitures pour le loyer par trois sous-classes différentes, à savoir `EconomyCar`, `MidSizeCar` et `FullSizeCar`.

```
class Car | available : Bool, rate : Nat .
class EconomyCar .
class MidSizeCar .
class FullSizeCar .
subclasses EconomyCar MidSizeCar FullSizeCar < Car .
```

Chaque objet de `location de` classe établira un rapport entre un client et une voiture, dont les marques sont maintenues dans les attributs `client` et `voiture`, respectivement. En plus de ces derniers, la `location de` classe est également déclarée avec le `dépôt d'`attributs, le `pickUpDate`, et le `dueDate` pour stocker, respectivement, le montant d'argent à gauche comme dépôt par le client, la date lesoù la voiture est prise par le client, et la date l'où la voiture devrait être retournée au magasin.

```
class Rental | deposit  : Nat, dueDate : Nat, pickUpDate : Nat,
               customer : Oid, car     : Oid .
```

Etant donné l'utilisation simple que nous allons faire des dates, nous pouvons les représenter, par exemple, en tant que nombres normaux. Puis, nous pouvons avoir un objet de calendrier qui garde la date du jour et obtient accru par une règle de réécriture comme suit :

```
class Calendar | date : Nat .
rl [new-day] :
  < O : Calendar | date : F >
  => < O : Calendar | date : F + 1 > .
```

Nous ne nous inquiétons pas ici de la fréquence avec laquelle la date obtient accrue, des problèmes possibles de synchronisation dans un arrangement distribué, ni avec aucune autre question liée aux spécifications du temps. Quatre actions peuvent être identifiées dans l'exemple :

1. un client loue une voiture,
2. un client renvoie une voiture louée,
3. un client est suspendu pour être en retard en payant sa dette ou pour être en retard en renvoyant une voiture louée, et
4. un client paye (une partie de) sa dette.

La location d'une voiture par un client est indiquée par la `location de voiture de` règle ci-dessous, qui fait participer le client louant la voiture, la voiture elle-même (qui doit être disponible, c.-à-d., pas actuellement loué), et un objet de calendrier fournissant la date du jour. La location peut avoir lieu si le client n'est pas suspendu, c'est-à-dire, si sa marque n'est pas dans l'ensemble de marques des clients suspendus du magasin, et si le client a assez d'argent comptant pour faire le dépôt correspondant. Noter qu'un client pourrait louer une voiture pendant moins de temps où elle vraiment va l'employer sur le but, parce qu'elle n'a pas assez d'argent pour le dépôt, ou préfère faire un plus petit dépôt. Selon la description du système, le paiement a lieu quand le renvoi de la voiture, bien qu'il y ait un supplément pour les jours la voiture n'était pas réservé.

```
crl [car-rental] :
  < U : Customer | cash : M, suspended : false >
  < I : Car | available : true, rate : Rt >
  < C : Calendar | date : Today >
  => < U : Customer | cash : M - Amnt >
     < I : Car | available : false >
     < C : Calendar | >
     < A : Rental | pickUpDate : Today, dueDate : Today + NumDays,
       car : I, deposit : Amnt, customer : U, rate : Rt >
  if Amnt := Rt * NumDays /\ M >= Amnt
  [nonexec] .
```

Noter que ces attributs d'un objet qui ne sont pas appropriés pour un axiome n'ont pas besoin d'être mentionnés. Les attributs n'apparaissant pas dans le côté droit d'une règle maintiendront leurs valeurs précédentes non modifiées. Noter que les variables `A` et `NumDays` apparaissent dans le côté ou la condition droit de la règle mais pas dans son côté à gauche (c'est la raison pour laquelle cette règle est déclarée pour être `nonexec`). Noter aussi bien l'utilisation des attributs `client` et `voiture` dans les objets de la `location de` classe, qui rend explicite qu'un rapport de location est entre le client et la voiture indiqués par ces attributs.

Un client renvoyant une voiture tard ne peut pas être forcé de payer le montant total d'argent dû au temps de retour. Peut-être elle n'a pas un tel montant d'argent à ce moment-là. Le retour d'une voiture louée est indiqué par les règles ci-dessous. La première règle manipule le cas dans lequel la

voiture est retournée à l'heure, c'est-à-dire, la date du jour est plus petite ou égale à la date due, et donc le dépôt est plus grand ou égal à la quantité due.

```
crl [on-date-car-return] :
  < U : Customer | cash : M >
  < I : Car | rate : Rt >
```

```
  < A : Rental | customer : U, car : I, pickUpDate : PDt,
                 dueDate : DDt, deposit : Dpst >
  < C : Calendar | date : Today >
  => < U : Customer | cash : (M + Dpst) - Amnt >
     < I : Car | available : true >
     < C : Calendar | >
  if (Today <= DDt) /\ Amnt := Rt * (Today - PDt)
  [nonexec] .
```

Dans ce cas-ci, une partie du dépôt doit être remboursée. Nous pouvons voir que l'objet de `location` disparaît dans le côté droit de la règle, il est enlevé de l'ensemble de locations, et la disponibilité de la voiture est reconstituée.

