Groupe de travail Réseau
Request for Comments : 3670
Catégorie : En cours de normalisation
janvier 2004
Traduction Claude Brière de L'Isle

B. Moore, IBM Corporation
D. Durham, Intel
J. Strassner, Intelliden, Inc.
A. Westerinen, Cisco Systems
W. Weiss, Ellacoya

Modèle d'informations pour décrire les mécanismes de qualité de service des appareils réseau sur le chemin des données

Résumé
Le présent document vise à définir un modèle d'informations pour décrire les mécanismes de qualité de service (QS) inhérents à différents appareils réseau, y compris les hôtes. En gros, ces mécanismes décrivent les propriétés communes pour choisir et conditionner le trafic à travers le chemin de transmission *(datapath)* d'un appareil réseau. Ce choix et ce conditionnement du trafic dans le chemin des données couvrent les deux architectures majeures de la qualité de service : les services différenciés, et les services intégrés.

Le présent document devrait être utilisé avec le modèle d'informations de politique de qualité de service (QPIM) pour modéliser comment les politiques peuvent être définies pour gérer et configurer le mécanisme de QS (c'est-à-dire, la fonction de classification, de marquage, de mesure, d'abandon, de mise en file d'attente, et de programmation) des appareils. Ensemble, ces deux documents décrivent comment écrire les règles de politique de QS pour configurer et gérer les mécanismes de QS présents dans les chemins de données des appareils.

Le présent document, ainsi que QPIM, sont des modèles d'information. C'est-à-dire qu'ils représentent les informations indépendamment d'un lien avec un type de répertoire spécifique.

Table des Matières

1. Introduction

L'objet du présent document est de définir un modèle d'informations pour décrire les mécanismes de qualité de service (QS) inhérents à différents appareils réseau, incluant les hôtes. Dans un sens large, ces mécanismes décrivent les attributs communs pour choisir et conditionner le trafic à travers le chemin de transmission *(datapath)* d'un appareil du réseau. Ce choix et le conditionnement du trafic dans le chemin des données s'étendent sur les deux architectures majeures de QS : les services différenciés (voir la [RFC2475]) et les services intégrés (voir la [RFC1633]).

Le présent document est destiné à être utilisé avec le modèle d'informations de politique de QS [RFC3644] pour modéliser comment des politiques peuvent être définies pour gérer et configurer les mécanismes de QS (c'est-à-dire, la classification, le marquage, le métrage, l'abandon, la mise en file d'attente, et la fonction de programmation) des appareils. Ensemble, ces deux documents décrivent comment écrire les règles de politique de QS pour configurer et gérer les mécanismes de QS présents dans le chemin des données des appareils.

Le présent document, ainsi que la [RFC3644], sont des modèles d'informations. C'est-à-dire qu'ils représentent les informations indépendamment d'un lien à un type spécifique de répertoire. Un document distinct pourrait être rédigé pour donner une transposition des données contenues dans le présent document en une forme convenable pour la mise en œuvre dans un répertoire qui utilise (L)DAP comme protocole d'accès. De même, un document pourrait être écrit pour fournir une transposition des données de la [RFC3644] en un répertoire. Ensemble, ces quatre documents (modèles d'informations et transposition de schéma de répertoire) décriraient alors comment écrire des règles de politique de QS qui puissent être utilisées pour mémoriser les informations dans les répertoires pour configurer les mécanismes de qualité de service d'appareils.

L'approche retenue dans le présent document définit un ensemble commun de classes qui peut être utilisé pour modéliser la QS dans le chemin des données d'un appareil. Les fabricants peuvent alors transposer ces classes, soit directement, soit en utilisant un format intermédiaire comme une PIB COP-PR, pour leurs propres mises en œuvre spécifiques d'appareils. Noter que l'élément de contrôle d'admission des services intégrés n'est pas inclus dans la portée de ce modèle.

La conception des hiérarchies de classe, association, et agrégation décrite dans le présent document est influencée par le sous modèle de qualité de service réseau défini par l'équipe de gestion distribuée (DTMF, *Distributed Management Task Force*) - voir [CIM]. Ces hiérarchies ne sont pas dérivées du modèle d'informations de cœur de politique [RFC3060]. C'est parce que la modélisation des mécanismes de QS d'un appareil est séparée et distincte de la modélisation des politiques qui gèrent ces mécanismes. Donc, il est nécessaire de séparer les mécanismes de qualité de service (le présent document) de leur contrôle (spécifié en utilisant le document de politique générale [RFC3060] augmenté du document de politique de qualité de service [RFC3644]).

Bien qu'il ne soit pas par lui-même un modèle de politique, le présent document ne dépend pas du document Extensions du modèle d'informations de cœur de politique [RFC3460]. Le filtrage de paquet au niveau de l'appareil par lequel un classeur partage un flux de trafic en plusieurs flux, se fonde sur les classes FilterEntryBase et FilterList définies dans la [RFC3460].

Les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" en majuscules dans ce document sont à interpréter comme décrit dans la [RFC2119].

1.1 Modèle conceptuel de gestion de politique

Le modèle d'informations de cœur de politique [RFC3060] décrit une méthodologie générale pour construire des règles de politique. Les extensions à PCIM [RFC3460] mettent à jour et étendent le PCIM d'origine. Une règle de politique agrège un ensemble de conditions de politique et un ensemble ordonné d'actions de politique. La sémantique d'une règle de politique est telle que si l'ensemble des conditions s'évalue à VRAI, l'ensemble des actions est alors exécuté.

Les conditions et actions de politique ont deux composantes principales : des opérandes et des opérateurs. Les opérandes peuvent être des constantes ou des variables. Pour spécifier une politique, il est nécessaire de spécifier :

- o les opérandes à examiner (aussi appelés variables d'état) ;
- o les opérandes à changer (aussi appelés variables de configuration) ;
- o les relations entre ces deux ensembles d'opérandes.

Les opérandes peuvent être spécifiés à haut niveau, comme Joe (un usager) ou Gold (un service). Les opérandes peuvent aussi être spécifiés à un niveau de détail beaucoup plus fin, qui est plus proche du fonctionnement de l'appareil. Des exemples sont une adresse IP ou l'allocation de bande passante d'une file d'attente. Le lien des valeurs légales ou des gammes de valeurs avec un opérande est implicite dans son utilisation. Par exemple, la valeur d'une adresse IP ne peut pas être un entier. Les concepts d'opérandes et de leurs gammes sont définis dans la [RFC3460].

Le second composant des conditions et actions de politique est un ensemble d'opérateurs. Les opérateurs peuvent exprimer à la fois des relations (supérieur à, membre d'un ensemble, OU booléen, etc.) et des allocations. Ensemble, opérateurs et opérandes peuvent exprimer diverses conditions et actions, comme :

Si Bob est un ingénieur...
Si l'adresse IP de source est dans le sous réseau Marketing ...
Régler l'adresse IP de Joe à 192.0.2.100
Limiter la bande passante de l'application x à 10 Mbit/s

On reconnaît que la définition de la sémantique d'opérateur est critique pour la définition des politiques. Cependant, la définition de ces opérateurs sort du domaine d'application du présent document qui va plutôt effectuer (avec la [RFC3644]) les premières étapes de l'identification et de la normalisation d'un ensemble de propriétés (opérandes) à utiliser pour définir les politiques pour les services différenciés et intégrés.

1.2 Objet et relation aux autres travaux de politique

Ce modèle établit un modèle canonique des mécanismes de QS d'un appareil réseau (par exemple, un routeur, un commutateur, ou un hôte) qui est indépendant de tout type spécifique d'appareil réseau. Cela permet de décrire le conditionnement du trafic en utilisant un ensemble commun d'abstractions, modélisé comme un ensemble de services et sous services.

Lorsque les concepts du présent document sont utilisés conjointement avec les concepts de la [RFC3644], on est capable de définir des politiques qui lient les services d'un réseau aux besoins des applications qui utilisent ce réseau. En d'autres termes, les exigences commerciales d'une organisation peuvent être reflétées dans un ensemble de politiques, et ces politiques peuvent être traduites en un ensemble de politiques de niveau inférieur qui contrôle et gère la configuration et le fonctionnement des appareils du réseau.

1.3 Exemples typiques d'usage de politique

Les politiques pourraient être mises en œuvre comme des règles de niveau inférieur en utilisant le modèle d'informations décrit dans la présente spécification. Par exemple, dans une politique de niveau inférieur, une condition pourrait être représentée comme une évaluation d'un attribut spécifique tiré du modèle. Donc, une condition comme "Si filtre = HTTP" serait interprétée comme un essai pour déterminer si des filtres HTTP ont été définis pour l'appareil. Une politique de haut niveau comme "Si protocole = HTTP, marquer alors avec le codet de service différencié 24," serait exprimée par une série d'actions dans une politique de niveau inférieur en utilisant les classes et attributs décrits ci-dessous :

1 Créer un filtre HTTP
2. Créer un marqueur DSCP de valeur 24
3. Lier le filtre HTTP au marqueur DSCP

Noter qu'à la différence de "marquer avec DSCP 24," ces actions de niveau inférieur ne sont pas effectuées sur un paquet lorsque il passe à travers l'appareil. Il y a plutôt des actions de configuration qui sont effectuées sur l'appareil lui-même, pour le rendre prêt à effectuer la ou les actions correctes sur le ou les paquets corrects. L'acte de passer d'une règle de politique de haut niveau à l'ensemble correct d'actions de configuration d'appareil de niveau inférieur est un exemple de ce que la [RFC3198] caractérise comme une "traduction de politique" ou "conversion de politique".

2. Approche

Les activités de QS dans l'IETF se sont principalement concentrées sur deux zones, les services intégrés (IntServ, *Integrated Services*) et les services différenciés (DiffServ, *Differentiated Services*) (voir les [RFC3198], [RFC1633] et [RFC2475]). Le présent document se concentre sur la spécification des propriétés et classes de qualité de service pour modéliser le chemin des données lorsque le trafic de paquets est conditionné. Cependant, le cadre défini par les classes dans le présent document a été conçu aussi en ayant en mémoire les besoins de la portion contrôle d'admission de IntServ.

2.1 Besoins communs de DiffServ et de IntServ

Examinons d'abord IntServ. IntServ a deux composants principaux. Un des composants est incorporé dans le chemin des données de l'appareil de réseautage. Ses fonctions incluent la classification et la régulation des flux individuels, et la programmation des paquets admis sur la liaison sortante. L'autre composant de IntServ est le contrôle d'admission, qui se concentre sur la gestion du protocole de signalisation (par exemple, les messages PATH et RESV de RSVP). Ce composant traite les demandes de réservation, gère la bande passante, externalise les prises de décision aux serveurs de politique, et interagit avec le gestionnaire des tableaux d'acheminement.

On examinera RSVP lors de la définition de la structure de ce modèle d'informations. Comme le présent document se concentre sur le chemin des données, les éléments de RSVP applicables au chemin de données seront examinés dans la structure des classes. Le modèle complet de IntServ sera traité, comme on l'a indiqué précédemment, dans un document ultérieur.

Le présent document modélise un petit sous ensemble du problème de la politique de QS, dans l'espoir de construire une méthodologie qui puisse être adaptée pour d'autres aspects de la QS en particulier, et de la construction de politique en général. L'accent dans le présent document est mis sur la QS pour les appareils qui mettent en œuvre le conditionnement du trafic dans le chemin des données.

DiffServ fonctionne exclusivement dans le chemin des données. Il a tous les mêmes composants du chemin des données IntServ, avec deux différences majeures. D'abord, DiffServ classe les paquets sur la seule base de leur champ DSCP, tandis que IntServ examine un sous ensemble du quintuplet d'adressage d'un flux standard. L'exception à cette règle se produit dans un routeur ou hôte à la frontière d'un domaine DiffServ. Un appareil dans cette position peut examiner le DSCP d'un paquet, son quintuplet d'adressage, d'autres champs du paquet, ou même des informations complètement extérieures au paquet, pour déterminer la valeur du DSCP avec laquelle marquer le paquet avant son transfert dans le domaine DiffServ. Cependant, les routeurs dans l'intérieur d'un domaine DiffServ vont seulement avoir besoin de classer sur la base du champ DSCP.

La seconde différence entre IntServ et DiffServ est que le protocole de signalisation utilisé dans IntServ (par exemple, RSVP) affecte la configuration du chemin des données d'une façon plus dynamique. C'est pourquoi chaque réservation RSVP nouvellement admise exige une reconfiguration du chemin des données. À l'opposé, DiffServ exige beaucoup moins de changements au chemin des données après que les comportements par bond (PHB, *Per Hop Behavior*) ont été configurés.

L'approche défendue dans le présent document pour la création de politiques qui contrôlent les divers mécanismes de QS des appareils de réseautage est d'abord d'identifier les attributs avec lesquels les politiques doivent être construites. Ces attributs sont les paramètres utilisés dans les expressions qui sont nécessaires pour construire les politiques. Il y a aussi un désir parallèle de définir les opérateurs, les relations, et les constructions de préséance nécessaires pour élaborer les conditions et actions qui constituent ces politiques. Cependant, ces efforts sortent du domaine d'application du présent document.

2.2 Besoins spécifiques de DiffServ

Les règles spécifiques de DiffServ se concentrent sur deux domaines particuliers : le cœur et les bords du réseau. Comme expliqué dans le document d'architecture DiffServ [RFC2475], les appareils à la bordure du réseau classent le trafic en différents flux de trafic. Le cœur du réseau transmet alors le trafic provenant des différents flux en utilisant un ensemble de comportements par bond (PHB). Un DSCP identifie chaque PHB. Le DSCP fait partie de l'en-tête IP de chaque paquet (comme décrit dans la [RFC2474]). Cela permet que de multiples flux de trafic soient agrégés dans un petit nombre de flux de trafic agrégés, où chaque flux de trafic agrégé est identifié par un DSCP particulier, et transmis en utilisant un PHB particulier.

Les attributs utilisés pour manipuler les capacités de QS dans le cœur du réseau visent principalement les caractéristiques de comportement de chaque PHB pris en charge. Aux bordures du réseau DiffServ, la complexité supplémentaire de la classification des flux, de la régulation, des transpositions RSVP, des nouveaux marquages, et d'autres facteurs doivent être pris en compte. Une modélisation supplémentaire sera nécessaire dans ce domaine. Cependant, d'abord, les normes pour les bordures du réseau DiffServ doivent être détaillées, pour permettre que les bordures soient incorporées dans le modèle de politique.

2.3 Besoins spécifiques de IntServ

Le présent document se concentre exclusivement sur les aspects de transmission de la qualité de service du réseau. Donc, bien que les aspects de transmission de IntServ soient considérés, la gestion de IntServ n'est pas prise en compte. Ce sujet sera traité dans un document à venir.

3. Méthodologie

Il y a un clair besoin de définition des attributs et comportements qui ensemble déterminent comment le trafic devrait être conditionné. Le présent document définit un ensemble de classes et relations qui représentent les mécanismes de QS utilisés pour conditionner le trafic ; la [RFC3644] est utilisée pour définir les politiques de contrôle des mécanismes de QS définis dans le présent document.

Cependant, certaines questions très basiques doivent être considérées lorsque on combine ces documents. L'examen de ces questions devrait aider à construire un schéma de gestion du fonctionnement et de la configuration des mécanismes de QS du réseau à travers l'utilisation des politiques de QS.

3.1 Niveau d'abstraction pour exprimer les politiques de QS

La première question à examiner est celle du niveau d'abstraction auquel les politiques de QS devraient être exprimées. Si on considère les politiques comme un ensemble de règles utilisées pour réagir aux événements et manipuler des attributs ou générer de nouveaux événements, on réalise que la politique représente un continuum de spécifications qui se rapportent aux objectifs et règles d'affaires du conditionnement de trafic fait par un appareil ou un ensemble d'appareils. Un exemple de politique de niveau affaires pourrait être "de 13:00 PST à 7:00 EST, revendre 40 % de la capacité réseau sur le marché libre". À l'opposé, une politique spécifique d'un appareil pourrait être "Si la longueur de file d'attente croît de façon géométrique sur une durée spécifiée, déclencher un événement de défaillance potentielle de la liaison".

Un modèle général pour ce continuum est montré à la Figure 1.