La deuxième règle traite le cas dans lequel la voiture est retournée tard.

```
  crl [late-car-return] :
    < U : Customer | debt : M >
    < I : Car | rate : Rt >
    < A : Rental | customer : U, car : I, pickUpDate : PDt,
                   dueDate : DDt, deposit : Dpst >
    < C : Calendar | date : Today >
    => < U : Customer | debt : (M + Amnt) - Dpst >
       < I : Car | available : true >
       < C : Calendar | >
    if DDt < Today    *** it is returned late
       /\ Amnt := Rt * (DDt - PDt) + ((Rt * (Today - DDt)) * (100 + 20)) quo 100
    [nonexec] .
```

Dans ce cas-ci la dette du client est augmentée par la partie du de quantité non couvert par le dépôt.

Des dettes peuvent être satisfaites à tout moment, le seul état étant que le montant payé est entre zéro et le montant d'argent possédé par le client à ce moment-là.

```
crl [pay-debt] :
  < U : Customer | debt : M, cash : N >
  => < U : Customer | debt : M - Amnt, cash : N - Amnt >
  if 0 < Amnt /\ Amnt <= N /\ Amnt <= M
  [nonexec] .
```

Des clients qui sont en retard en renvoyant une voiture louée ou en payant leurs dettes « peuvent » être suspendus. La première règle traite le cas dans lequel un client a une dette en attente, et le second manipule le cas dans lequel un client est en retard en renvoyant une voiture louée.

```
crl [suspend-late-payers] :
  < U : Customer | debt : M, suspended : false >
  => < U : Customer | suspended : true >
  if M > 0 .

crl [suspend-late-returns] :
  < U : Customer | suspended : false >
  < I : Car | >
  < A : Rental | customer : U, car : I, dueDate : DDt >
  < C : Calendar | date : Today >
  => < U : Customer | suspended : true >
     < I : Car | >
```

```
  < A : Rental | >
  < C : Calendar | >
if DDt < Today .
```

5.8. Conclusion

Dans ce chapitre, nous avons présenté les concepts généraux de la logique de réécriture. C'est un cadre formel de raisonnement correct et d'analyse d'une grande variété de systèmes concurrents en particulier, ayant des états et évoluant en termes de transitions. Ses différents concepts sont implémentés à travers l'environnement Maude. Il permet d'une part, la programmation déclarative des systèmes modélisés et d'autre part, leurs prototypage et vérification. Dans la suite nous nous focalisons sur la spécification de la sémantique formelle et opérationnelle en logique de réécriture, d'un serveur de politiques, présenté dans le chapitre précédent.

CHAPITRE 6

6. CONCEPTION D'UN SERVEUR DE POLITIQUES

6.1. Introduction

Nous présentons dans cette section notre proposition de modèle d'information pour représenter les notions de qualité de service dans le cadre de la logique de réécriture. L'objectif principal du modèle est de représenter d'une part l'état actuel de l'environnement du réseau, et d'autre part, les informations nécessaires à la mise en place de nouveaux services, ou des nouveaux profils pour des services déjà existants.

Nous définissons dans les sous-sections suivantes les hypothèses prises en compte pour la définition du modèle et la terminologie utilisée, qui correspondent à l'ensemble de classes définies par notre modèle.

6.2. Le modèle d'information pour les politiques de QoS

Nous intégrons à travers cette section un modèle d'information du type CIM pour les politiques de Qos dans le cadre de la logique de réécriture. Cela fournit un moyen permettant de partager l'information de manière uniforme dans une institution en capturant l'ensemble des informations nécessaires à la mise en place de la solution de contrôle et de gestion à considérer.

CIM est un modèle orienté objet qui représente et organise l'information dans un environnement géré. Les *objets* CIM comprennent les ordinateurs, les périphériques, les fichiers, les logiciels, les utilisateurs, les organisations, les réseaux, etc.

En complément de la description des données brutes (composants), CIM permet la définition d'associations et de méthodes. Les *associations* décrivent les relations entre objets, les dépendances, les connexions, etc. CIM définit une série d'objets en respectant un ensemble de classes de base et d'associations.

Le modèle conçu hérite en plus des avantages de l'approche objet (abstraction, héritage, etc), ceux de la logique de réécriture comme formalisme de spécification formelle.

Les schémas CIM sont décrits au format MOF (*Managed Object Format*). Il s'agit d'un format de description de la structure et du contenu du modèle. Dans notre cas, nous retenons de ce format essentiellement les deux éléments suivants :

1. un méta-modèle décrivant les composants du modèle,
2. un schéma de nommage des composants gérés,

Les autres éléments du modèle de l'information CIM (langage support pour la spécification des objets gérés, modèle de l'information de référence servant de base à la spécification de nouveaux composants gérés, ensemble de recommandations sur l'intégration de modèles de l'information) se retrouvent implicitement dans notre formalisation.

Le méta-modèle définit les composants du modèle qui servent de briques de base à la construction des spécifications.

Ces composants forment les éléments de base de la structure de l'information dans l'approche CIM. Construire un modèle de l'information consiste à définir un schéma dans l'approche CIM et une théorie orientée objet en logique de réécriture. Ce dernier est composé de classes (objets gérés, associations, indications), d'instances et de qualifieurs. Tout composant au sein d'un schéma doit avoir un nom unique. L'identification d'une instance se fait via les valeurs des attributs de celle-ci définis comme clefs d'identification.

Afin de permettre la construction de nouvelles spécifications et de fournir un support textuel pour la formalisation des modèles de l'information, le langage Maude à base de logique de réécriture constitue un bon support. Ce langage fournit une syntaxe et une sémantique au méta-modèle CIM permettant de spécifier tout composant d'un modèle de l'information (qualifieur, classe, association, etc.). L'utilisation des outils de l'environnement Maude se fait de façon naturelle et est d'un très grand intérêt pour la saisie, l'analyse et le prototypage du méta modèle CIM.

Le caractère de *réflectivité* dans Maude permet de générer automatiquement des « générateurs de modèles CIM»

Notons ici que l'approche adoptée est générale, elle est appliquée à n'importe quel modèle d'information du type CIM, en particulier le modèle PCIM considéré par l'IETF comme extension de CIM aux techniques de gestion des politiques.

PCIM définit, en logique de réécriture, un modèle de description des politiques quels que soient leurs domaines d'application. Il est fondé sur deux hiérarchies de classes :

- les classes structurelles, qui forment les éléments de base des politiques,
- les classes d'association qui déterminent les relations entre les éléments.

Le réseau est considéré comme une machine à états, c'est-à-dire une théorie système dans laquelle on spécifie les états du réseau. Les politiques servent à contrôler les changements d'état en identifiant l'état courant et en définissent les transitions possibles.

Les politiques forment un ensemble de conditions vraies ou fausses qui peuvent être composées pour réaliser une politique plus complexe. Ces politiques forment à leur tour un ensemble d'actions associées aux conditions qui modifient la configuration d'un ou de plusieurs éléments et qui

introduisent des ordres d'exécution comme la priorité des flux ou l'ordonnancement des paquets dans le réseau.

L'expression de PCIM en logique de réécriture permet de le rendre plus flexible en permettant aux différents domaines d'homogénéiser leurs concepts. Les principales extensions concernant une meilleure gestion des priorités, l'ajout de règles simplifiées fondées sur les variables, la possibilité d'avoir des règles conditionnées par d'autres règles se font de façon assez naturelle.

Enfin, remarquons que le travail de définition des politiques et d'administration est pris en charge au sein de l'IETF par le groupe de travail POLICY. Le but de ce groupe est de proposer des solutions pour supporter la QoS. Le travail de spécification de l'IETF s'appuie sur le travail réalisé par plusieurs autres organismes, et notamment le DMTF *(Desktop Management Task Force)* et le DEN (*Directory Enabled Networks*). Dans notre initiative reste indépendante des constats faits par ces organismes.

Exemple

Supposant que l'on doit fournir un service vidéo Svideo. Cette opération doit être mise en application comme politique : Fournissez le service Svideo visuel pour les utilisateurs autorisés entre les points autorisés, mais seulement à des heures convenues.

En traduit cela par la règle suivante :

```
IF user IN ApprovedUsers
AND service IN VideoServices
AND source IN VideoSources
AND destination IN VideoDestinations
AND time IN ApprovedTimePeriods
THEN provide JitterFreeMPEG2
```

On peut voir dans cet exemple les différentes conditions qui exécutent la règle de politique

```
IF the user is a member of an approved group (ApprovedUsers)
that are authorized to have this service)
AND the service requested is one supported (VideoServicesgroup)
AND the source of the request is approved (in the VideoSources
group or has been authenticated)
AND the destination is approved (in the VideoDestinations group
or has been authenticated)
AND the time requested is OK (in ApprovedTimePeriods)
```

Ici, les types de condition de politique sont :

- l'utilisateur, le service, la source, la destination, et le temps

Les éléments d'état de politique sont :

- ApprovedUsers, VideoServices, VideoSources, VideoDestinations, et ApprovedTimePeriods, qui est tous des groupes prédéfinis d'objets.

L'action de la règle de politique est :

```
IF the conditions are satisfied
THEN provide the user with video having a QoS defined by the Svideo Service
```

Si les conditions sont satisfaites alors on peut fournir le service demandé

6.3. Le schéma fonctionnel du serveur de politiques de QoS

Nous aborderons cette section par l'intégration et la gestion de la politique de l'exemple précédant en Maude :

Commençons par les aspects statiques de ce système, c.-à-d., sa structure. Nous pouvons identifier deux classes principales, à savoir le service demandé et les clients. Il y a deux genres spéciaux de clients (clients occasionnels, clients réguliers), et deux genres de sévices (service de qualité normale, service de qualité supérieur).

Les clients peuvent demandé un service. Ce rapport peut être représenté par une *classe de* `videoservices` qui, en plus des références aux objets impliqués dans le rapport, à quelques attributs additionnels. Le système exige également un certain contrôle du temps, qui nous obtenons avec une classe représentant des calendriers qui fournit la date du jour et simule le passage du temps.