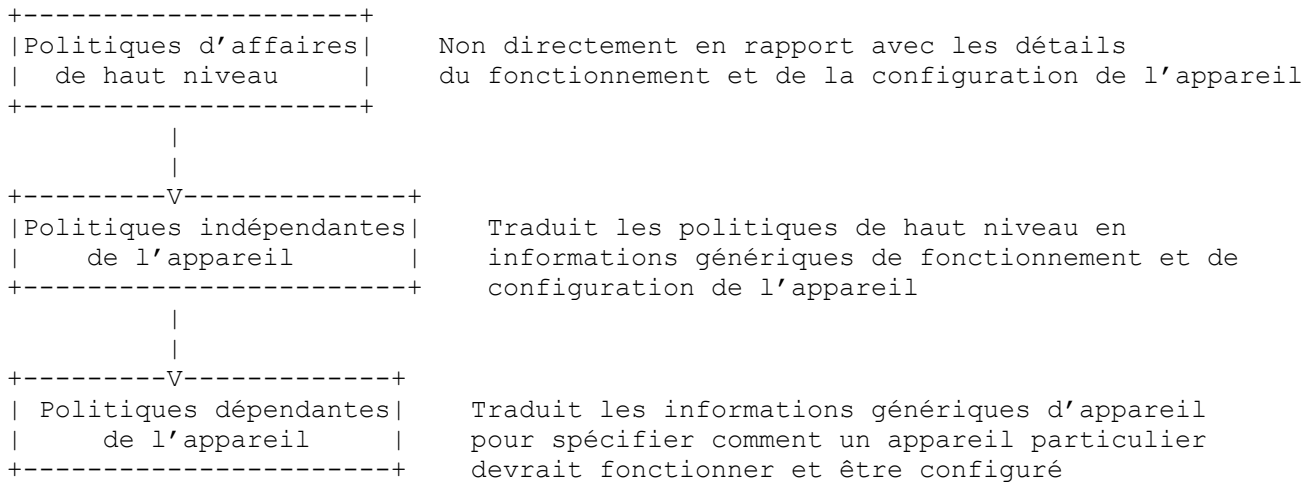


Figure 1. L'espace de la politique

Les politiques d'affaires de haut niveau sont utilisées pour exprimer les exigences des différentes applications, et donnent la priorité aux applications qui reçoivent un "meilleur" traitement lorsque le réseau est encombré. Le but est alors d'utiliser les politiques pour mettre en rapport directement les besoins de fonctionnement et de configuration d'un appareil avec les règles d'affaires que l'administrateur du réseau essaye de mettre en œuvre dans le réseau auquel appartient l'appareil.

Les politiques indépendantes de l'appareil traduisent les politiques d'affaires en un ensemble généralisé de politiques de fonctionnement et de configuration qui est indépendant de tout appareil spécifique, mais dépend d'un ensemble particulier de mécanismes de QS, comme la détection précoce aléatoire (RED, *random early detection*) d'abandon ou la programmation round robin pondérée. Non seulement cela permet que différents types d'appareils (routeurs, commutateurs, hôtes, etc.) soient contrôlés par les politiques de QS, mais cela permet aussi que des appareils de fabricants différents soient contrôlés en utilisant

les mêmes types de mécanismes de QS. Cela permet que ces différents appareils fournissent chacun le conditionnement relatif correct au même type de trafic.

À l'opposé, les politiques dépendantes de l'appareil traduisent les politiques indépendantes de l'appareil en politiques spécifiques d'un certain appareil. La raison d'effectuer une distinction entre politique indépendante et politique dépendante de l'appareil est que dans un certain réseau, de nombreux appareils différents qui ont des capacités différentes doivent être contrôlés ensemble. Les politiques indépendants de l'appareil donnent une couche commune d'abstraction pour gérer plusieurs appareils de capacités différentes, tandis que les politiques dépendantes de l'appareil mettent en œuvre le conditionnement spécifique requis. Le présent document donne un ensemble commun d'abstractions pour représenter les mécanismes de QS d'une façon indépendante de l'appareil.

Le présent document se concentre sur la représentation de mécanismes de QS indépendants de l'appareil. Les mécanismes de QS sont modélisés dans un détail suffisant pour fournir une représentation commune indépendante de l'appareil des politiques de QS. Ils peuvent aussi être utilisés pour fournir une base pour la spécialisation, permettant à chaque fabricant de déduire un ensemble de classes spécifique du fabricant qui représentent comment le conditionnement du trafic est fait pour cet ensemble d'appareils du fabricant.

3.2 Spécification des paramètres de politique

Les politiques sont une fonction de paramètres (attributs) et d'opérateurs (booléens, arithmétiques, relationnels, etc.). Donc, tous deux doivent être définis au titre de la même politique afin de conditionner correctement le trafic. Si les paramètres de la politique sont spécifiés trop étroitement, ils vont refléter les mises en œuvre individuelles de QS dans chaque appareil. Comme il y a actuellement peu de consensus dans l'industrie sur le modèle correct de mise en œuvre de la QS, la plupart des attributs définis ne vont être applicables qu'aux caractéristiques de quelques appareils individuels. De plus, normaliser toutes les mises en œuvre potentielles serait une tâche sans fin car de nouvelles mises en œuvre continuent d'apparaître sur le marché.

D'un autre côté, si les paramètres de la politique sont spécifiés de façon trop large, il est impossible de développer des politiques significatives. Par exemple, si on se concentre sur ce qu'on appelle l'ensemble Olympique de politiques, une politique d'affaires comme "Bob obtient le service Or" n'a clairement pas de sens pour la grande majorité des appareils existants. Cela parce que l'appareil n'a aucun moyen de déterminer qui est Bob, ou quels mécanismes de QS devraient être configurés de façon à fournir le service Or.

De plus, le service Or peut représenter un seul service, ou il peut identifier un ensemble de services qui sont en relation les uns avec les autres. Dans ce dernier cas, ces services peuvent avoir des caractéristiques de conditionnement différentes.

Le présent document définit un ensemble de paramètres qui entrent dans un modèle canonique pour modéliser les éléments du chemin de transmission d'un appareil qui met en œuvre le conditionnement de trafic à qualité de service. En définissant ce modèle d'une façon indépendante de l'appareil, les paramètres nécessaires peuvent être abstraits de façon appropriée.

3.3 Spécification des services de politique

Les administrateurs veulent la souplesse d'une capacité de définition du conditionnement du trafic sans avoir une compréhension précise des différents mécanismes de QS qui mettent en œuvre ce conditionnement. De plus, les administrateurs veulent pouvoir grouper différents services, pour décrire un concept de niveau supérieur tel qu'un "Service Or". Ce service de niveau supérieur pourrait être vu comme fournissant le traitement pour fournir la qualité de service "Or".

Ces deux objectifs dictent le besoin de l'ensemble d'abstractions suivant :
- une façon souple pour décrire un service
- doit être capable de grouper différents services qui peuvent utiliser ensemble différentes technologies (par exemple, DiffServ et IEEE 802.1Q)
- doit être capable de définir un ensemble de sous services qui ensemble constituent un service de niveau supérieur
- doit être capable d'associer un service et l'ensemble de mécanismes de QS utilisés pour conditionner le trafic pour ce service
- doit être capable de définir des politiques qui gèrent les mécanismes de QS utilisés pour mettre en œuvre un service.

Le présent document traite cet ensemble de problèmes en définissant un ensemble de classes et associations qui peuvent représenter des concepts abstraits comme un "service Or" et lier chacun de ces services abstraits à un ensemble spécifique de mécanismes de QS qui mettent en œuvre le conditionnement qu'ils exigent. De plus, le présent document définit le concept de "sous services" pour permettre de définir le Service Or soit comme un seul service, soit comme un ensemble de services qui ensemble devraient être traités comme une entité atomique.

Ces abstractions étant posées, les politiques (comme définies dans la [RFC3644]) peuvent être écrites pour contrôler les mécanismes et services de QS définis dans le présent document.

3.4 Niveau d'abstraction pour définir les attributs et classes de QS

Le présent document définit un ensemble de classes et propriétés pour prendre en charge les politiques qui configurent les mécanismes de QS des appareils. Le présent document se concentre sur la représentation des services dans le chemin des données qui prennent en charge aussi bien DiffServ (pour le conditionnement de trafic agrégé) et IntServ (pour le conditionnement de trafic fondé sur le flux). Les classes et propriétés pour modéliser les services de contrôle d'admission IntServ pourront être définis dans de futurs documents.

Les classes et propriétés dans le présent document sont conçues pour être utilisées en conjonction avec les classes et propriétés de politique de QS définies dans la [RFC3644]. Par exemple, pour préserver les caractéristiques de délai affectées à un utilisateur final, un administrateur de réseau peut souhaiter créer des politiques qui surveillent les longueurs de files d'attente dans un appareil, et ajuster les allocations de ressources lorsque les budgets de délais sont tendus (peut-être par suite d'un changement de topologie du réseau). La classes et propriétés du présent document définissent les services et mécanismes spécifiques exigés pour mettre en œuvre ces services. Les classes et propriétés définies dans la [RFC3644] fournissent la structure globale de la politique qui gère et configure ce service.

Cette combinaison de spécification de bas niveau (utilisant le présent document) et de structuration de haut niveau (qui utilise la [RFC3644]) de services réseau permet aux administrateurs de réseau de définir les nouveaux services requis par le réseau, qui se rapportent directement aux objectifs d'affaires, tout en assurant que de tels services peuvent être gérés. Cependant, cet objectif (de créer et gérer des politiques orientées service) ne peut être réalisé que si des politiques peuvent être construites qui sont capables de prendre en charge diverses mises en œuvre de QS. La solution est de modéliser les capacités de QS des appareils au niveau du comportement. Cela signifie pour les services de conditionnement du trafic réalisés dans le chemin des données que le modèle doit prendre en charge les caractéristiques suivantes :

- o modéliser un service réseau générique avec des capacités de QS
- o modéliser comment le conditionnement de trafic est lui-même défini
- o modéliser comment les statistiques sont rassemblées pour surveiller les service de conditionnement du trafic à QS - cette facette du modèle sera ajoutée dans un document futur.

Le présent document modélise un service réseau, et l'associe à un ou plusieurs mécanismes de QS qui sont utilisés pour mettre en œuvre ce service. Il modélise aussi sous une forme canonique les divers composants qui sont utilisés pour conditionner le trafic, de sorte que des services de conditionnement de trafic standard aussi bien que personnalisés puissent être décrits.

3.5 Caractérisation des propriétés de QS

Les propriétés et classes de QS seront décrites plus en détail à la Section 4. Cependant, on va examiner les caractéristiques de base de ces propriétés, pour comprendre la méthodologie de leur représentation.

Il y a essentiellement deux types de propriétés, d'état et de configuration. Les propriétés de configuration décrivent l'état désiré d'un appareil, et incluent les propriétés et classes pour représenter les seuils désirés ou proposés, les allocations de bande passante, et comment classer le trafic. Les propriétés d'état décrivent l'état réel de l'appareil. Cela inclut des propriétés pour représenter les valeurs opérationnelles courantes des attributs dans les appareils configurés via les propriétés de configuration, ainsi que les propriétés qui représentent l'état (longueur de file d'attente, consommation excédentaire de capacité, taux de pertes, et ainsi de suite).

Pour être corrélés et utilisés ensemble, ces deux types de propriétés doivent être modélisés en utilisant un modèle d'informations commun. La possibilité de modéliser les propriétés d'état et leurs réglages de configuration correspondants est réalisée en utilisant les mêmes classes dans ce modèle – bien que des instances individuelles des classes devraient être appelées avec les noms appropriés ou être placées dans des conteneurs différents pour distinguer les valeurs d'état courant des réglages de configuration désirés.

Les informations d'état sont traitées de façon très limitée par QDDIM. Actuellement, seul CurrentQueueDepth est proposé comme attribut sur QueuingService. La majorité du modèle se rapporte à la configuration. Ceci étant dit, on suppose que ce modèle est une transposition directe de mémoire dans un appareil. Toutes les manipulations de classes et propriétés de modèle affectent directement l'état de l'appareil. Chaque mise en œuvre est libre, si elle le souhaite, d'utiliser aussi ces classes pour représenter la configuration désirée.

On reconnaît que ces propriétés supplémentaires sont nécessaires pour modéliser complètement l'état actuel. Cependant, beaucoup des propriétés définies dans le présent document représentent exactement les variables d'état qui seront configurées par les propriétés de configuration. Donc, la définition des propriétés de configuration a une correspondance exacte avec les propriétés d'état, et peut être utilisée pour modéliser la configuration aussi bien actuelle (état) que désirée/proposée.

3.6 Déduction du modèle d'informations de QS

La question du contexte conduit aussi à une autre question : comment les informations spécifiées dans les modèles de cœur et de politique de QS (respectivement les [RFC3060], [RFC3460], et [RFC3644]) s'intègrent avec les informations définies dans le présent document ? Pour le dire autrement, d'où devraient être déduits les concepts indépendants de l'appareil pour conduire aux attributs de QS spécifiques de l'appareil ?

On pensait autrefois que la QS faisait partie du modèle de politique. Cette façon de voir n'est pas complètement pertinente, et conduit à une certaine confusion. La QS est un ensemble de services qui peuvent être contrôlés en utilisant la politique. Ces services sont représentés comme des mécanismes d'appareils. Un point important ici est que les services de QS, tout comme les autres types de services (par exemple, la sécurité) sont fournis par des mécanismes inhérents à un certain appareil. Cela signifie que tous les appareils sont bien sûr égaux à la création. Par exemple, bien que deux appareils puissent avoir le même type de mécanisme (par exemple, une file d'attente) l'un peut être une simple mise en œuvre (c'est-à-dire, une file d'attente FIFO) tandis qu'une autre peut être beaucoup plus complexe et robuste (par exemple, une mise en file d'attente équitable à pondération fondée sur la classe (CBWFQ, *class-based weighted fair queuing*)). Cependant, ces deux appareils peuvent être utilisés pour fournir des services de QS, et tous deux doivent être contrôlés par une politique. Donc, une politique indépendantes de l'appareil peut ordonner à des appareils de mettre en file d'attente certain trafic, et une politique spécifique de l'appareil peut être utilisée pour contrôler la mise en file d'attente dans chaque appareil.

De plus, la politique est utilisée pour contrôler ces mécanismes, non pour les représenter. Par exemple, les services de QS sont mis en œuvre avec des classeurs, des mesureurs, des marqueurs, des élimineurs, des files d'attentes, et des programmateurs. De même, la sécurité est aussi une caractéristique des appareils, comme les capacités d'authentification et de chiffrement représentent des services qu'effectuent les appareils en réseau sans considération des interactions avec les serveurs de politique). Ces services de sécurité peuvent utiliser certains des mécanismes utilisés par les services de QS, comme les concepts de filtres. Cependant, ils vont pour la plupart exiger des mécanismes différents de ceux utilisés par la QS, même si les deux ensembles de services sont mis en œuvre dans les mêmes appareils.

Donc, la similarité entre le modèle de QS et les modèles des autres services n'est pas tant qu'ils contiennent quelques mécanismes communs mais plutôt qu'ils modélisent comment un appareil met en œuvre ses propres services.

À ce titre, la modélisation de la QS devrait faire partie d'un schéma d'appareil de réseautage plutôt que d'un schéma de politique. Cela permet au schéma d'appareils de réseautage de se concentrer sur les mécanismes de modélisation d'appareil, et au schéma de politique de le faire sur la sémantique de la représentation de la politique elle-même (conditions, actions, opérateurs, etc.). Bien que le présent document se concentre sur la définition d'un modèle d'informations pour représenter les services de QS dans le chemin des données d'un appareil, le but ultime est d'être capable d'appliquer des politiques qui contrôlent ces services dans les appareils du réseau. De plus, ces deux schémas (appareil et politique) doivent être étroitement intégrés afin de permettre que la politique contrôle les services de QS.

3.7 Représentation d'attribut

La dernière question à considérer est celle de la façon de représenter les attributs. Si les attributs de QS sont représentés comme des nombres absolus (par exemple, la classe AF2 obtient 2 Mbit/s de bande passante) il est plus difficile de les rendre uniformes à travers plusieurs accès dans un appareil ou sur plusieurs appareils, à cause des grandes variation des capacités des liaisons. Cependant, exprimer les attributs en termes relatifs ou proportionnels (par exemple, la classe AF2 obtient 5 % de la bande passante totale de la liaison) rend plus difficile d'exprimer certains types de conditions et actions, comme :

(Si BandePassanteConsommée = BandePassanteAllouée Alors ...)

Il y a en fait trois approches pour traiter ce problème :

- Plusieurs propriétés peuvent être définies pour exprimer la même valeur de diverses formes. Cette idée a été rejetée à cause de la difficulté de garder synchronisées ces différentes propriétés (par exemple, lorsque une propriété change, toutes les autres doivent être mises à jour).

- Des propriétés multimodales peuvent être définies pour exprimer la même valeur, en termes différents, sur la base du mode d'accès ou d'allocation. Cette option a été rejetée parce qu'elle complique significativement le modèle et est impossible à exprimer dans les protocoles d'accès de répertoire actuels (par exemple, (L)DAP).

- Les propriétés peuvent être exprimées comme "absolues", mais les opérateurs dans le schéma de politique vont devoir être plus sophistiqués. Donc, pour représenter un pourcentage, des opérateurs de division et multiplication sont nécessaires (par exemple, la classe AF2 obtient 0,05 * de la bande passante totale de la liaison). C'est l'approche qui a été retenue dans le présent document.

3.8 Modèle mental

Le modèle mental pour construire ce schéma se fonde sur les travaux du groupe de travail Services différenciés. Ce schéma se fonde sur les informations fournies dans les versions actuelles du modèle de gestion DiffServ informel [RFC3290], la MIB DiffServ [RFC3289], la PIB [RFC3317], ainsi que sur les informations de l'ensemble de RFC qui constitue la définition de base de DiffServ lui-même ([RFC2474], [RFC2475], [RFC2597], et [RFC3246]). De plus, un ensemble commun de terminologie est disponible dans la [RFC3198].