La *classe client « ApproverUsers »* a trois attributs, à savoir `argent comptant`, abonnement valide et abonnement `suspendu`. Une telle classe peut être définie par la déclaration suivante de Maude :

```
Class ApprovedUsers / cash : Nat, AbVa : Bool, AbSus : Bool.
```

Nous définition les différents types de service pour les clients par deux sous-classes différentes, à savoir « servicenormale, servicesupérieur », avec biens sure la contrainte de temps « times » et la disponibilité du service.

```
Class Svideo / available : Bool, Times : Nat
Class Servicenormale.
Class Servicesupérieur
Subclasses Servicenormale Servicesupérieur < Svideo
```

Chaque élément classe videoservice établira un rapport entre le client et le type de service demander, En plus de ça, la le videoservice est également déclarée avec des `d'`attributs comme le `pickUptimes`, et le `duetimes` pour respectivement, déterminé le début du service et la fin de ce dernier.

```
Class VideoServices / PickUpTimes : Nat, DueTimes : Nat
                      ApprouvedUsers : Oid, Svideo : Oid
```

Etant donné l'utilisation simple que nous allons faire des dates, nous pouvons les représenter, par exemple, en tant que nombres normaux. Puis, nous pouvons avoir un objet de calendrier qui garde la date du jour et obtient accru par une règle de réécriture comme suit :

```
Class calendar / date : Nat.
Rl [new-day] :
  < 0 : calendar / date : F >
    < 0 : calendar / date : F + 1 >
```

Trois actions peuvent être identifiées dans l'exemple :

5. un client demande un service,
6. la fin du service pour un client ,
7. suspendre un service pour un client pour défaut d'abonnement

La demande d'un service par un client et indiqué par la règle ci-dessus. La disponibilité du service reste un élément indispensable pour satisfaire la demande du client.

Ci cette élément ai valider reste a vérifier les closes du contra, entre autre la date du début et la fin de ce service et le type d'abonnement souscrit

```
Crl [svideo-service] :
  < U : ApprouvedUsers / cash : M, suspended : false >
  < I : Svideo /  available : true, Times : rt >
  < C : calendar / data : today >
    < U : ApprouvedUsers / cash : M – amnt >
    < I : Svideo /  available : false >
    < C : calendar / >
    < A : videoservice / PickUpTimes : today, DueTimes :
          today + numdays, svideo : I, Approvedusers : U,
           times : rt >
  if Amnt : rt * numdays / M : Amnt
    [nonexec]
```

Noter que ces attributs d'un objet qui ne sont pas appropriés pour un axiome n'ont pas besoin d'être mentionnés. Les attributs n'apparaissant pas dans le côté droit d'une règle maintiendront leurs valeurs précédentes non modifiées. Noter que les variables `A` et `NumDays` apparaissent dans le côté ou la condition droit de la règle mais pas dans son côté à gauche (c'est la raison pour laquelle cette règle est déclarée pour être `nonexec`). Noter aussi bien l'utilisation des attributs ApprovedUsers et Svideo dans les objets de la classe vidéoservice, qui rend explicite qu'un rapport de service entre le client et le type de service.

La règle suivante étudie le cas de la fin d'un service pour un client a la date et l'heure prédéfinie

```
Crl [servicefin] :
   < U : ApprouvedUsers / cash : M >
   < I : Svideo / times : At >
   < A : Videoservice / ApprouvedUsers : U, Svideo : I ?
                        pickuptimes : pdt, duetimes : today >
   < C : calendar / date : today >
        < U : ApprouvedUsers / cash : (M + dpst) - amnt >
        < I : Svideo / available : true >
        < C : calendar / >
   If (today = ddt) / Amnt = rt * (today –pdt)
   [nonexec]
```

Dans le dernier cas nous considérons la fin d'un service pour un client pour défaut d'abonnement

```
Crl [suspend-service]
   < U : ApprouvedUsers / debt : M, suspended : false >
     < U : ApprouvedUsers / suspended : trus >
   If M > 0.
```

6.4. Conclusion

Nous avons présenté dans ce chapitre notre modèle d'information. Notre motivation principale est de pouvoir faciliter le déploiement des services réseaux dynamiquement. Ce déploiement dynamique de service est basé sur la caractérisation des besoins de service et l'état du réseau. Les profils de service identifient une configuration possible du service (quantité de ressources, de paramètres de configuration, etc..). Les profils de déploiement identifient une demande de service d'un client. Ainsi, en utilisant cette information notre modèle fournit une vision *en temps réel* sur l'état du réseau et sur l'acceptation ou pas des nouveaux services dans le réseau.

Cet outil permet d'établir, pour toute demande de nouveau service, l'ensemble de composants nécessaires avec leur état (localisation, disponibilité de ressources, etc.), ainsi que leur relation d'interdépendance. Cette cartographie permettra à un gestionnaire de prendre des décisions de déploiement efficaces.

CHAPITRE 7

7. CONCLUSION ET PERSPECTIVES

La Qualité de Service ou QoS déferle sur les réseaux. C'est aujourd'hui le sujet le plus confus dans ce domaine, comportant de multiples définitions pour répondre à de multiples objectifs. Cette thèse tente de faire le point sur cette notion dans l'Internet, dans l'optique d'aider l'administrateur réseau à se retrouver dans la jungle des mécanismes avancés par les constructeurs. En effet, la Qualité de Service dans l'Internet est encore un vaste chantier, à tel point qu'il est difficile d'en faire la synthèse! Aujourd'hui de multiples groupes de travail de l'IETF travaillent dans des domaines touchant la Qualité de Service, Certains tiennent compte de nouveaux développements dans les réseaux tels que la spécification 802.1p ou les techniques de "label switching", tandis que d'autres se focalisent sur les politiques de contrôle dans les réseaux ou la définition de nouveaux protocoles de routage. Tous ces travaux sont néanmoins étroitement imbriqués et permettent de construire progressivement les morceaux du puzzle Qos adapté à l'Internet.