Ce modèle est construit autour de deux hiérarchies de classes fondamentales qui sont liées ensemble en utilisant un jeu d'associations. Les deux hiérarchies de classes découlent des classes de base de QoSService et ConditioningService. Un ensemble d'associations rapporte les sous classes de niveau inférieur de QoSService aux services de QS de niveau supérieur, met en relation les différents types de services de conditionnement en traitant une classe de trafic, et met en rapport un ensemble de services de conditionnement avec un service de QS spécifique. Cette combinaison d'associations nous permet de voir l'appareil comme fournissant un ensemble de services qui peuvent être configurés, dans un ensemble modulaire de blocs de construction, pour bâtir des services spécifiques de l'application. Donc, le présent document peut être utilisé pour modéliser les normes existantes et futures ainsi que les services de QS réseau spécifiques d'application.

3.8.1 Classe QoSService

La première des classes définies ici, QoSService, est utilisée pour représenter des services réseau de niveau supérieur qui exigent un conditionnement spécial de leur trafic. Une instance de QoSService (ou d'une de ses sous classes) est utilisée pour mettre ensemble un groupe de services de conditionnement qui, du point de vue du gestionnaire de système, sont tous utilisés pour livrer un service commun. Donc, l'ensemble de classeurs, marqueurs, et services de conditionnement en rapport qui fournissent le service premium à l'ensemble "sélectionné" de trafics utilisateurs peut être regroupé dans un service de QS premium.

QoSService a un ensemble de sous classes qui représentent différentes approches de livraison de services IP. L'ensemble actuellement défini de sous classes est un FlowService pour la livraison de QS en mode flux et un DiffServService pour la livraison de service de QS DiffServ en mode agrégé.

Les services de QS peuvent être mis en relation les uns avec les autres comme homologues, ou ils peuvent être mis en œuvre comme des services subordonnés les uns aux autres. L'agrégation QoSSubService indique qu'un ou plusieurs objets QoSService sont subordonnés à un objet QoSService particulier. Par exemple, cela nous permet de définir le service Or comme une combinaison de deux services DiffServ, un pour le traitement du trafic de haute qualité, et un pour servir le reste du trafic. Chacun de ces objets DiffServService sera associé à un ensemble de classeurs, marqueurs, etc., de sorte que le trafic de haute qualité obtiendra le marquage EF et la mise en file d'attente appropriée.

La classe DiffServService elle-même a une sous classe AFService. Cette sous classe est utilisée pour représenter la notion spécifique que plusieurs marquages appropriés au sein du groupe de PHB AF travaillent ensemble pour fournir un seul service. Lorsque d'autres groupes de PHB DiffServ qui utilisent plus d'un codet seront définis , ils seront probablement des candidats pour des sous classes DiffServService supplémentaires.

Les transpositions dans une technologie spécifique de ces services, représentant l'utilisation spécifique d'un marquage de PHB ou d'un marquage 802.1Q, sont capturées au sen de la hiérarchie ConditioningService, plutôt que dans les sous classes de QoSService.

Ces concepts sont décrits dans la Figure 2. Noter que les deux associations sont des agrégations : un objet QoSService agrège à la fois l'ensemble d'objets QoSService qui lui sont subordonnés, et l'ensemble d'objets ConditioningService qui le réalisent. Voir à la Section 4 les définitions de classe et d'association.

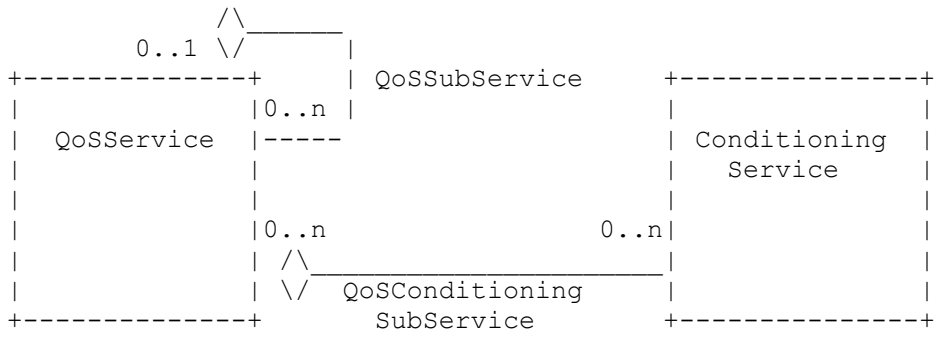


Figure 2 : QoSService et ses agrégations

3.8.2 Classe ConditioningService

L'objectif des classes ConditioningService est de décrire la séquence des conditionnements de trafic qui est appliquée à un certain flux de trafic sur l'interface d'entrée à travers laquelle il pénètre dans un appareil, et ensuite sur l'interface de sortie par laquelle il quitte l'appareil. Cela se fait en utilisant un ensemble de classes et relations. La décision d'acheminement dans le cœur de l'appareil, qui choisit quelle interface de sortie va utiliser un paquet particulier n'est pas représentée dans ce modèle.

Une seule classe de base, ConditioningService, est la super classe pour un ensemble de sous classes représentant les mécanismes qui conditionnent le trafic.

Ces sous classes définissent des primitives de conditionnement indépendantes de l'appareil (incluant des classeurs, mesureurs, marqueurs, éliminateurs, files d'attentes) qui ensemble mettent en œuvre le conditionnement du trafic sur une interface. Ce modèle abstrait ces services en un ensemble commun de blocs de construction modulaires qui peuvent être utilisés, sans considération de la mise en œuvre de l'appareil, pour modéliser le conditionnement de trafic interne à un appareil.

Les différents mécanismes de conditionnement doivent être mis en rapport les uns avec les autres pour décrire comment le trafic est conditionné. Il existe plusieurs variations importantes de la façon dont ces services sont mis en rapport les uns avec les autres :

- une interface d'entrée ou de sortie particulière peut ne pas exiger tous les types de ConditioningServices,
- plusieurs instances du même mécanisme peuvent être nécessaires sur une interface d'entrée ou de sortie,
- il n'y a pas d'ordre établi des applications pour les ConditioningServices sur une interface d'entrée ou de sortie.

Donc, ce modèle n'impose pas un ordre fixe entre les sous classes de ConditioningService, ni n'identifie une sous classe de ConditioningService qui doive apparaître en premier ou en dernier parmi les ConditioningServices sur une interface d'entrée ou de sortie. Ce modèle lie plutôt ensemble les diverses instances de ConditioningService sur une interface d'entrée ou de sortie en utilisant les associations NextService, NextServiceAfterMeter, et NextServiceAfterConditioningElement. Il y a aussi des associations séparées, appelées IngressConditioningServiceOnEndpoint et EgressConditioningServiceOnEndpoint, qui, respectivement, lient une interface d'entrée à son premier ConditioningService, et lient une interface de sortie à son ou ses derniers ConditioningServices.

3.8.3 Préservation des informations de QS de l'entrée à la sortie

Il y a une divergence importante entre le modèle QDDIM et la [RFC3290]. Dans la [RFC3290], le trafic passe à travers un appareil réseau en trois étapes :

- Il entre par une interface d'entrée, où il peut recevoir le conditionnement de QS.
- Il traverse le cœur d'acheminement, où une logique sortant du domaine d'application de la QS détermine quelle interface de sortie il va utiliser pour quitter l'appareil.
- Il peut recevoir plus de conditionnement de QS sur l'interface de sortie choisie, et il quitte ensuite l'appareil.

Dans ce modèle, aucune information sur le conditionnement de QS que reçoit un paquet sur l'interface d'entrée n'est communiquée avec le paquet à travers le cœur d'acheminement à l'interface de sortie.

Le modèle QDDIM relâche cette restriction, pour permettre de communiquer des informations sur le traitement qu'a reçu un paquet sur une interface d'entrée avec le paquet à l'interface de sortie. (Ce relâchement ajoute une capacité qui est présente dans de nombreux appareils réseau.) QDDIM représente ce transfert d'informations en termes de préambule de paquet, qui dit combien d'appareils le mettent en œuvre. Mais les mises en œuvre sont libres d'utiliser d'autres mécanismes pour arriver au même résultat.

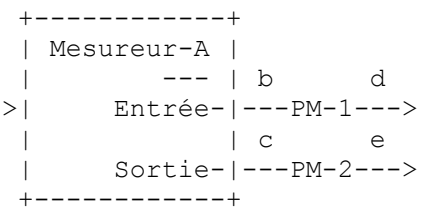


Figure 3 : Mesureur suivi de deux marqueurs de préambule

La Figure 3 montre un exemple dans lequel les résultats du mesureur sont capturés dans un préambule de paquet. Les flèches marquées avec une seule lettre représentent les instances de l'association NextService (a, d, et e), ou de son association homologue NextServiceAfterMeter (b et c). Le PreambleMarker PM-1 ajoute au préambule du paquet l'indication que le paquet est sorti du mesureur A comme trafic conforme. De façon similaire, le PreambleMarker PM-2 ajoute aux préambules des paquets qui le traversent l'indication qu'ils sont sortis du mesureur A comme trafic non conforme. Un PreambleMarker ajoute ces informations à ce qui est déjà présent dans un préambule de paquet, par opposition à une réécriture de ce qui y est.

Pour nourrir l'interopérabilité, le format de base des informations capturées par un PreambleMarker est spécifié. (Les mises en œuvre sont bien sûr libres de représenter en interne ces informations d'une façon différente – ceci est juste la façon dont elles sont représentées dans le modèle.) Les informations sont représentées par une propriété de chaîne ordonnée multi valeurs FilterItemList, où chaque valeur individuelle de la propriété est de la forme "<type>,<valeur>". Lorsque un PreambleMarker "ajoute" ses informations à celles qui étaient déjà présentes dans un préambule de paquet, il le fait en ajoutant des éléments supplémentaires du format indiqué à la fin de la liste.

QDDIM donne un ensemble limité de <types> qu'un PreambleMarker peut utiliser :
- ConformingFromMeter : la valeur est le nom du mesureur.
- PartConformingFromMeter : la valeur est le nom du mesureur.
- NonConformingFromMeter : la valeur est le nom du mesureur.
- VlanId : la valeur est l'identifiant de LAN virtuel (VLAN ID).

Les mises en œuvre peuvent reconnaître d'autres <types> en plus de ceux-là. Si des collisions de <type> spécifiques de la mise en œuvre devenaient un problème, il serait possible que les <types> deviennent une gamme administrée par l'IANA dans une future révision du présent document.

Pour faire usage des informations qu'un PreambleMarker mémorise dans un préambule de paquet, une sous classe PreambleFilter spécifique de FilterEntryBase est définie, pour correspondre aux chaînes "<type>,<valeur>". Pour simplifier le cas où il y a juste un seul niveau de mesure dans un appareil, mais des mesureurs individuels différents sur chaque interface d'entrée, PreambleFilter permet un caractère générique de "any" pour la partie <valeur> des trois filtres en rapport avec la mesure. Avec ce caractère générique, un administrateur peut spécifier un classeur pour choisir tous les paquets qui se sont trouvés conformes (ou partiellement conformes, ou non conformes) par leurs mesures respectives, sans avoir à nommer chaque mesureur individuellement dans un ClassifierElement séparé.

Une fois qu'un résultat de mesure a été mémorisé dans un préambule de paquet, il est disponible pour être utilisé par tout classeur suivant. Donc, bien que la motivation de cette capacité ait été décrite en termes de préservation des informations de conditionnement de QS provenant d'une interface d'entrée pour une interface de sortie, un résultat de mesure antérieur peut aussi être utilisé pour classer les paquets ultérieurement dans le chemin des données sur la même interface où réside de mesureur.

3.9 Classeurs, listes de filtres, et entrées de filtres

Le présent document utilise un certain nombre de classes pour modéliser les classeurs définis dans la [RFC3290] : ClassifierService, ClassifierElement, FilterList, FilterEntryBase, et diverses sous classes de FilterEntryBase. Deux associations sont aussi impliquées : ClassifierElementUsesFilterList et EntriesInFilterList. Le modèle QDDIM n'utilise pas de classe FilterEntry de CIM.

Dans la [RFC3290], un seul flux de trafic venant d'un classeur est partagé en plusieurs flux de trafic qui le quittent, sur la base desquels un ensemble ordonné de filtres correspond chaque paquet du flux entrant. Un filtre correspond soit à un champ dans le paquet lui-même, soit éventuellement à d'autres attributs associés au paquet. Dans le cas de classeur multi champs (MF, *multi-field)*) les paquets sont affectés aux flux sortants sur la base du contenu de plusieurs champs dans l'en-tête de paquet. Par exemple, un classeur MF peut affecter des paquets à un flux sortant sur la base de son quintuplet d'adressage IP complet.

Pour optimiser la représentation des classeurs MF, on introduit des sous classes de FilterEntryBase qui permettent de représenter plusieurs champs d'en-tête de paquet en rapport dans un seul objet. Ces sous classes sont IPHeaderFilter et 8021Filter. Avec IPHeaderFilter, par exemple, les critères de choix des paquets sur la base de tous les cinq champs d'en-tête du quintuplet IP et le DSCP DiffServ peuvent être représentés par une FilterList contenant un objet IPHeaderFilter. Parce que ces deux classes ont des applications qui vont au delà de celles considérées dans le présent document, elles sont définies, tout comme la classe abstraite FilterEntryBase, dans le document plus général [RFC3460] plutôt qu'ici.

L'objet FilterList est toujours nécessaire, même si il ne contient qu'une seule entrée d'objet de filtre (c'est-à-dire, une sous classe FilterEntryBase). C'est parce que un ClassifierElement peut seulement être associé à une FilterList, par opposition à une FilterEntry individuelle. FilterList est aussi défini dans la [RFC3460].

L'agrégation EntriesInFilterList (aussi définie dans la [RFC3460]) a une propriété EntrySequence, qui dans le passé (dans CIM) pouvait être utilisée pour spécifier un ordre d'évaluation sur les entrées de filtre dans une FilterList. Maintenant cependant, la propriété EntrySequence ne prend en charge qu'une seule valeur : '0'. Cette valeur indique que les FilterEntries sont ajoutées ensemble par l'opérateur logique ET pour déterminer si un paquet correspond au sélecteur MF que représente la FilterList.

Un ClassifierElement spécifie le point de départ d'une politique spécifique ou d'un chemin de données. Chaque ClassifierElement utilise l'association NextServiceAfterClassifierElement pour déterminer le prochain service de conditionnement à appliquer aux paquets.

Un ClassifierService définit un groupement de ClassifierElements. Il y a certaines instances où un ClassifierService spécifie en fait une agrégation de ClassifierServices. Un cas pratique serait celui où chaque ClassifierService spécifie un groupe de politiques associé à une application particulière et un autre ClassifierService groupe les instances de ClassifierService spécifiques d'une application. Dans ce cas particulier, les instances spécifiques d'application de ClassifierService sont spécifiées une seule fois, mais les combinaisons uniques de ces ClassifierServices sont spécifiées, en tant que de besoin, en utilisant d'autres instances de ClassifierService. Les instances de ClassifierService qui groupent d'autres instances de ClassifierService ne sont pas autorisées à spécifier une FilterList en utilisant l'association ClassifierElementUsesFilterList. Cette utilisation spéciale de ClassifierService ne sert que de fonction de collecte de classeurs.

3.10 Modélisation des abandonneurs

Dans la [RFC3290], une distinction est faite entre les abandonneurs absolus et les abandonneurs algorithmiques. Dans QDDIM, les deux types d'abandonneurs sont modélisés avec la classe DropperService, ou avec une de ses sous classes. Dans les deux cas, la file d'attente à partir de laquelle l'abandonneur élimine les paquets est liée à l'abandonneur par une instance de l'association NextService. L'abandonneur jour toujours le rôle PrecedingService dans ces associations, et la file d'attente joue toujours le rôle FollowingService. Il y a toujours exactement une file d'attente à partir de laquelle un abandonneur élimine les paquets.

Comme un abandonneur absolu élimine tous les paquets de sa file d'attente, il n'a besoin d'aucune configuration au delà d'un lien NextService à cette file d'attente. Pour un abandonneur algorithmique, plus de configuration est cependant nécessaire :
- o un algorithme d'abandon spécifique ;
- o des paramètres pour l'algorithme (par exemple, la taille du baquet de jetons) ;
- o la ou les sources d'entrée à l'algorithme ;
- o les éventuels paramètres par entrée pour l'algorithme.

Les deux premiers de ces éléments sont représentés par des propriétés de la classe DropperService, ou des propriétés d'une de ses sous classes. Les deux derniers impliquent cependant des classes et associations supplémentaires.

3.10.1 Configuration des abandonneurs de tête et de queue

HeadTailDropQueueBinding est l'association qui identifie les entrées pour l'algorithme exécuté par un abandonneur de queue. Cette association n'est pas utilisée pour un abandonneur de tête, parce qu'un abandonneur de tête a toujours exactement une entrée à son algorithme d'abandon, et cette entrée est toujours la file d'attente de laquelle il élimine les paquets. Pour un abandonneur de queue, cette association est définie comme ayant une cardinalité de plusieurs à plusieurs. Il y a cependant deux cas distincts :

Un abandonneur lié à plusieurs files d'attentes : cela représente le cas où l'algorithme d'abandon pour l'abandonneur implique des entrées de plus d'une file d'attente. L'abandonneur n'élimine cependant que d'une file d'attente, celle à laquelle il est lié par une association NextService. Mais la décision d'abandon peut être influencée par l'état de plusieurs files d'attente. Pour les classes HeadTailDropper et HeadTailDropQueueBinding, la règle pour combiner les multiples entrées est la simple addition : si la somme des longueurs des files d'attente surveillées excède la valeur de QueueThreshold de l'abandonneur,

les paquets sont alors éliminés. Cette règle de combinaison des entrées peut cependant être supplantée par une règle différente dans les sous classes d'une de ces classes ou des deux.

Une file d'attente liée à plusieurs abandonneurs : cela représente le cas où l'état d'une file d'attente (qui est normalement aussi la file d'attente de laquelle les paquets sont éliminés) fournit une entrée à plusieurs algorithmes d'abandon d'abandonneurs. Un cas d'utilisation est ici un classeur qui partage un flux de trafic en, disons, quatre parts, représentant quatre classes de trafic. Chacune des parts passe par un HeadTailDropper séparé, puis est refusionnées dans la même file d'attente. Le résultat net est une seule file d'attente contenant des paquets provenant de quatre types de trafic, avec, disons, les seuils d'abandon suivants :

- Classe 1 – pleine à 90 %
- Classe 2 – pleine 80 %
- Classe 3 – pleine 70 %
- Classe 4 – pleine 50 %

Ici les pourcentages représentent l'état global de la file d'attente. Avec cette configuration, lorsque la file d'attente en question devient pleine à 50 %, les paquets de classe 4 seront éliminés plutôt que joints à la file d'attente, lorsque elle devient pleine à 70 %, les paquets des classes 3 et 4 seront éliminés, etc.

Les deux cas décrits ici peuvent aussi se produire ensemble, si un abandonneur reçoit des entrées provenant de plusieurs files d'attente, une ou plusieurs d'entre elles peuvent aussi fournir des entrées à d'autres abandonneurs.

3.10.2 Configuration des abandonneurs RED

Comme un abandonneur de queue, un abandonneur RED, représenté par une instance de la classe REDDropperService, peut prendre en entrées les états de plusieurs files d'attente. Dans ce cas, cependant, il y a une étape supplémentaire : chacune de ces entrées peut être lissée avant que l'abandonneur RED ne l'utilise, et le processus de lissage lui-même doit être paramétré. Par conséquent, en plus de REDDropperService et QueuingService, une troisième classe, DropThresholdCalculationService, est introduite, pour représenter la paramétrisation par file d'attente de ce processus de lissage.

Le diagramme qui suit illustre comment ces classes fonctionnent ensemble :

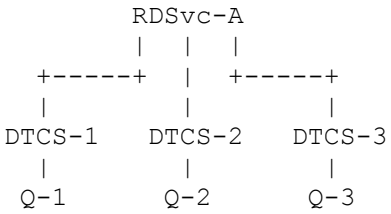


Figure 4 : Entrées pour un abandonneur RED

Ainsi REDDropperService-A (RDSvc-A) utilise des entrées de trois files d'attentes pour prendre sa décision d'élimination. (Comme toujours, RDSvc-A est lié à la file d'attente de laquelle il élimine les paquets via l'association NextService.) Pour chacune de ces trois files d'attente, il y a une instance de service de calcul de seuil d'abandon (DTCS, DropThresholdCalculationService) qui représente la pondération de lissage et l'intervalle de temps à utiliser lorsque on examine cette file d'attente. Donc chaque instance de DTCS est liée à exactement une file d'attente, bien qu'une seule file d'attente puisse être examinée (avec des valeurs de pondération et de temps différentes) par plusieurs instances de DTCS. Aussi, une instance de DTCS et la file d'attente derrière elle peut être vue comme une "unité de réutilisation". Ainsi une seule DTCS peut être référencée par plusieurs RDSvc.

Sauf si elle est écrasée par une règle différente dans une sous classe de REDDropperService, la règle qu'un abandonneur RED utilise pour combiner les entrées lissées du DTCS afin de créer une valeur à utiliser pour prendre une décision d'abandon est une simple addition.

3.11 Modélisation des files d'attentes et des programmateurs

Pour apprécier la raison de ce modèle assez complexe à programmer, on doit considérer la nature assez complexe des programmateurs, ainsi que les grandes variations des algorithmes et des mises en œuvre. Bien que ces variations soient larges, on a identifié quatre exemples qui servent d'essais du modèle et justifient sa complexité.

3.11.1 Programmateur hiérarchique simple

Un programmateur hiérarchique simple a les propriétés suivantes. D'abord, lorsque une opportunité de programmation est

donnée à un ensemble de files d'attente, une seule file d'attente viable est déterminée sur la base de certains critères de programmation, comme la bande passante ou la priorité. Le résultat du programmateur est l'entrée d'un autre programmateur qui traite le premier programmateur (et ses files d'attentes) comme une seule file d'attente logique. Donc, si le premier programmateur a déterminé le paquet approprié à libérer sur la base d'une priorité allouée à chaque file d'attente, le second programmateur peut spécifier une limite/allocation de bande passante pour l'ensemble entier de files d'attente agrégées par le premier programmateur.

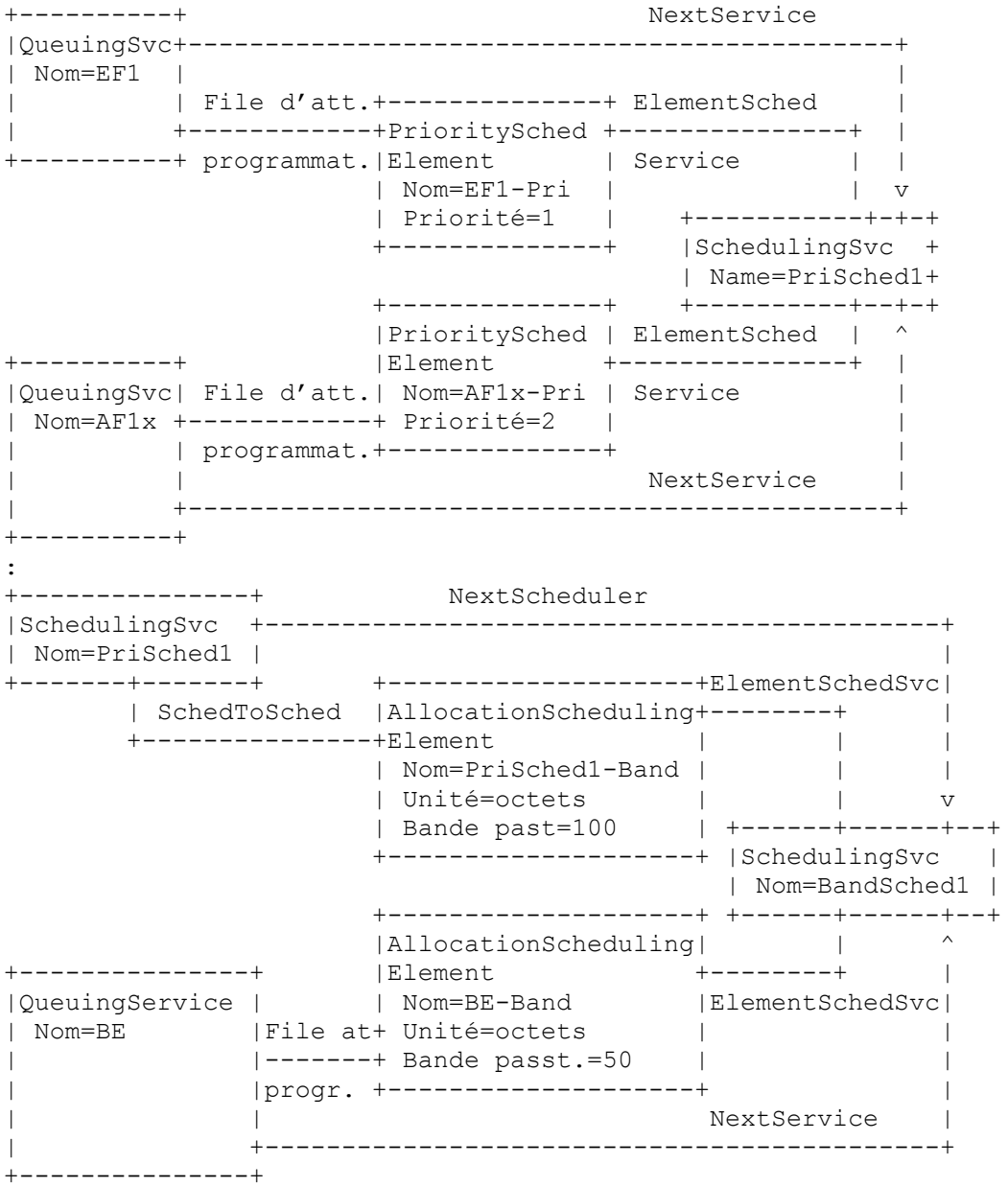


Figure 5. Exemple 1 : Programmateur hiérarchique simple

La Figure 5 illustre l'exemple et comment il va instancier le modèle. Dans la figure, NextService détermine le premier programmateur après la file d'attente. NextScheduler détermine l'ordre suivant des programmateurs. De plus, l'association ElementSchedulingService détermine l'ensemble de paramètres de programmation utilisé par un programmateur spécifique. Les paramètres de programmation peuvent être liés aux files d'attentes ou aux programmateurs. Dans le cas de SchedulingElement EF1-Pri, le lien est avec une file d'attente, de sorte que l'association QueueToSchedule est utilisée. Dans le cas de SchedulingElement PriSched1-Band, le lien est avec un autre programmateur, de sorte que l'association SchedulerToSchedule est utilisée. Noter qu'à cause des contraintes d'espace du document, le SchedulingService PRISched1 est représenté deux fois, pour montrer comment il est connecté à tous les autres objets.

3.11.2 Programmateur hiérarchique complexe

Un programmateur hiérarchique complexe a les mêmes caractéristiques qu'un programmateur simple, excepté que les critères

pour le second programmateur sont déterminés sur la base de la file d'attente plutôt que sur la base d'une agrégation. Un scénario peut être un ensemble de programmateurs à priorité liés. Dans ce cas, chaque file d'attente reçoit une priorité relative. Cependant, aucune file d'attente ne peut excéder une allocation de bande passante qui est unique pour cette file d'attente. Pour la prise en charge de ce scénario, la file d'attente doit être liée à deux programmateurs séparés. La Figure 6 illustre cette situation, en décrivant une file d'attente EF et une file d'attente au mieux (BE) qui pointent toutes deux sur un programmateur de priorité via l'association NextService. L'association NextScheduler entre le programmateur de priorité et le programmateur de bande passante définit à son tour l'ordre de la hiérarchie de programmation. Noter aussi que chaque programmateur a un ensemble distinct de paramètres de programmation qui sont liés en retour à chaque file d'attente. Cela montre la nécessité de la prise en charge de deux ensembles de paramètres ou plus par file d'attente.

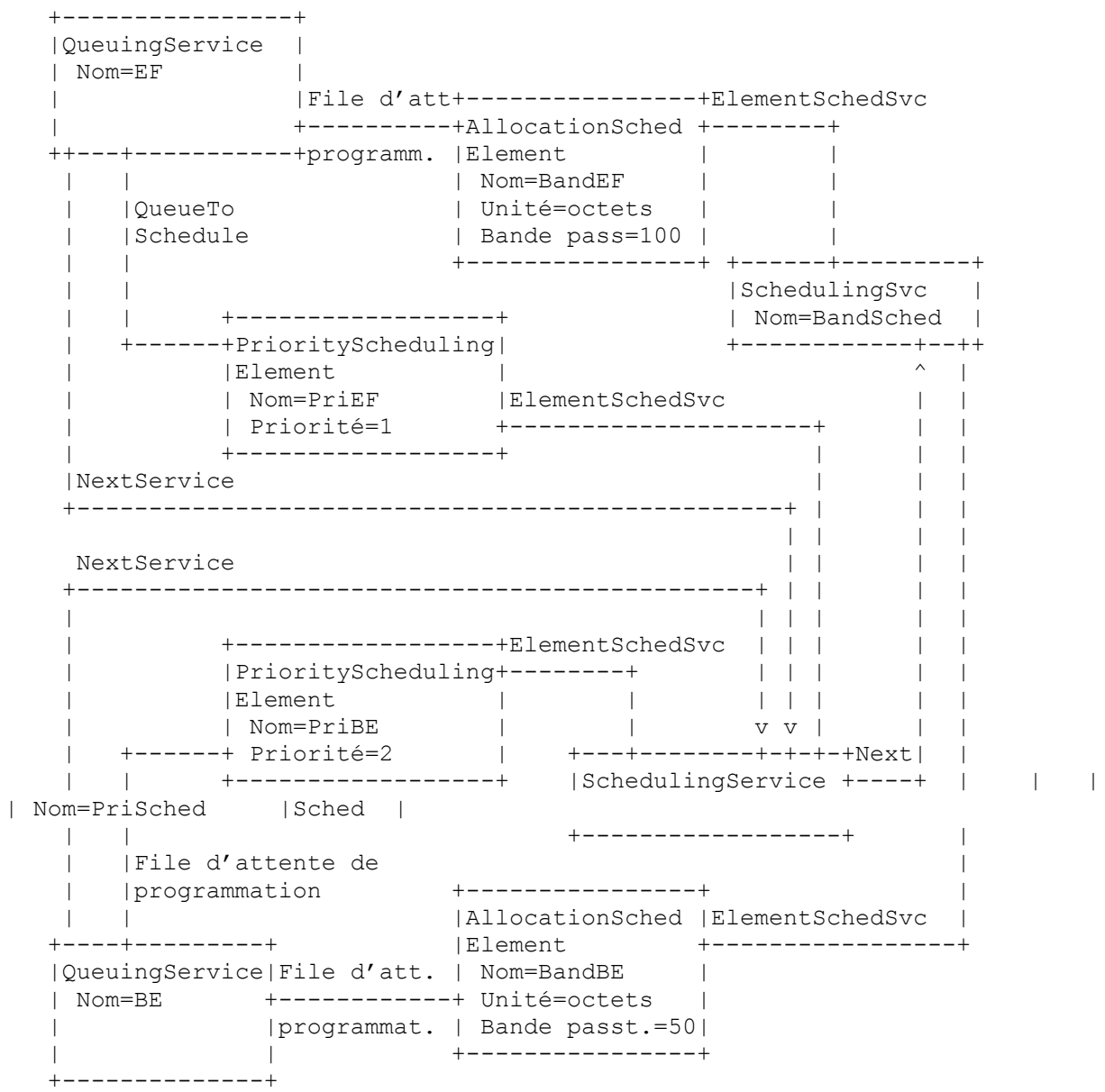


Figure 6. Exemple 2 : Programmateur hiérarchique complexe

3.11.3 Programmateur de capacité excédentaire

Un programmateur de capacité excédentaire présente des exigences similaires pour prendre en charge deux ensembles de paramètres de programmation par file d'attente. Cependant, dans ce scénario, les raisons sont un peu différentes. Supposons qu'un ensemble de files d'attente aient chacune des limites de bande passante allouées pour s'assurer qu'aucune classe de trafic n'étouffe une autre classe de trafic. Le résultat peut être qu'une ou plusieurs files d'attente ont excédé leur allocation alors que des files d'attente qui mériteraient une opportunité de programmation sont vides.

La question est alors de combien est l'excès (au repos) de bande passante allouée. On peut concevoir que les critères pour l'excès de capacité sont complètement différents de ceux qui déterminent les allocations en cas de charge uniforme. Ceci

pourrait être pris en charge par une programmation hiérarchisée. Cependant, le problème est que les critères pour utiliser le programmateur suivant sont différents de ceux des deux derniers cas. Précisément, le prochain programmateur devrait seulement être utilisé si une opportunité de programmation existe qui a été passée par le programmateur précédent.

Lorsque un programmateur choisit de renoncer à une décision de programmation, il se comporte comme un programmateur qui ne conserve pas le travail. Les programmateurs qui conservent le travail, par définition, vont toujours tirer parti d'une opportunité de programmation, sans considération de la file d'attente qui est desservie et de la quantité de bande passante qu'elle a consommé dans le passé. Ce point conduit à une réflexion intéressante. La sémantique d'un programmateur qui ne conserve pas le travail est équivalente à celle d'un mesureur, en ce que si un paquet est dans un profil, il a une opportunité de programmation, et si il est hors du profil, il n'a pas d'opportunité de programmation. Cependant, avec les mesureurs il y a une sémantique qui détermine la prochaine action à accomplir lorsque le paquet est dans le profil et lorsque le paquet est hors du profil. De façon similaire, avec le programmateur qui ne conserve pas le travail, il faut qu'il y ait un moyen pour déterminer le prochain programmateur lorsque un programmateur choisit de ne pas utiliser une opportunité de programmation.