Pour plus de détails sur les mécanismes ou pour comprendre les enjeux économiques actuels de la Qualité de Service, nous avons regroupé quelques définitions relatives à la Qualité de Service (QoS), nous présentons les critères ou indicateurs généralement retenus pour caractériser les performances d'un réseau et assurer un service de bonne qualité. Ensuite, nous décrivons quelques mécanismes et approches permettant d'implémenter la Qualité de Service dans l'Internet sous forme de services différenciés, ainsi que l'architecture "Integrated Services" développée par l'IETF. Pour vérifier que les conditions de Qualité de Service sont respectées, il faut pouvoir mesurer la QoS. Mais vite fais nous avons constaté les limites de ces approches surtout, car parallèlement au modèle de réservation de ressources et de services intégrés posant des problèmes de déploiement à large échelle. L'objet de ces travaux s'organise autour d'une idée, utiliser la notion de politique pour construire des modèles d'évaluation de la qualité de service dans les réseaux multiservices. L'approche des serveurs de politiques permet d'utilisés pour le contrôle d'admission ou l'ingénierie de trafic. Nous avons proposé dans un travail antérieur à celui-ci, de généraliser cette modélisation à l'estimation de la qualité de service le long d'un chemin. Nous avons élaboré un schéma original de prédiction de la QoS qui s'appuie sur une gestion de la qualité de service à base de serveur de politique afin de contrôler d'admission sur paramètres. L'utilisation de modèles connexionnistes pour le contrôle d'admission n'est en soi pas nouvelle, elle présente beaucoup de limites. Les serveurs de politique ne prennent pas directement la décision d'acceptation, mais fournissent une évaluation des critères de QoS à partir d'une description des trafic s'entrants. Ainsi la décision d'admission peut être prise sur la base des estimations de l'administrateur du réseau, en considérant les contrats de chaque connexion et éventuellement à partir de règles définissant une politique de contrôle d'admission. Ainsi le modèle suggéré s'inscrit dans le contexte du contrôle d'admission à

base de règles. Par ailleurs, les modèles d'évaluations sur chaque routeur ne se basent plus sur la description donnée a priori à l'entrée du réseau mais sur une estimation de ce qu'il sera en arrivant au niveau du routeur. Pour cela, les serveurs de politique doivent estimer, en plus des critères de QoS, les caractéristiques du trafic de sortie. Celles-ci seront alors utilisées par le routeur suivant pour faire ses estimations.

Dans le présent travail, nous avons repris le modèle informel suggéré et nous l'avons intégré dans un cadre de raisonnement formel, à base de la logique de réécriture qui est un cadre formel de raisonnement correct et d'analyse d'une grande variété de systèmes concurrents en particulier, ayant des états et évoluant en termes de transitions. Ainsi, la complexité du serveur de politique est alors mieux gérée : une compréhension approfondie par la mise en évidence des ambiguïtés laissées dans l'ombre par les spécifications informelles, l'utilisation d'outils de preuve et d'évaluation symbolique existants autour de cette logique. En outre, cette modélisation constitue une première application de la logique de réécriture pour l'expression et la vérification des propriétés non fonctionnelles (QoS) d'un système.

Notre approche de conception est inspirée de celle basée LDAP. Dans la première étape, un modèle d'information pour les politiques de QoS de type QPIM est défini en utilisant principalement les modèles orienté objet de Maude. Dans une deuxième étape, nous avons proposé un schéma fonctionnel formel d'un serveur de gestion de politiques de QoS en exploitant les caractéristiques syntaxiques et sémantiques de Maude.

Les perspectives sont nombreuses car nos travaux ne représentent qu'un premier pas vers la modélisation par les serveurs de politique de la qualité de service. Nous avons en effet considéré des trafics sporadiques idéalisés de politique de service et nous avons ignoré certains mécanismes, complexes, qui entrent en jeu dans les réseaux comme les algorithmes de rejet des paquets (e.g. WRED, *Weighted Random Early Detection*), la réémission par les sources de ces paquets perdus, le contrôle de trafic réactif (e.g. TCP), etc. Les modèles d'évaluation de la QoS que nous avons construits sont par conséquent spécifiques aux modèles que nous avons considérés.

Mais c'est en grande partie l'intérêt des modèles d'apprentissage de pouvoir s'adapter spécifiquement aux observations qui constituent leur base d'apprentissage. L'étape suivante sera donc de considérer la qualité de service de systèmes réels.

Enfin, l'utilisation de l'évaluation par les serveurs de politique pour considérer plus finement les exigences de qualité de service des clients dans les problèmes d'optimisation dans les réseaux est prometteuse. Ils sont un bon compromis entre les méthodes de simulation, plus réalistes mais extrêmement coûteuses et l'approximation, peu coûteuse mais peu réaliste, notamment dans les réseaux à différentiation de service. La prise en compte de cette qualité de service au travers de

contraintes de bout en bout peut permettre de considérer véritablement les exigences des clients au niveau de routage comme nous l'avons vu, mais aussi lors du dimensionnement du réseau voire de la conception même du réseau.