La Figure 7 illustre ce dernier scénario. Elle paraît très similaire à la Figure 6, sauf que les liens entre le programmateur d'allocation et le programmateur à pondération comparative (WRR, *weighted round-robin*) utilisent une association FailNextScheduler. Cette association indique explicitement le fait que la seule fois où le programmateur WRR sera utilisé est lorsque il y a des files d'attente non vides que le programmateur d'allocation a rejetées pour des considérations de programmation. Noter que la Figure 7 est incomplète, en ce que normalement il y aurait plusieurs autres files d'attente liées à un programmeur d'allocation et à WRR.

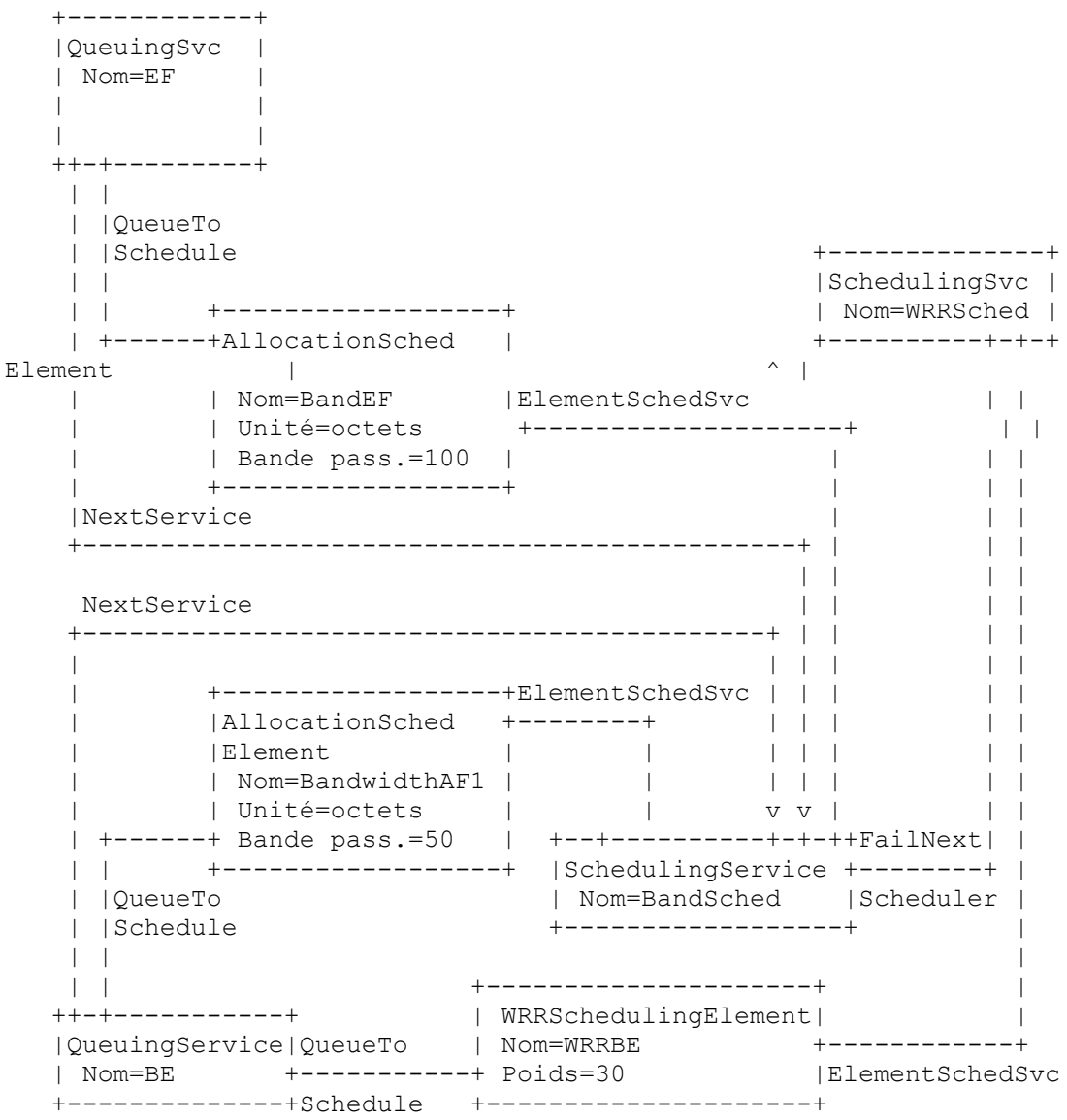


Figure 7. Exemple 3 : Programmateur de capacité excédentaire

3.11.4 Programmateur hiérarchique CBQ

Un programmateur hiérarchique à mise en file d'attente fondée sur la classe (CBQ, *class-based queuing*) est le quatrième scénario à considérer. Dans la CBQ hiérarchique, chaque file d'attente reçoit une allocation spécifique de bande passante. Les files d'attente sont groupées en un programmateur logique. Ce programmateur logique est à son tour une allocation de bande passante agrégée qui est égale à la somme des files d'attente qu'il programme. À leur tour, les programmateurs logiques peuvent être agrégés en programmateurs logiques de niveau supérieur. En prenant une perspectives de haut en bas, le programmateur logique du haut a 100 % de la capacité de la liaison. Cette allocation est divisée entre les programmateurs logiques en dessous de lui de telle sorte que la somme des allocations soit égale à 100 %. Ces programmateurs de second niveau peuvent à leur tour morceler leur allocation entre les programmeurs de troisième rang et ainsi de suite jusqu'au dernier niveau qui morcelle ses allocations entre des files d'attente spécifiques représentant des classes d'une granularité relativement fine de trafic. L'aspect unique de la CBQ hiérarchique est que lorsque la bande passante est insuffisante pour une allocation spécifique, les programmateurs les plus hauts dans l'arborescence sont essayés pour voir si une autre portion de l'arborescence aurait des capacités inutilisées.

La Figure 8 montre cet exemple avec deux niveaux. L'exemple est partagé en deux pour des contraintes d'espace, d'où il résulte que l'instance de service de programmation CBQTier1 est représentée deux fois. Noter que l'allocation totale au niveau supérieur est de 50 Mbit. L'allocation vocale est de 22 Mbit. Les 23 Mbit restants sont partagés entre FTP et la Toile. Donc, si le trafic de la Toile consomme en fait 20 Mbit (5 Mbit en excès de l'allocation) si FTP consomme 5 Mbit, il est alors possible au programmateur CBQTier1 d'offrir 3 Mbit de son allocation au trafic de la Toile. Cependant, ce n'est pas assez, de sorte que l'association FailNextScheduler doit être traversée pour déterminer si il y aurait des capacités en excès disponibles dans la classe vocale. Si la classe vocale consomme seulement 15 Mbit de ses 22 Mbit d'allocation, il y a des ressources suffisantes pour permettre de passer le trafic de la Toile. Noter que FailNextScheduler est utilisé comme association. La raison en est que le programmateur CBQTier1 a en fait échoué à programmer un paquet à cause de l'insuffisance de ressources. On peut concevoir qu'une variante de CBQ hiérarchique permette aussi une programmation réussie. Donc, les deux associations sont nécessaires.

Noter qu'à cause des contraintes d'espace du document, le service de programmation CBQTier1 est représenté deux fois, pour montrer comment il est connecté à tous les autres objets.

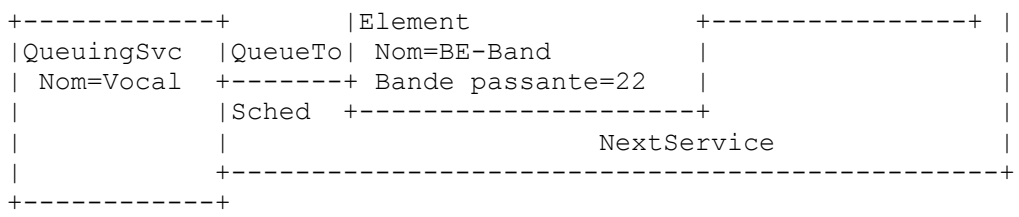


Figure 8. Exemple 4 : Programmateur hiérarchique CBQ

4. Hiérarchie des classes

Les paragraphes qui suivent présentent les hiérarchies de classes et d'associations qui ensemble constituent le modèle d'informations pour modéliser les capacités de qualité de service au niveau de l'appareil.

4.1 Associations et agrégations

Les associations et agrégations sont un moyen de représenter les relations entre deux objets (ou théoriquement plus). Dépendance, agrégation, et autres relations, sont modélisées comme des classes contenant deux (ou plus) références d'objet. On devrait noter que les agrégations représentent des relations soit "tout-partie" soit de "collection". Par exemple, l'agrégation peut être utilisée pour représenter la relation de contenance entre un système et les composants qui constituent le système.

Comme les associations et les agrégations sont des classes, elles peuvent bénéficier de toutes les caractéristiques orientées objet qu'ont d'autres classes non relationnelles. Par exemple, elles peuvent contenir des propriétés et des méthodes, et l'héritage peut être utilisé pour préciser leur sémantique de façon qu'elles représentent des types plus spécialisés de leurs super classes.

Noter qu'un objet d'association (ou d'agrégation) est traité comme une unité atomique (une instance individuelle) même si elle se rapporte à, ou collecte, de multiples objets. C'est une caractéristique constitutive d'une association (ou d'une agrégation) – bien que les éléments individuels qui se rapportent aux autres objets aient leur propre identité, l'objet d'association (ou d'agrégation) qui est construit en utilisent ces objets a aussi sa propre identité et son propre nom.

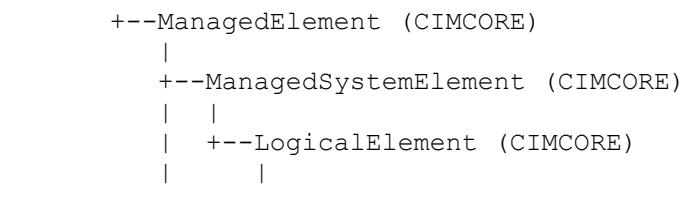
Il est important de noter que les associations et agrégations forment une hiérarchie d'héritage qui est séparée de la hiérarchie d'héritage de classe. Bien que les associations et agrégations soient normalement bidirectionnelles, il n'y a rien qui empêche de définir des associations ou agrégations d'ordre supérieur. Cependant, de telles associations et agrégations sont par nature plus complexes à définir, comprendre, et utiliser. En pratique, les associations et agrégations d'ordre plus élevé que binaire sont rarement utilisées, à cause de leur complexité accrue et de leur manque de généralité. Toutes les associations et agrégations définies dans le présent modèle sont binaires.

Noter aussi que par définition, les associations et agrégations ne peuvent pas être unaires.

On note finalement qu'associations et agrégations définies entre deux classes n'affectent pas les classes elles-mêmes. C'est-à-dire que l'ajout ou la suppression d'une association ou agrégation n'affecte pas les interfaces des classes qu'elle connecte.

4.2 Structure des hiérarchies de classe

La structure des hiérarchies d'héritage de classe, association, et agrégation pour gérer les chemins des données des appareils de qualité de service est montrée, respectivement, dans les Figures 9, 10, et 11. La notation (CIMCORE) identifie une classe définie dans le modèle de cœur de CIM. Prière de se reporter à [CIM] pour la définition de ces classes. De même, la notation [RFC3460] identifie une classe définie dans le document Extensions au modèle d'informations de cœur de politique. Ce modèle a été influencé par [CIM], et est compatible avec l'effort Réseaux à répertoire activé (DEN, *Directory Enabled Network*).




+--Service (CIMCORE)
+--ConditioningService
+--ClassifierService
+--ClassifierElement
+--MeterService
+--AverageRateMeterService
+--EWMAMeterService
+--TokenBucketMeterService
+--MarkerService
+--PreambleMarkerService
+--TOSMarkerService
+--DSCPMarkerService
+--ManagedElement (CIMCORE)
+--ManagedSystemElement (CIMCORE)
+--LogicalElement (CIMCORE)
+--Service (CIMCORE)
+--8021QMarkerService
+--DropperService
+--HeadTailDropperService
+--RedDropperService
+--QueuingService
+--PacketSchedulingService
+--NonWorkConservingSchedulingService
+--QoSService
+--DiffServService
+--AFService
+--FlowService
+--DropThresholdCalculationService
+--FilterEntryBase [RFC3460]
+--IPHeaderFilter [RFC3460]
+--8021Filter [RFC3460]
+--PreambleFilter
+--FilterList [RFC3460]

```
   |     |
   |     +--ServiceAccessPoint (CIMCORE)
   |        |
   |        +--ProtocolEndpoint
+--ManagedElement (CIMCORE)
   |
   +--ManagedSystemElement (CIMCORE)
   |  |
   |  +--LogicalElement (CIMCORE)
   |     |
   |     +--Service (CIMCORE)
   |
   +--Collection (CIMCORE)
   |  |
   |  +--CollectionOfMSEs (CIMCORE)
   |     |
   |     +--BufferPool
   |
   +--SchedulingElement
      |
      +--AllocationSchedulingElement
      |
      +--WRRSchedulingElement
      |
      +--PrioritySchedulingElement
         |
         +--BoundedPrioritySchedulingElement
```

Figure 9 : Hiérarchie d'héritage de classe

La hiérarchie d'héritage pour les associations définies dans le présent document est montrée à la Figure 10.

```
+--Dependency (CIMCORE)
|  |
|  +--ServiceSAPDependency (CIMCORE)
|  |  |
|  |  +--IngressConditioningServiceOnEndpoint
|  |  |
|  |  +--EgressConditioningServiceOnEndpoint
|  |
|  +--HeadTailDropQueueBinding
|  |
|  +--CalculationBasedOnQueue
|  |
|  +--ProvidesServiceToElement (CIMCORE)
|  |  |
|  |  +--ServiceServiceDependency (CIMCORE)
|  |     |
|  |     +--CalculationServiceForDropper
|  |
|  +--QueueAllocation
|  |
|  +--ClassifierElementUsesFilterList
|
+--AFRelatedServices
|
+--NextService
|  |
|  +--NextServiceAfterClassifierElement
|  |
|  +--NextScheduler
|    |
|    +--FailNextScheduler
|
```

```
+--NextServiceAfterMeter
|
+--QueueToSchedule
|
+--SchedulingServiceToSchedule
```

Figure 10. Hiérarchie d'héritage de classe d'association

La hiérarchie d'héritage pour les agrégations définies dans le présent document est montrée à la Figure 11.

```
+--MemberOfCollection (CIMCORE)
|  |
|  +--CollectedBufferPool
|
+--Component (CIMCORE)
|  |
|  +--ServiceComponent (CIMCORE)
|  |  |
|  |  +--QoSSubService
|  |  |
|  |  +--QoSConditioningSubService
|  |  |
|  |  +--ClassifierElementInClassifierService
|  |
|  +--EntriesInFilterList [RFC3460]
|
+--ElementInSchedulingService
```

Figure 11 : Hiérarchie d'héritage de classe d'agrégation

4.3 Définition des classes

Ce paragraphe présente les classes et propriétés qui constituent le modèle d'informations pour décrire les fonctionnalités en rapport avec la qualité de service dans les appareils réseau, incluant les hôtes. Ces définitions sont dérivées de définitions du modèle cœur de CIM [CIM]. Seules les classes en rapport avec la qualité de service sont définies dans le présent document. Cependant, d'autres classes tirées du modèle cœur de CIM, ainsi que de la [RFC3460], sont décrites brièvement. Le lecteur est invité à se reporter à [CIM] et la [RFC3460] pour plus d'informations. Les associations et agrégations sont définies au paragraphe 4.4.

4.3.1 Classe abstraite ManagedElement

C'est une classe abstraite définie dans le modèle cœur de CIM. C'est la racine de la hiérarchie d'héritage de classe entière de CIM. Parmi les associations qui se réfèrent à elle, il y en a deux qui sont des sous classes dans le présent document : Dependency et MemberOfCollection, qui est une agrégation. Les propriétés de ManagedElement sont Caption et Description. Toutes deux sont des chaînes de forme libre pour décrire une instance d'objet. Prière de se référer à [CIM] pour la définition complète de cette classe.

4.3.2 Classe abstraite ManagedSystemElement

C'est une classe abstraite définie dans le modèle de cœur de CIM ; c'est une sous classe de ManagedElement. ManagedSystemElement sert de classe de base pour les hiérarchies de classe PhysicalElement et LogicalElement. LogicalElement, à son tour, est la classe de base pour un nombre important de hiérarchies CIM, incluant System. Tout composant distinguable d'un système est un candidat à l'inclusion dans cette hiérarchie de classes, incluant des composants physiques (par exemple, les processeurs et les cartes) et des composants logiques (par exemple, des composants logiciels, services, et autres objets).

Aucune des associations auxquelles participe cette classe n'est utilisée directement dans le modèle d'état d'appareil de qualité de service. Cependant, l'agrégation Component, qui met en rapports un ManagedSystemElement avec un autre, est la classe de base pour les deux agrégations qui forment le cœur du modèle d'état d'appareil de qualité de service : QoSSubService et QoSConditioningSubService. De façon similaire, l'association ProvidesServiceToElement, qui met en rapports un ManagedSystemElement avec un Service, est la classe de base pour l'association CalculationServiceForDropper du modèle.