D'autre part, l'intégration du modèle d'évaluation de la QoS que nous avons construit dans le cadre de la logique de réécriture ouvre la voie à d'autres thèmes de recherches, en particulier l'exploitation des caractéristiques syntaxiques et sémantiques de cette logique :

- L'expression et la vérification des propriétés non fonctionnelles des systèmes informatiques en général.
- La formulation correcte des métas politiques.
- La structuration des politiques de QoS selon leur sémantique et régler ainsi le problème de conflit sémantique qui peut se poser dans certaine situations.

CHAPITRE 8

8. BIBLIOGRAPHIE

[Agoulmine 2002] Agoulmine Nazim, *"Gestion de la QoS dans Internet via l'approche PBM"*, DNAC 2002 (école d'hiver), 14-18 janvier 2002, Sainte - Luce, Martinique,

[Agoulmine 2003] Agoulmine Nazim, Cherkaoui Omar, *"Pratique de la gestion de réseau"*, 2003, Eyrolles, ISBN:2-212-11259-9.

[Baader et Nipkow 1998] Franz Baader et Tobias Nipkow, *Term Rewriting and All That*, Cambridge University Press, février 1998.

[Beker, Davie, 2000] Fred Baker, Bruce Davie, Francois Le Faucheur, Carol Iturralde, *RSVP Reservations Aggregation*, 03/13/2000, draft-ietf-issll-rsvp-aggr-02.txt, travail en cours

[Callon, Swallow, 1999] Ross Callon, George Swallow, N. Feldman, A Viswanathan, P. Doolan, A. Fredette, *A Framework for MPLS*, 09/22/1999, draft-ietf-mpls-framework-05.txt, travail en cours

[Clark, fang, 1998] D. Clark, W. Fang, *Explicit Allocation of Best Effort Packet Delivery Service*, IEEE/ACM Transactions on Networking, August 1998, Vol. 6, No. 4, pp. 362-373.

[Clavel et al. 1999] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer et J. F. Quesada, *Maude: Specification and Programming in Rewriting Logic*, Maude system documentation, Computer Science Laboratory, SRI International, 8 mars 1999.

[Cohen 2003] "Policy QoS Information Model", Ron Cohen, John Strassner, Bob Moore, Yoram Snir, Yoram Ramberg, draft-ietf-policy-qos-info-model-05.txt

[CRU 2003] CRU LDAP Forum, http://www.cru.fr/ldap/

[Cui et al., 2004] Cui, Y., Li, B., and Nahrstedt, K. (2004). oStream: Asynchronous streaming multicast in application-layer ovelay networks. *IEEE Journal on Selected Areas in Communications*, 22(1):91–106.

[Damianou 2001]Nicodemos Damianou, Naranker Dulay, Emil Lupu, Morris Sloman," The ponder policy specification language", janvier 2001, (www.doc.ic.ac.uk/~mss/Papers/Ponder-Policy01V5.pdf)

[Damianou, 2002] Damianou, N. C. A Policy Framework for Management of Distributed Systems. PhD thesis, Imperial College of Science, Technology and Medicine, University of London, Department of Computing, 2002.

[Damianou, Dulay, 2002] Damianou, N., Dulay, N., Lupu, E., Sloman, M., and Tonouchi, T. Tools for Domain-based Policy Management of Distributed Systems. In IEEE/IFIP Network Operations and Management Symposium (NOMS2002) (Apr. 2002), pp. 213_ 218.

[Damianou, Dulay, 2005] Damianou, N., Dulay, N., Lupu, E., and Sloman, M. Ponder : A Language for Specifying Security and Management Policies for Distributed Systems. The Language Speci_cation - Version 2.3. Imperial College Research Report DoC 2000/1, Oct. 2000. http ://www.doc.ic.ac.uk/ ncd/policies/_les/ accédé en Avril 2005.

[Delot 2000] Thierry Delot, "Lightweight Directory Access Protocol (LDAP)", présentation de LDAP, Laboratoire PRiSM de l'université de Versailles. http://www-adele.imag.fr/~donsez/cours/ldapdelot.pdf

[Donsez 2002] Didier Donsez, "*Annuaires (Directory)*", cours de l'université de Grenoble, version du 19/03/2003, http://www-adele.imag.fr/~donsez/cours/

[Durhan 2003] Information Model for Describing Network Device QoS Datapath Mechanisms, David Durham, John Strassner, Walter Weiss, Andrea Westerinen, Bob Moore, draft-ietf-policy-qos-device-info-model-10.txt

[Festor 2000] Olivier Festor, Nizar Ben Youssef " *WBEM* ", rapport de recherche avril 2000

[Festor 2001] Olivier Festor, Nizar Ben Youssef "*Représentation des spécifications CIM dans un Serveur d'annuaire LDAP* ", rapport technique, juillet 2001, (ftp://ftp.inria.fr/INRIA/publication/publi-pdf/RT/RT-0250.pdf)

[Fidalgo 2002] FIDALGO joseane f., SADOK djamel f. h., KELNER judith, FIDALGO robson do n., *"A Simple Performance Policy Management Environment ",* Net-Con 2002 (Network Control and Engineering for QoS, Security and Mobility (IFIP and IEEE publication)), 22-25 octobre 2002, Paris, France.

[Group, 2003] Group, O. M. UML 2.0 OCL Speci_cation, Oct. 2003. http ://www.omg.org/docs/ptc/03-10-14.pdf, accÈdÈ en mai 2005.

[Le Boudec, 1997] Jean-Yves Le Boudec - *An Introduction to Network Calculus* - http://ica1www.epfl.ch/ netCalSlides.pdf

[Lupu, Sloman, 1997] Lupu, E., and Sloman, M. Con_ict Analysis for Management Policies. In Proceedings of the 5th IFIP/IEEE International Symposium on Integrated Network management IM'97, San Diego, CA (1997).

[Lupu, Sloman, 1999] Lupu, E. C., and Sloman, M. Con_icts in Policy-Based Distributed Systems Management. IEEE Transactions on Software Engineering 25, 6 (Nov. 1999), 852_869.

[Martí-Oliet et Meseguer 1993] N. Martí-Oliet et J. Meseguer, *Rewriting logic as a Logical and Semantic Framework*, rapport technique, SRI International, Computer Science Laboratory, Menlo Park, août 1993.

[Martí-Oliet et Meseguer 1996] N. Martí-Oliet et J. Meseguer, Rewriting logic as a Logical and Semantic Framework, *Electronic Notes in Theoretical Computer Science*, 4, 1996.

[Mélin 2001] Jean-Louis Mélin, "*Qualité de Service sur IP*", livre publié en février 2001 par Eyrolles.

[Mes 02] J. Meseguer, *« Rewriting Logic Revisited»* Slides of tutorial presented at WRLA 2002, Pisa, Italy, September 2002.

[Meseguer 1998] José Meseguer, "Research Directions in Rewriting Logic", *Computational Logic*, NATO Advanced Study Institute, U. Berger et H. Schwichtenberg éditeurs, Springer-Verlag, 1998.

[Mirtain 2003] Ressources Inria Laurent Mirtain.
http://www-sop.inria.fr/semir/personnel/Laurent.Mirtain/LDAP.html

[Ohlebusch 2002] Enno Ohlebusch, *Advanced Topics in Term Rewriting*, Springer-Verlag, XV, 2002.

[RFC 2205] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin. *Resource ReSer-Vation Protocol (RSVP) -- Version 1 Functional Specification.* Septembre 1997.

[RFC 2205] S. Shenker, C. Partridge, R. Guerin. *Specification of Guaranteed Quality of Service*. Septembre 1997.

[RFC 2211] J. Wroclawski. *Specification of the Controlled-Load Network Element Service*. Septembre 1997.

[RFC 2330] V. Paxson, G. Almes, J. Mahdavi, M. Mathis,.*Framework for IP Performance Metrics.* Mai 1998.

[RFC 2474] K. Nichols, S. Blake, F. Baker, D. Black., *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers.* Decembre 1998.

[RFC 2475] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss. *An Architecture for Differentiated Service.* Decembre 1998.

[RFC 2597] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski., *Assured Forwarding PHB Group.* Juin 1999.

[RFC 2598] V. Jacobson, K. Nichols, K. Poduri. *An Expedited Forwarding PHB.* Juin 1999.

[RFC 2698] J. Heinanen, R. Guerin, *A Two Rate Three Color Marker*, Juin 1999.

[RFC 2044] F. Yergeau,, "*UTF-8, a transformation format of Unicode and ISO 10646*", octobre 1996, (www.faqs.org/rfcs/rfc2044.html).

[RFC 2212] S. Shenker, C. Partridge, R. Guerin , "*Specification of Guaranteed Quality of Service*", septembre 1997, (www.faqs.org/rfcs/rfc2212.html).

[RFC 2247] S. Kille, M. Wahl, A. Grimstad, R. Huber, S. Sataluri "*Using Domains in LDAP/X.500 Distinguished Names*", janvier 1998, (www.faqs.org/rfcs/rfc2247.html).

[RFC 2251] M. Wahl, T. Howes, S. Kille, "*Lightweight Directory Access Protocol (v3)*", decembre 1997, (www.faqs.org/rfcs/rfc2251.html).

[RFC 2251] M. Wahl, T. Howes, S. Kille, "Lightweight Directory Access Protocol (v3)", décembre 1997, (www.faqs.org/rfcs/rfc2251.html).

[RFC 2252] M. Wahl, A. Coulbeck, T. HowesS. Kille, "*Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions*:", decembre 1997, (www.faqs.org/rfcs/rfc2252.html).

[RFC 2253] M. Wahl, S. Kille , T. Howes , " *Lightweight Directory Access Protocol (v3):UTF-8 string representation* of Distinguished Names ", decembre 1997, (www.faqs.org/rfcs/rfc2212.html).

[RFC 2254] T. Howes, " *The String Representation of LDAP Search Filters*", decembre 1997, (www.faqs.org/rfcs/rfc2254.html).