Prière de se référer à [CIM] pour la définition complète de cette classe.

4.3.3 Classe abstraite LogicalElement

C'est une classe abstraite définie dans le modèle de cœur de CIM. C'est une sous classe de la classe ManagedSystemElement, et c'est la classe de base pour tous les composants logiques d'un système géré, tel que des fichiers, des processus, ou des capacités de système sous la forme d'appareils et services logiques. Aucune des associations auxquelles cette classe participe ne relève du modèle d'état d'appareil de qualité de service. Prière de se référer à [CIM] pour la définition complète de cette classe.

4.3.4 Classe abstraite Service

C'est une classe abstraite définie dans le modèle de cœur de CIM. C'est une sous classe de la classe LogicalElement, et c'est la classe de base pour tous les objets qui représentent un "service" ou fonctionnalité dans un système. Un Service est un objet d'utilisation générale qui est utilisé pour configurer et gérer la mise en œuvre de la fonctionnalité. Comme on l'a noté ci-dessus au paragraphe 4.3.2, cette classe participe à l'association ProvidesServiceToElement. Prière de se référer à [CIM] pour la définition complète de cette classe.

4.3.5 Classe ConditioningService

C'est une sous classe concrète du cœur de service de classe de CIM ; elle représente la capacité de définir comment le trafic est conditionné dans le chemin de transmission des données d'un appareil. Les sous classes de ConditioningService définissent les types particuliers de conditionnement qui sont faits. Six types fondamentaux de conditionnement sont définis dans le présent document. Ce sont les services effectués par un classeur, un mesureur, un marqueur, un abandonneur, une file d'attente, et un programmeur. D'autres types, plus sophistiqués de conditionnement pourront être définis dans de futurs documents.

ConditioningService est une classe concrète parce que au moment où elle a été définie dans CIM, sa super classe était concrète. Bien que cette classe puisse être instanciée, une de ses instances ne pourrait rien réaliser, à cause de la nature du conditionnement, et des paramètres qui le contrôlent, qui ne sont spécifiés que dans les sous classes de ConditioningService.

Deux associations auxquelles participe ConditioningService sont critiques pour son utilisation dans la QS - QoSConditioningSubService et NextService. QoSConditioningSubService agrège les ConditioningServices en un service de QS particulier (comme AF) pour décrire la fonctionnalité de conditionnement spécifique qui sou tend ce service de QS dans un appareil particulier. NextService indique le ou les services de conditionnement suivants pour les différents flux de trafic. La définition de cette classe est la suivante :

NOM : ConditioningService
DESCRIPTION : classe concrète pour définir comment le trafic est conditionné dans le chemin de transmission des données d'un hôte ou appareil réseau.
DÉRIVÉE DE : Service
TYPE : Concret
PROPRIÉTÉS : (aucune)

4.3.6 Classe ClassifierService

Le concept d'un classeur vient de la [RFC3290]. ClassifierService est une classe concrète qui représente une entité logique dans une interface d'entrée ou de sortie d'un appareil, qui prend un seul flux d'entrée, et le trie en un ou plusieurs flux de sortie. Le tri est fait par un ensemble de filtres qui choisissent les paquets sur la base de leur contenu, ou éventuellement d'autres attributs associés au paquet. Chaque flux de sortie est le résultat de la satisfaction d'un filtre particulier.

La représentation des classeurs dans QDDIM est en relation étroite avec celle présentée dans les [RFC3289] et [RFC3290]. Plutôt que d'être relié directement à ses FilterLists, un classeur est modélisé ici comme une agrégation de ClassifierElements. Chacun de ces ClassifierElements est alors relié à une seule FilterList, par l'association ClassifierElementUsesFilterList.

Un classeur est modélisé comme sous classe de ConditioningService afin qu'il puisse être agrégé dans un QoSService (en utilisant l'agrégation QoSConditioningSubService) et puisse utiliser l'association NextService pour identifier les objets ConditioningService suivants pour les différents flux de trafic.

ClassifierService est conçu pour permettre une classification hiérarchique. Lorsque une classification hiérarchique est utilisée, un ClassifierElement peut pointer sur un autre ClassifierService. Lorsque elle est utilisée à cette fin, ClassifierElement ne doit pas utiliser l'association ClassifierElementUsesFilterList. La définition de cette classe est la suivante :

NOM : ClassifierService
DESCRIPTION : classe concrète qui décrit comment un flux de trafic d'entrée est trié en plusieurs flux de sortie en utilisant un ou plusieurs filtres.
DÉRIVÉE DE : ConditioningService
TYPE : Concret
PROPRIÉTÉS : (aucune)

4.3.7 Classe ClassifierElement

Le concept de ClassifierElement vient de la [RFC3289]. Cette classe concrète représente le lien, au sein d'un seul ClassifierService, entre une FilterList qui spécifie un ensemble de critères pour choisir les paquets dans un flux de paquets entrant dans le ClassifierService, et le prochain ConditioningService dans lequel vont les paquets choisis après qu'ils ont quitté le ClassifierService. ClassifierElement n'a pas de propriétés en propre. Il est présent pour servir à ancrer une agrégation à son classeur, et pour les associations à leur FilterList et au prochain ConditioningService.

Lorsque un ClassifierElement est associé à un ClassifierService par l'association NextServiceAfterClassifierElement, le ClassifierElement peut ne pas utiliser l'association ClassifierElementUsesFilterList. De plus, lorsque un ClassifierElement est associé à un ClassifierService comme décrit ci-dessus, l'ordre de traitement du ClassifierService associé est une fonction de la propriété ClassifierOrder de l'agrégation ClassifierElementInClassifierService. Par exemple, supposons ce qui suit :

1. ClassifierService (C1) agrège les ClassifierElements (E1), (E2) et (E3) avec les valeurs relatives de ClassifierOrder de 1, 2, et 3.
2. Les associations ClassifierElements (E1) et (E3) avec les FilterLists respectivement (F1) et (F3) utilisent l'association ClassifierElementUsesFilterList.
3. (E1) et (E3) sont associés aux mesureurs (M1) et (M3) à travers leurs associations respectives NextServiceAfterClassifierElement.
4. (E2) est associé à ClassifierService (C2) à travers son association NextServiceAfterClassifierElement.
5. ClassifierService (C2) agrège ClassifierElements (E4) et (E5) avec les valeurs relatives de ClassifierOrder de 1 et 2.
6. Les ClassifierElements (E4) et (E5) ont des associations aux FilterLists respectivement (F4) et (F5) en utilisant l'association ClassifierElementUsesFilterList.

Dans cet exemple, le traitement des paquets satisferait aux FilterLists dans l'ordre de (F1), (F4), (F5), et (F3).
La définition de cette classe est la suivante :

NOM : ClassifierElement
DESCRIPTION : classe concrète représentant le processus par lequel un classeur utilise un filtre pour choisir les paquets à transmettre à un prochain service de conditionnement spécifique.
DÉRIVÉE DE : ClassifierService
TYPE : Concret
PROPRIÉTÉS : (aucune)

4.3.8 Classe MeterService

C'est une classe concrète qui représente le métrage du trafic réseau. Le métrage est la fonction de surveillance des heures d'arrivée des paquets d'un flux de trafic, et de détermination du niveau de conformité de chaque paquet à un profil de trafic préétabli. Un mesureur a la capacité d'invoquer différents ConditioningServices pour le trafic conforme et non conforme. Le trafic qui quitte un mesureur peut recevoir un conditionnement supplémentaire (par exemple, abandonné ou mis en file d'attente) en acheminant le paquet sur un autre élément de conditionnement. Prière de se reporter à la [RFC3290] pour plus d'informations sur le métrage.

Cette classe est la classe de base pour définir différents types de mesureurs. À ce titre, elle contient des propriétés communes partagées par toutes les sous classes de mesureurs. Elle est modélisée comme un ConditioningService de sorte qu'elle peut être agrégée dans un QoSService (en utilisant l'association QoSConditioningSubService) pour indiquer que sa fonctionnalité sous-tend ce service de QS. MeterService participe aussi à l'association NextServiceAfterMeter, pour identifier les objets ConditioningService suivants pour le trafic conforme et non conforme. La définition de cette classe est la suivante :

NOM : MeterService
DESCRIPTION : classe concrète qui décrit la surveillance du trafic par rapport à un profil pré établi.
DÉRIVÉE DE : ConditioningService
TYPE : Concret
PROPRIÉTÉS : MeterType, OtherMeterType, ConformanceLevels

Note : La propriété MeterType et les sous classes MeterService fournissent des informations similaires. La propriété

MeterType est définie pour les besoins d'interrogation et pour une future expansion. Il est possible que tous les MeterServices n'exigent pas qu'une sous classe les définisse. Dans ce cas, MeterService sera instancié directement, et la propriété MeterType fournira la seule façon d'identifier le type de mesureur.

4.3.8.1 Propriété MeterType

Cette propriété est un entier non signé de 16 bits qui est utilisé pour spécifier le type particulier de mesureur représenté par une instance de MeterService. Les valeurs d'énumération suivantes sont définies :
1 – autre
2 – mesureur de taux moyen
3 – mesureur de moyenne mobile à pondération exponentielle
4 – mesureur de baquet de jetons

Note : si la valeur de MeterType n'est pas une de ces quatre valeurs, elle DEVRAIT être interprétée comme étant '1' (autre).

4.3.8.2 Propriété OtherMeterType

C'est une propriété de chaîne qui définit une description d'un type de mesureur spécifique d'un fabricant. Elle est utilisée lorsque la valeur de la propriété MeterType dans l'instance est égale à 1.

4.3.8.3 Propriété ConformanceLevels

Cette propriété est un entier non signé de 16 bits. Elle indique le nombre de niveaux de conformité pris en charge par le mesureur. Par exemple, quand seule la mesure "dans le profil" par opposition à "hors profil" est prise en charge, ConformanceLevels est égale à 2.

4.3.9 Classe AverageRateMeterService

C'est une sous classe concrète de MeterService qui représente un simple mesureur, appelé un mesureur de taux moyen. Ce type de mesureur mesure le taux moyen auquel les paquets lui sont soumis sur une période spécifiée. Les paquets sont définis comme conformes si leur taux d'arrivée moyen n'excède pas le taux de mesure spécifié du mesureur. Tout paquet qui cause le dépassement du taux de mesure spécifié est défini comme non conforme. Pour plus d'informations, voir la [RFC3290]. La définition de cette classe est la suivante :

NOM : AverageRateMeterService
DESCRIPTION : classe concrète qui classe le trafic en conforme ou non conforme, selon que l'arrivée d'un paquet cause le dépassement d'une valeur prédéterminée du taux moyen d'arrivée.
DÉRIVÉE DE : MeterService
TYPE : Concret
PROPRIÉTÉS : AverageRate, DeltaInterval

4.3.9.1 Propriété AverageRate

C'est un entier non signé de 32 bits qui définit le taux utilisé pour déterminer si les paquets admis sont conformes ou non. La valeur est spécifiée en kilobits par seconde.

4.3.9.2 Propriété DeltaInterval

C'est un entier non signé de 64 bits qui définit la durée pendant laquelle la mesure moyenne devrait être prise. La valeur est spécifiée en microsecondes.

4.3.10 Classe EWMAMeterService

C'est une sous classe concrète de la classe MeterService qui représente une mesure de moyenne mobile à pondération exponentielle. Ce mesureur est un simple filtre passe bas qui mesure le taux d'entrée des paquets sur un petit intervalle d'échantillonnage fixé. Tout paquet admis qui pousse le taux moyen au delà d'une limite prédéfinie est défini comme non conforme. Voir la [RFC3290] pour plus d'informations. La définition de cette classe est la suivante :

NOM : EWMAMeterService
DESCRIPTION : classe concrète qui classe le trafic admis en conforme ou non conforme selon que l'arrivée d'un paquet cause

ou non le dépassement par le taux d'arrivée moyen dans un petit intervalle d'échantillonnage fixé d'une valeur prédéterminée.
DÉRIVÉE DE : MeterService
TYPE : Concret
PROPRIÉTÉS : AverageRate, DeltaInterval, Gain

4.3.10.1 Propriété AverageRate

Cette propriété est un entier non signé de 32 bits qui définit le taux moyen par rapport auquel le taux moyen de l'échantillon de paquets devrait être mesuré. Tout paquet qui cause le dépassement du taux échantillonné est réputé non conforme. La valeur est spécifiée en kilobits par seconde.

4.3.10.2 Propriété DeltaInterval

Cette propriété est un entier non signé de 64 bits qui définit l'intervalle d'échantillonnage utilisé pour mesurer le taux d'arrivée. Le taux calculé est moyen sur cet intervalle et confronté à la propriété AverageRate. Tous les paquets dont le taux d'arrivée moyen calculé est inférieur à la valeur de AverageRate sont réputés conformes. La valeur est spécifiée en microsecondes.

4.3.10.3 Propriété Gain

Cette propriété est un entier non signé de 32 bits représentant l'inverse de la constante de temps (par exemple, la réponse en fréquence) de ce qui est essentiellement un simple filtre passe bas. Par exemple, la valeur de 64 pour cette propriété représente une valeur de constante de temps de 1/64.

4.3.11 Classe TokenBucketMeterService

C'est une sous classe concrète de la classe MeterService qui représente la mesure du trafic réseau en utilisant un baquet de jetons. Deux types de mesures de baquet de jetons sont définies en utilisant cette classe – une mesure simple de baquet à deux paramètres, et une mesure à plusieurs étapes.

Un simple baquet de jetons a généralement deux paramètres, un débit de jetons moyen et une taille de salve, et a deux niveaux de conformité : "conforme" et "non conforme". Cette classe définit aussi une taille de salve par excès, qui permet au mesureur d'avoir trois niveaux de conformité ("conforme", "partiellement conforme", et "non conforme"). Dans ce cas, les paquets qui excèdent la taille de salve par excès sont réputés non conformes, tandis que les paquets qui excèdent la plus petite taille de salve mais sont inférieurs à la taille de salve par excès sont réputés partiellement conformes. Le fonctionnement de ces mesureurs est décrit dans la [RFC3290]. La définition de cette classe est la suivante :

NOM : TokenBucketMeterService
DESCRIPTION : classe concrète qui classe le trafic admis par rapport à un baquet de jetons. Deux ou trois niveaux de conformité peuvent être définis.
DÉRIVÉE DE : MeterService
TYPE : Concret
PROPRIÉTÉS : AverageRate, PeakRate, BurstSize, ExcessBurstSize

4.3.11.1 Propriété AverageRate

Cette propriété est un entier non signé de 32 bits qui spécifie les taux attribués au mesureur. La valeur est exprimée en kilobits par seconde.

4.3.11.2 Propriété PeakRate

Cette propriété est un entier non signé de 32 bits qui spécifie le taux de crête du mesureur. La valeur est exprimée en kilobits par seconde.

4.3.11.3 Propriété BurstSize

Cette propriété est un entier non signé de 32 bits qui spécifie le nombre maximum de jetons disponibles pour le taux attribué (spécifié par la propriété AverageRate). La valeur est exprimée en kilo-octets.

4.3.11.4 Propriété ExcessBurstSize

Cette propriété est un entier non signé de 32 bits qui spécifie le nombre maximum de jetons disponible pour le taux de crête (spécifié par la propriété PeakRate). La valeur est exprimée en kilo-octets.

4.3.12 Classe MarkerService

C'est une classe concrète qui représente le processus général de marquage de certains champs dans un paquet du réseau avec une certaine valeur. Les sous classes de MarkerService identifient des champs particuliers à marquer, et introduisent des propriétés pour représenter les valeurs à utiliser dans le marquage de ces champs. Les marqueurs sont généralement invoqués par suite d'une correspondance de classeur précédente. Le fonctionnement des divers types de marqueurs est décrit dans la [RFC3290].

MarkerService est une classe concrète parce que au moment où elle a été définie dans CIM, sa super classe était concrète. Bien que cette classe puisse être instanciée, une de ses instances ne pourrait rien accomplir parce que le champ à marquer et la valeur à utiliser ne sont spécifiés que dans les sous classes de MarkerService.