[RFC 2255] T. Howes, M. Smith, " *The LDAP URL Format*", decembre 1997, (www.faqs.org/rfcs/rfc2255.html).

[RFC 2256] M. Wahl, "A *Summary of the X.500(96) User Schema for use with LDAPv3* ", decembre 1997, (www.faqs.org/rfcs/rfc2256.html).

[RFC 2307] L. Howard , " *An Approach for Using LDAP as a Network Information Service*", Mars 1998, (www.faqs.org/rfcs/rfc2307.html).

[RFC 2377] A. Grimstad, R. Huber , S. Sataluri , M. Wahl " *Naming Plan for Internet Directory-Enabled Applications*", septembre 1998, (www.faqs.org/rfcs/rfc2377.html).

[RFC 2474] K. Nichols, S. Blake, F. Baker, D. Black, *"Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*", décembre 1998, (www.faqs.org/rfcs/rfc2474.html).

[RFC 2589] Y. Yaacovi, M. Wahl , T. Genovese , " *Lightweight Directory Access Protocol (v3) : Extensions for Dynamic Directory Services*", Mai 1999, (www.faqs.org/rfcs/rfc2589.html).

[RFC 2596] M. Wahl, T. Howes, "*Use of Language Codes in LDAP* ", Mai 1999, (www.faqs.org/rfcs/rfc2596.html).

[RFC 2748] D. Durham, J. Boyle , R. Cohen, S. Herzog, R. Rajan, A. Sastry, "*The COPS (Common Open Policy Service) Protocol*", janvier 2000, (www.faqs.org/rfcs/rfc2748.html).

[RFC 2753] R. Yavatkar, D. Pendarakis, R. Guerin, "*A Framework for Policy-based Admission Control*", janvier 2000, (www.faqs.org/rfcs/rfc2753.html).

[RFC 2829] M. Wahl, H. Alvestrand , J. Hodges , R. Morgan "*Authentication Methods for LDAP* ", Mai 2000, (www.faqs.org/rfcs/rfc2829.html).

[RFC 2830] J. Hodges, R. Morgan , M. Wahl , " *Lightweight Directory Access Protocol (v3) : Extensions for Transport Layer Security*", Mai 2000, (www.faqs.org/rfcs/rfc2830.html).

[RFC 2849] G. Good, "*The LDAP Data Interchange Format (LDIF) - Technical Specification*", juin 2000 1997, (www.faqs.org/rfcs/rfc2849.html).

[RFC 2891] T. Howes, M. Wahl , A. Anantha , "*LDAP Control Extension for Server Side Sorting of Search Results*", august 2000, (www.faqs.org/rfcs/rfc2891.html).

[RFC 3060] B. Moore, E. Ellesson, J. Strassner, A. Westerinen, "Policy Core Information Model - Version 1 Specification", février 2001, (www.faqs.org/rfcs/rfc3060.html).

[RFC 3062] K. Zeilenga , "*LDAP Password Modify Extended Operation*", fevrier 2001, (www.faqs.org/rfcs/rfc3062.html).

[RFC 3112] K. Zeilenga , "*LDAP Authentication Password Schema*", Mai 2001, (www.faqs.org/rfcs/rfc3112.html).

[RFC 3198] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, S. Waldbusser, "Terminology for Policy-Based Management", novembre 2001 (www.faqs.org/rfcs/rfc3198.html)

[RFC 3377] J. Hodges , R. Morgan, "*Lightweight Directory Access Protocol (v3) : technical Specification* ", septembre 2002, (www.faqs.org/rfcs/rfc3377.html).

[RFC 3384] E. Stokes, R. Weiser , R. Moats , R. Huber, "*Lightweight Directory Access Protocol (version 3) Replication Requirements* ", octobre 2002, (www.faqs.org/rfcs/rfc3384.html).

[RFC 3460] B. Moore, "Policy Core Information Model (PCIM) Extensions", janvier 2003, (www.faqs.org/rfcs/rfc3460.html).

[Rizcallah 2002] RIZCALLAH marcel, *"Annuaires LDAP", "2ème édition",* 2002, Eyrolles, ISBN:2-212-11033-2.

[Stardust 1999] Stardust.com, Inc, *Introduction to QoS Policies,* juillet 1999

[Strassner 1998] Policy Framework Definition Language,John Strassner/Stephen Schleimer, draft-ietf-policy-framework-pfdl-00.txt "CIM Tutorial", *DMTF*, Mars 1999 .*http://www.csd.uwo.ca/faculty/hanan/cs434/qos1.pdf*)

http://www.dmtf.org/education/cimtutorial.php

[Yu, 2003] Yu, W. (2003). Peer-to-peer approach for global deployment of voice over IP service. In *Proceedings of the 12th International Conference on Computer Communications and Networks (ICCCN'03)*, Dallas, U.S.A.

[Zhang, Yavatkar, 2000] Lixia Zhang, R. Yavatkar, Fred Baker, Bruce Davie, Peter Ford, Bob Braden, John Wroclawski, M. Speer, Y. Bernet, Eyal Felstaine, *A Framework For Integrated Services Operation Over Diffserv Networks*, 03/13/2000, draft-ietf-issll-diffserv-rsvp-04.txt travail en cours.