MarkerService est modélisé comme un ConditioningService, de sorte qu'il peut être agrégé dans un QoSService (en utilisant l'association QoSConditioningSubService) pour indiquer que sa fonctionnalité sous-tend ce service de QS. Elle participe à l'association NextService pour identifier l'objet ConditioningService suivant qui agit sur le trafic après qu'il a été marqué par le marqueur. La définition de cette classe est la suivante :

NOM : MarkerService
DESCRIPTION : classe concrète représentant le processus général de marquage d'un champ sélectionné dans un paquet avec une valeur spécifiée. Les paquets sont marqués dans l'ordre pour contrôler le conditionnement qu'ils vont recevoir ultérieurement.
DÉRIVÉE DE : ConditioningService
TYPE : Concret
PROPRIÉTÉS : (aucune)

4.3.13 Classe PreambleMarkerService

C'est une classe concrète qui modélise la mémorisation des résultats du conditionnement du trafic dans un préambule de paquet. Voir au paragraphe 3.8.3 la discussion de la façon dont, et des raisons pour lesquelles, QDDIM modélise la capacité de mémoriser ces résultats dans un préambule de paquet. Une instance de PreambleMarkerService ajoute à un préambule de paquet une chaîne en deux parties de la forme "<type>,<valeur>". Le paragraphe 3.8.3 donne une liste des chaînes <type> définies par QDDIM. Les mises en œuvre peuvent prendre en charge d'autres <type> en plus de celles-la. La définition de cette classe est la suivante :

NOM : PreambleMarkerService
DESCRIPTION : classe concrète représentant la sauvegarde des résultats de conditionnement de trafic dans un préambule de paquet.
DÉRIVÉE DE : MarkerService
TYPE : Concret
PROPRIÉTÉS : FilterItemList[ ]

4.3.13.1 Propriété multi valeurs FilterItemList

Cette propriété est une liste ordonnée de chaînes, où chaque chaîne a le format "<type>,<valeur>". Voir au paragraphe 3.8.3 la liste de <types> définie dans QDDIM, et la nature de la <valeur> associée pour chacun de ces types.

4.3.14 Classe ToSMarkerService

C'est une classe concrète qui représente le marquage du champ ToS dans l'en-tête de paquet IPv4 [RFC0791]. Suivant la pratique courante, la valeur à écrire dans ce champ est représentée par un entier non signé de 8 bits. La définition de cette classe est la suivante :

NOM : ToSMarkerService
DESCRIPTION : classe concrète représentant le processus de marquage du champ type de service (ToS) dans l'en-tête de paquet IPv4 avec une valeur spécifiée. Les paquets sont marqués afin de contrôler le conditionnement qu'ils vont recevoir ultérieurement.
DÉRIVÉE DE : MarkerService
TYPE : Concret

PROPRIÉTÉS : ToSValue

4.3.14.1 Propriété ToSValue

Cette propriété est un entier non signé de 8 bits, représentant une valeur à utiliser pour marquer le champ type de service (ToS) dans l'en-tête de paquet IPv4. Le champ ToS est défini comme un octet complet, de sorte que la gamme pour cette propriété est de 0 à 255. Certaines mises en œuvre exigent cependant que le bit de moindre poids dans le champ ToS soit toujours '0'. De telles mises en œuvre sont par conséquent incapables de prendre en charge une valeur de TosValue impaire.

4.3.15 Classe DSCPMarkerService

C'est une classe concrète qui représente le marquage du codet de service différencié (DSCP) au sein du champ DS dans les en-têtes de paquet IPv4 et IPv6, comme défini dans la [RFC2474]. Suivant les pratiques courantes, la valeur à écrire dans ce champ est représentée par un entier non signé de 8 bits. La définition de cette classe est la suivante :

NOM : DSCPMarkerService
DESCRIPTION : classe concrète représentant le processus de marquage du champ DSCP dans un paquet avec une valeur spécifiée. Les paquets sont marqués afin de contrôler le conditionnement qu'ils vont recevoir ensuite.
DÉRIVÉE DE : MarkerService
TYPE : Concret
PROPRIÉTÉS : DSCPValue

4.3.15.1 Propriété DSCPValue

Cette propriété est un entier non signé de 8 bits, représentant une valeur à utiliser pour marquer le DSCP au sein du champ DS dans l'en-tête de paquet IPv4 ou IPv6. Comme le DSCP consiste en 6 bits, les valeurs de cette propriété sont limitées à la gamme 0 à 63. Lorsque le DSCP est marqué, les deux bits restants du champ DS sont laissés inchangés.

4.3.16 Classe 8021QMarkerService

C'est une classe concrète qui représente le marquage du champ de priorité d'usager définie dans la spécification [IEEE802Q]. Suivant les pratiques courantes, la valeur à écrire dans ce champ est représentée par un entier non signé de 8 bits. La définition de cette classe est la suivante :

NOM : 8021QMarkerService
DESCRIPTION : classe concrète représentant le processus de marquage du champ Priorité dans une trame conforme à 802.1Q avec une valeur spécifiée. Les trames sont marquées afin de contrôler le conditionnement qu'elles vont ensuite recevoir.
DÉRIVÉE DE : MarkerService
TYPE : Concret
PROPRIÉTÉS : PriorityValue

4.3.16.1 Propriété PriorityValue

Cette propriété est un entier non signé de 8 bits, représentant une valeur à utiliser pour marquer le champ Priorité dans l'en-tête 802.1Q. Comme le champ Priorité consiste en 3 bits, les valeurs pour cette propriété sont limitées à la gamme 0 à 7. Quand le champ Priorité est marqué, les bits restants sont laissés inchangés dans cet octet.

4.3.17 Classe DropperService

C'est une classe concrète qui représente la capacité d'abandonner sélectivement du trafic réseau, ou d'invoquer un autre ConditioningService pour un autre traitement du trafic qui n'est pas éliminé. C'est la classe de base pour différents types d'abandonneurs. Les abandonneurs se distinguent par l'algorithme qu'ils utilisent pour éliminer du trafic. Voir dans la [RFC3290] plus d'informations sur les divers types d'abandonneurs. Noter que cette classe englobe à la fois les abandonneurs absolus et les abandonneurs algorithmiques de la [RFC3290].

DropperService est modélisé comme un ConditioningService de sorte qu'il peut être agrégé dans un QoSService (en utilisant l'association QoSConditioningSubService) pour indiquer que sa fonctionnalité sous-tend ce service de QS. Il participe à l'association NextService pour identifier l'objet ConditioningService suivant qui agit sur tout trafic restant qui n'est pas éliminé.

NextService a une sémantique spéciale pour les abandonneurs en plus de la sémantique générale "que se passe t-il ensuite ?" qui s'applique à tous les ConditioningServices. La ou les files d'attente à partir desquelles un abandonneur particulier élimine des paquets sont identifiées par la ou les chaînes suivantes d'associations NextService "à droite" de l'abandonneur jusqu'à ce qu'elles atteignent une file d'attente. La définition de cette classe est la suivante :

NOM : DropperService
DESCRIPTION : classe concrète de base qui décrit les caractéristiques communes des abandonneurs.
DÉRIVÉE DE : ConditioningService
TYPE : Concret
PROPRIÉTÉS : DropperType, OtherDropperType, DropFrom

Note : La propriété DropperType et la sous classe DropperService fournissent des informations similaires. La propriété DropperType est définie pour des interrogations, ainsi que pour des cas où une sous classe de DropperService n'est pas nécessaire pour modéliser un type particulier d'abandonneur. Par exemple, l'abandonneur absolu défini dans la [RFC3290] est modélisé comme instance de la classe DropperService avec son DropperType réglé à '4' ("Abandonneur absolu").

4.3.17.1 Propriété DropperType

C'est un entier non signé de 16 bits qui définit le type d'abandonneur. Les valeurs incluent :
1 – autre
2 – aléatoire
3 – Tête-queue
4 – Abandonneur absolu

Note : si la valeur de DropperType n'est pas une de ces quatre valeurs, cela DEVRAIT être interprété comme '1' (autre).

4.3.17.2 Propriété OtherDropperType

Cette propriété de chaîne est utilisée en conjonction avec la propriété DropperType. Quand la valeur de DropperType est '1' (c'est-à-dire, autre) le nom du type d'abandonneur apparaît alors dans cette propriété.

4.3.17.3 Propriété DropFrom

C'est un entier non signé de 16 bits qui indique le point dans la file d'attente associée à partir duquel les paquets devraient être éliminés. Les valeurs définies sont :
- o inconnu (0)
- o en tête (1)
- o en queue (2)

Note : si la valeur de DropFrom est '0' (inconnu), ou si ce n'est pas une des trois valeurs citées ici, les paquets PEUVENT alors être éliminés à partir de tout point de la file d'attente associée.

4.3.18 Classe HeadTailDropperService

C'est une classe concrète qui représente les informations de seuil d'un abandonneur de tête ou de queue. La propriété héritée DropFrom indique si une instance particulière de cette classe représente un abandonneur de tête ou un abandonneur de queue. Un abandonneur de tête examine toujours la file d'attente de laquelle il élimine des paquets, et cette file d'attente est toujours en rapport avec l'abandonneur comme service suivant dans l'association NextService. La définition de cette classe est la suivante :

NOM : HeadTailDropperService
DESCRIPTION : classe concrète utilisée pour décrire un abandonneur de tête ou de queue.
DÉRIVÉE DE : DropperService
TYPE : Concret
PROPRIÉTÉS : QueueThreshold

4.3.18.1 Propriété QueueThreshold

C'est un entier non signé de 32 bits qui indique la profondeur de file d'attente à laquelle le trafic sera éliminé. Pour un abandonneur de queue, tout le trafic nouveau arrivant est éliminé. Pour un abandonneur de tête, les paquets au début de la file d'attente sont éliminés pour faire de la place aux nouveaux paquets, qui sont ajoutés à la fin. La valeur est exprimée en octets.

4.3.19 Classe REDDropperService

C'est une classe concrète qui représente la capacité à éliminer du trafic réseau en utilisant un algorithme de détection précoce aléatoire (RED, *Random Early Detection*). Cet algorithme est décrit dans [RED]. L'objet d'un algorithme RED est d'éviter l'encombrement (par opposition à la gestion de l'encombrement). Au lieu d'attendre que les files d'attente se remplissent, puis d'éliminer un grand nombre de paquets, RED fonctionne en surveillant la profondeur moyenne de la file d'attente. Quand la profondeur de la file d'attente excède un seuil minimum, les paquets sont éliminés au hasard. Ces éliminations causent le ralentissement du taux de transmission de TCP pour les connexions qui rencontrent des éliminations de paquet. Les autres connexions TCP ne sont pas affectées par ces éliminations. Voir plus d'informations sur les abandonneurs dans la [RFC3290].

Un abandonneur RED élimine toujours les paquets dans une seule file d'attente, qui est en rapport avec l'abandonneur comme le service suivant dans l'association NextService. La ou les files d'attente examinées par l'algorithme d'abandon sont trouvées en suivant l'association CalculationServiceForDropper pour trouver le DropThresholdCalculationService de l'abandonneur, et ensuite en suivant la ou les associations CalculationBasedOnQueue pour trouver la ou les files d'attentes observées. La définition de cette classe est la suivante :

NOM : REDDropperService
DESCRIPTION : classe concrète utilisée pour décrire l'élimination en utilisant l'algorithme RED (ou une de ses variantes).
DÉRIVÉE DE : DropperService
TYPE : Concret
PROPRIÉTÉS : MinQueueThreshold, MaxQueueThreshold, ThresholdUnits, StartProbability, StopProbability

Note : dans la [RFC3289], il y a un seul diffServRandomDropTable qui représente la catégorie générale d'abandon aléatoire. (RED est un type d'abandon aléatoire, mais il y a aussi des types d'abandon aléatoire distincts de RED.) La classe REDDropperService correspond aux colonnes du tableau qui s'applique à l'algorithme RED en particulier.

4.3.19.1 Propriété MinQueueThreshold

C'est un entier non signé de 32 bits qui définit la profondeur minimum moyenne de file d'attente dans laquelle les paquets sont sujet à élimination. Les unités sont identifiées par la propriété ThresholdUnits. La pente de cette fonction de probabilité d'élimination est décrite par les propriétés Start/StopProbability.

4.3.19.2 Propriété MaxQueueThreshold

C'est un entier non signé de 32 bits qui définit la longueur moyenne maximum de file d'attente à laquelle les paquets sont sujets pour être toujours éliminés, sans considération de l'algorithme d'élimination et des probabilités utilisés. Les unités sont identifiées par la propriété ThresholdUnits.

4.3.19.3 Propriété ThresholdUnits

C'est un entier non signé de 16 bits qui identifie les unités des propriétés MinQueueThreshold et MaxQueueThreshold. Les valeurs définies sont :

- octets (1)
- paquets (2)

Note : si la valeur de ThresholdUnits n'est pas une de ces deux, elle DEVRAIT être interprétée comme valeur '1' (octets).

4.3.19.4 Propriété StartProbability

C'est un entier non signé de 32 bits ; en conjonction avec la propriété StopProbability, elle définit la pente de la fonction de probabilité. Cette fonction gouverne le taux auquel les paquets sont sujets à élimination, comme fonction de la longueur de la file d'attente.

Cette propriété exprime une probabilité d'abandon en abandons par millier de paquets. Par exemple, la valeur 100 indique une probabilité d'abandon de 100 par 1000 paquets, c'est-à-dire, 10 %. Les valeurs minimales et maximales sont 0 et 1000.

4.3.19.5 Propriété StopProbability

C'est un entier non signé de 32 bits ; en conjonction avec la propriété StartProbability elle définit la pente d'une fonction de probabilité d'abandon. Cette fonction gouverne le taux d'abandon des paquets comme fonction de la longueur de la file d'attente.

Cette propriété exprime une probabilité d'abandon en abandons par millier de paquets. Par exemple, la valeur 100 indique une probabilité d'abandon de 100 par 1000 paquets, c'est-à-dire, 10 %. Les valeurs minimales et maximales sont 0 et 1000.

4.3.20 Classe QueuingService

C'est une classe concrète qui représente la capacité à mettre en file d'attente le trafic réseau, et de spécifier les caractéristiques pour déterminer l'encombrement à long terme. Voir dans la [RFC3290] plus d'informations sur la fonction de mise en file d'attente.

QueuingService est modélisée comme un ConditioningService de sorte qu'elle peut être agrégée dans un QoSService (en utilisant l'association QoSConditioningSubService) pour indiquer que sa fonction sous-tend ce service de QS. La définition de cette classe est la suivante :

NOM : QueuingService
DESCRIPTION : classe concrète qui décrit la capacité à mettre en file d'attente le trafic réseau et de spécifier les caractéristiques pour déterminer l'encombrement à long terme.
DÉRIVÉE DE : ConditioningService
TYPE : Concret
PROPRIÉTÉS : CurrentQueueDepth, DepthUnits

4.3.20.1 Propriété CurrentQueueDepth

C'est un entier non signé de 32 bits, qui fonctionne comme une jauge (en lecture seule) représentant la profondeur actuelle de cette file d'attente. Cette valeur peut être importante pour diagnostiquer un comportement inattendu d'un DropThresholdCalculationService *(service de calcul de seuil d'abandon)*.

4.3.20.2 Propriété DepthUnits

C'est un entier non signé de 16 bits qui identifie les unités pour la propriété CurrentQueueDepth. Les valeurs définies sont :
o octets (1)
o paquets (2)
Note : si la valeur de DepthUnits n'est pas une de ces deux valeurs, elle DEVRAIT être interprétée comme '1' (octets).

4.3.21 Classe PacketSchedulingService

C'est une classe concrète qui représente un service de programmation, qui est un processus qui détermine quand un paquet en file d'attente devrait être retiré d'une file et envoyé sur une interface de sortie. Noter que les interfaces de sortie peuvent être des interfaces de réseau physiques ou des interfaces de composants internes aux systèmes, comme des entretoises ou des arrières plans. Dans l'un et l'autre cas, si plusieurs files d'attente sont impliquées, les programmateurs sont utilisés pour donner l'accès à l'interface.

Chaque instance d'un PacketSchedulingService décrit un programmateur du point de vue des files d'attente qu'il dessert. Voir plus d'informations sur les programmateurs dans la [RFC3290].

PacketSchedulingService est modélisée comme un ConditioningService de sorte qu'elle peut être agrégée dans un QoSService (en utilisant l'association QoSConditioningSubService) pour indiquer que sa fonction sous-tend ce service de QS. Elle participe à l'association NextService pour identifier l'objet ConditioningService suivant, s'il en est, qui agit sur le trafic après qu'il a été traité par le programmateur. La définition de cette classe est la suivante :

NOM : PacketSchedulingService
DESCRIPTION : classe concrète utilisée pour déterminer quand un paquet devrait être retiré d'une file d'attente et envoyé sur une interface de sortie.
DÉRIVÉE DE : ConditioningService
TYPE : Concret
PROPRIÉTÉS : SchedulerType, OtherSchedulerType

4.3.21.1 Propriété SchedulerType

Cette propriété est un entier non signé de 16 bits, et définit le type de programmateur. Les valeurs sont :
1 - autre

2 - FIFO
3 - Priorité
4 - Allocation
5 – Priorité limitée
6 - paquet Round Robin pondéré
Note : si la valeur de SchedulerType n'est pas une des six valeurs, elle DEVRAIT être interprétée comme la valeur '2' (FIFO).

4.3.21.2 Propriété OtherSchedulerType

Cette propriété de chaîne est utilisée en conjonction avec la propriété SchedulerType. Quand la valeur de SchedulerType est 1 (c'est-à-dire, autre) le type de programmateur est alors spécifié dans cette propriété.

4.3.22 Classe NonWorkConservingSchedulingService

Cette classe n'ajoute aucune propriété au delà de celles qu'elle hérite de sa super classe, PacketSchedulingService. Elle participe cependant à une association supplémentaire, FailNextScheduler. La définition de cette classe est la suivante :

NOM : NonWorkConservingSchedulingService
DESCRIPTION : classe concrète représentant un programmateur qui est capable de fonctionner d'une façon qui ne conserve pas le travail.
DÉRIVÉE DE : PacketSchedulingService
TYPE : Concret
PROPRIÉTÉS : (aucune)

4.3.23 Classe QoSService

C'est une classe concrète qui représente la capacité à conceptualiser un service de QS comme un ensemble de sous services coordonnés. Cela permet à l'administrateur du réseau de transposer les règles d'affaire dans le réseau, et au concepteur du réseau de gérer le réseau de façon telle qu'il puisse fournir des fonctions différentes pour différents flux de trafic.

Cette classe a deux objets principaux. D'abord, elle sert de classe de base commune pour définir les divers sous services nécessaires pour construire des services de QS de niveau supérieur. Ensuite, elle sert comme moyen de consolider les relations entre les différents types de services de QS et les différents types de ConditioningServices.

Par exemple, le service "or" peut être défini comme un QoSService qui agrège ensemble les deux services de QS. Chacun de ces services de QS pourrait être représenté par une instance de la classe DiffServService, une pour servir les paquets de très forte demande (représentés par une instance de DiffServService lui-même) et une pour le service fourni à la plupart des paquets, représenté par une instance de AFService, qui est une sous classe de DiffServService. L'instance de DiffServService de très forte demande va alors utiliser l'agrégation QoSConditioningSubService pour agréger ensemble les classeurs nécessaires pour indiquera quel trafic elle s'applique, et les mesureurs appropriés pour les limites de contrat, le marqueur pour marquer le PHB EF dans les paquets, et les services de conditionnement en rapport avec la mise en file d'attente. L'instance de AFService va aussi utiliser l'agrégation QoSConditioningSubService pour agréger ses classeurs et mesureurs, les différents marqueurs utilisés pour marquer les différents PHB AF dans les paquets, et les services de conditionnement en rapport avec la mise en file d'attente nécessaires pour délivrer le traitement de paquet.

QoSService est modélisé comme un type de Service, qui est utilisé comme point d'ancrage pour définir un ensemble de sous services qui mettent en œuvre les caractéristiques de conditionnement désirées pour les différents types de flux. Elle va diriger le type spécifique de services de conditionnement à utiliser afin de mettre en œuvre ce service. La définition de cette classe est la suivante :

NOM : QoSService
DESCRIPTION : classe concrète utilisée pour représenter un service ou ensemble de services de QS, comme défini par un administrateur de réseau.
DÉRIVÉE DE : Service
TYPE : Concret
PROPRIÉTÉS : (aucune)

4.3.24 Classe DiffServService

C'est une classe concrète qui représente l'utilisation de services DiffServ standard ou personnalisés pour mettre en œuvre un service de QS (de niveau supérieur). Noter qu'un objet DiffServService peut être juste un objet d'un ensemble d'objets

QoSSubServices coordonné qui ensemble mettent en œuvre un service de QS de niveau supérieur.

DiffServService est modélisée comme une sous classe de QoSService. Cela lui permet de se rapporter à un service de QS de niveau supérieur via QoSSubService, ainsi qu'à des objets ConditioningService spécifiques (par exemple, de mesure, d'abandon, de mise en file d'attente, et autres) via QoSConditioningSubService. La définition de cette classe est la suivante :

NOM : DiffServService
DESCRIPTION : classe concrète utilisée pour représenter un service DiffServ associé à un comportement par bond particulier.
DÉRIVÉE DE : QoSService
TYPE : Concret
PROPRIÉTÉS : PHBID

4.3.24.1 Propriété PHBID

Cette propriété est un entier non signé de 16 bits, qui identifie un comportement par bond particulier, ou une famille de comportements par bond. La valeur est ici un code d'identification de comportement par bond, comme défini dans la [RFC3140]. Noter que comme ils sont définis, ces codes d'identification utilisent les codets par défaut recommandés pour les PHB au titre de leur structure. Ces valeurs peuvent bien être différentes de la valeur réelle utilisée dans le marqueur, car la valeur marquée dépend du domaine. La capacité à indiquer le code d'identification de PHB associé à un service est utile pour lier le service de QS aux documents de référence, et pour la coordination et le fonctionnement inter domaines.

4.3.25 Classe AFService

C'est une classe concrète qui représente une spécialisation du concept général de transmission du trafic réseau, en ajoutant une sémantique spécifique qui caractérise le fonctionnement du service de transmission assurée (AF, *Assured Forwarding*) [RFC2597].

La [RFC2597] définit quatre classes AF différentes, pour représenter quatre différents traitements du trafic. Une quantité différente de ressources de transmission, comme l'espace de mémoire tampon, la bande passante, est allouée à chaque classe AF. Au sein de chaque classe AF, les paquets IP sont marqués avec une des trois valeurs possibles de préséance d'abandon. La préséance d'abandon d'un paquet détermine l'importance relative de ce paquet par rapport aux autres paquets au sein de la même classe AF, si de l'encombrement survient. Une interface encombrée va essayer d'éviter d'abandonner des paquets marqués avec une plus faible valeur de préséance d'abandon, en éliminant plutôt les paquets marqués d'une valeur de préséance d'abandon supérieure.

Noter que la [RFC2597] définit douze DSCP qui ensemble représentent le groupe AF de comportements par bond. Les mises en œuvre sont libres d'étendre cela (par exemple, en ajoutant plus de classes et/ou de préséances d'abandon).

La classe AFService est modélisée comme une spécialisation de DiffServService, qui à son tour est une spécialisation de QoSService. Cela lui permet d'être en relation avec les services de QS de niveau supérieur, ainsi qu'avec les sous services de conditionnement de niveau inférieur (par exemple, classification, mesure, abandon, mise en file d'attente, et autres). La définition de cette classe est la suivante :

NOM : AFService
DESCRIPTION : classe concrète pour décrire les caractéristiques communes des services différenciés qui sont utilisés pour affecter la transmission du trafic, en utilisant le groupe de PHB AF.
DÉRIVÉE DE : DiffServService
TYPE : Concret
PROPRIÉTÉS : ClassNumber, DropperNumber

4.3.25.1 Propriété ClassNumber

Cette propriété est un entier non signé de 8 bits qui indique le nombre de classes AF que cette mise en œuvre AF utilise. Parmi les instances agrégées qui utilisent l'agrégation QoSConditioningSubService avec une instance de AFService, on DEVRAIT trouver des marqueurs avec autant de valeurs distinctes que de ClassNumber de l'instance AFService.

4.3.25.2 Propriété DropperNumber

Cette propriété est un entier non signé de 8 bits qui indique le nombre de valeurs de préséance d'abandon qu'utilise cette mise en œuvre AF. Le nombre de valeurs de préséance d'abandon est le nombre PAR CLASSE AF. Les abandonneurs correspondants seront trouvés dans la collection de services de conditionnement agrégés avec l'agrégation

QoSConditioningSubService.

4.3.26 Classe FlowService

Cette classe représente un service qui prend en charge un microflux particulier. Le microflux est identifié par la propriété FlowID à valeur de chaîne. Dans certaines mises en œuvre, une instance de cette classe correspond à une entrée dans le tableau de flux de la mise en œuvre. La définition de cette classe est la suivante :

NOM : FlowService
DESCRIPTION : classe concrète représentant un microflux.
DÉRIVÉE DE : QoSService
TYPE : Concret
PROPRIÉTÉS : FlowID

4.3.26.1 Propriété FlowID

Cette propriété est une chaîne qui contient un identifiant d'un microflux.

4.3.27 Classe DropThresholdCalculationService

Cette classe représente une entité logique qui calcule une profondeur moyenne de file d'attente sur la base d'une pondération lissée et d'un intervalle de temps d'échantillonnage. Elle fait ce calcul au nom d'un abandonneur RED, pour permettre à l'abandonneur de prendre ses décisions d'abandon de paquets sur la base d'une profondeur de file d'attente moyenne lissée pour la file d'attente. La définition de cette classe est la suivante :

NOM : DropThresholdCalculationService
DESCRIPTION : classe concrète représentant une entité logique qui calcule une profondeur moyenne de file d'attente sur la base d'une pondération lissée et d'un intervalle de temps d'échantillonnage. Ces dernières sont des propriétés de ce service, décrivant comment il opère et ses paramètres nécessaires.
DÉRIVÉE DE : Service
TYPE : Concret
PROPRIÉTÉS : SmoothingWeight, TimeInterval

4.3.27.1 Propriété SmoothingWeight

Cette propriété est un entier non signé de 32 bits, entre 0 et 100 000 – spécifié en millièmes. Elle définit l'historique de pondération qui affecte le calcul de la profondeur moyenne de file d'attente courante. Le calcul de la profondeur courante de la file d'attente utilise l'inverse de cette valeur comme facteur, et un moins cet inverse comme facteur pour la moyenne de l'historique. Le calcul prend la forme :

moyenne = (ancienne moyenne * (1 - inverse de pondération lissée)) + (profondeur actuelle de file d'attente * inverse de pondération lissée)

Les mises en œuvre peuvent choisir de limiter l'ensemble de valeurs acceptables à un ensemble spécifié, comme une puissance de 2. Les valeurs minimale et maximale sont 0 et 100 000.

4.3.27.2 Propriété TimeInterval

Cette propriété est un entier non signé de 32 bits, définissant le nombre de nanosecondes entre chaque calcul de profondeur de file d'attente moyenne/lissée. Si cette propriété n'est pas spécifiée, le CalculationService peut déterminer un intervalle approprié.

4.3.28 Classe abstraite FilterEntryBase

FilterEntryBase est la classe de base abstraite à partir de laquelle toutes les classes d'entrées de filtre sont dérivées. Elle sert de point d'extrémité de l'agrégation EntriesInFilterList, qui groupe des entrées de filtre en listes de filtres. Ses propriétés incluent des propriétés de désignation CIM et une propriété booléenne IsNegated (pour "NE PAS" correspondre aux informations spécifiées dans une instance d'une de ses sous classes).

Parce que FilterEntryBase est d'applicabilité générale, elle est définie dans la [RFC3460]. Voir dans la [RFC3460] la définition de cette classe.

4.3.29 Classe IPHeaderFilter

Cette classe concrète rend possible la représentation d'un filtre entier d'en-tête IP dans un seul objet. Une propriété IpVersion identifie si les adresses IP dans une instance sont IPv4 ou IPv6. (Comme les adresses IP de source et de destination viennent du même en-tête de paquet, elles vont toujours être du même type.) Voir dans la [RFC3460] la définition de cette classe.

4.3.30 Classe 8021Filter

Cette classe concrète permet d'exprimer des adresses MAC 802.1 de source et de destination, ainsi que des champs Identifiant de protocole 802.1, Priorité, et Identifiant de VLAN dans un seul objet. Voir dans la [RFC3460] la définition de cette classe.

4.3.31 Classe PreambleFilter

C'est une classe concrète qui modélise le classement des paquets en utilisant les résultats du conditionnement de trafic mémorisés dans un préambule de paquet par un PreambleMarkerService. Voir au paragraphe 3.8.3 la discussion de comment, et pourquoi, les modèles QDDIM ont la capacité de mémoriser ces résultats dans un préambule de paquet. Une instance de PreambleFilter est utilisée pour choisir les paquets sur la base d'une chaîne en deux parties identifiant un résultat spécifique. La logique de cette confrontation est "au moins un". C'est-à-dire qu'un paquet avec plusieurs résultats dans son préambule satisfait à un filtre si au moins un de ses résultats correspond au filtre. La définition de cette classe est la suivante :

NOM : PreambleFilter
DESCRIPTION : classe concrète représentant les critères de choix des paquets sur la base des résultats du conditionnement du trafic précédent mémorisés dans un préambule de paquet.
DÉRIVÉE DE : FilterEntryBase
TYPE : Concret
PROPRIÉTÉS : FilterItemList[ ]

4.3.31.1 Propriété multi valeurs FilterItemList

Cette propriété est une liste ordonnée de chaînes, où chaque chaîne a le format "<type>,<valeur>". Voir au paragraphe 3.8.3 une liste des <types> définis dans QDDIM, et la nature de la <valeur> associée pour chacun de ces types.

Noter qu'il y a deux terminologies parallèles pour caractériser les résultats de mesures. La valeur d'énumération de "conforme(1)" est parfois décrite comme "dans le profil," et la valeur "nonConforme(3)" est parfois décrite comme "hors profil".

4.3.32 Classe FilterList

C'est une classe concrète qui agrège des instances de (sous classes de) FilterEntryBase via l'agrégation EntriesInFilterList. Il est possible d'agréger différents types de filtres dans une seule FilterList - par exemple, des filtres d'en-tête de paquet (représentés par la classe IPHeaderFilter) et des filtres de sécurité (représentés par des sous classes de FilterEntryBase définies par IPsec).

La propriété d'agrégation EntriesInFilterList.EntrySequence est toujours réglée à 0, pour indique que les entrées de filtre agrégées sont ajoutées ensemble par l'opérateur logique ET pour former un sélecteur pour une classe de trafic. Voir dans la [RFC3460] la définition de cette classe.

4.3.33. Classe abstraite ServiceAccessPoint

C'est une classe abstraite définie dans le modèle de cœur de CIM. C'est une sous classe de la classe LogicalElement, et c'est la classe de base pour tous les objets qui donnent accès à CIM_Services. Elle représente la gestion de l'utilisation ou de l'invocation d'un Service. Prière de se référer à [CIM] pour la définition complète de cette classe.

4.3.34 Classe ProtocolEndpoint

C'est une classe concrète dérivée de ServiceAccessPoint, qui décrit un point de communication à partir duquel les services du réseau ou la pile de protocole du système peuvent être atteints. Prière de se référer à [CIM] pour la définition complète de cette classe.

4.3.35 Classe abstraite Collection

C'est une classe abstraite définie dans le modèle de cœur de CIM. C'est la super classe pour toutes les classes qui représentent des groupements ou sacs, et qui ne portent pas d'état ou "state". (Ces derniers seraient plus correctement modélisés comme ManagedSystemElements.) Prière de se référer à [CIM] pour la définition complète de cette classe.

4.3.36 Classe abstraite CollectionOfMSEs

C'est une classe abstraite définie dans le modèle de cœur de CIM. C'est une sous classe de la super classe Collection, qui restreint le contenu de la Collection à ManagedSystemElements. Prière de se référer à [CIM] pour la définition complète de cette classe.

4.3.37 Classe BufferPool

C'est une classe concrète qui représente la collection des mémoires tampon utilisées par un QueuingService. (L'association QueueAllocation représente cet usage.) L'existence et la gestion des mémoires tampon individuelles pourront être modélisées dans un futur document. Au niveau d'abstraction actuel, la modélisation de l'existence de la BufferPool est nécessaire. À long terme cela ne sera pas suffisant.

Dans les mises en œuvre où il y a plusieurs tailles de mémoire tampon, une instance de BufferPool devrait être définie pour chaque ensemble de mémoires tampon de taille identique ou similaire. Ces instances de réservoirs de mémoires tampon peuvent alors être groupées en utilisant l'agrégation CollectedBuffersPool.

Noter que cette classe est dérivée de CollectionOfMSEs, et non de Forwarding ou ConditioningService. Une BufferPool est seulement une collection de mémorisations, et n'est PAS un Service. La définition de cette classe est la suivante :

NOM : BufferPool
DESCRIPTION : classe concrète représentant une collection de mémoires tampon.
DÉRIVÉE DE : CollectionOfMSEs
TYPE : Concret
PROPRIÉTÉS : Name, BufferSize, TotalBuffers, AvailableBuffers, SharedBuffers

4.3.37.1 Propriété Name

Cette propriété est une chaîne d'une longueur maximum de 256 caractères. C'est le nom commun ou l'étiquette par laquelle l'objet est connu.

4.3.37.2.Propriété BufferSize

Cette propriété est un entier non signé de 32 bits, identifiant le nombre approximatif d'octets dans chaque mémoire tampon dans le réservoir de mémoires tampon. Une mise en œuvre va normalement grouper les mémoires tampon d'environ la même taille, pour réduire le nombre de réservoirs de mémoires tampon à gérer. Ce modèle ne spécifie pas le degré dont les mémoires tampon dans le même réservoir diffèrent en taille.

4.3.37.3 Propriété TotalBuffers

Cette propriété est un entier non signé de 32 bits, qui rapporte le nombre total de mémoires tampon individuelles dans le réservoir.

4.3.37.4 Propriété AvailableBuffers

Cette propriété est un entier non signé de 32 bits, rapportant le nombre de mémoires tampon du réservoir qui ne sont pas actuellement allouées à une instance d'un QueuingService. Les mémoires tampon allouées à un QueuingService pourraient être soit utilisées (c'est-à-dire, qui contiennent actuellement des paquets de données) soit allouées à une file d'attente en attendant l'arrivée de nouveaux paquets de données.

4.3.37.5 Propriété SharedBuffers

Cette propriété est un entier non signé de 32 bits, rapportant le nombre de mémoires tampon dans le réservoir qui a été simultanément alloué à plusieurs instances de QueuingService.

4.3.38 Classe abstraite SchedulingElement

C'est une classe abstraite qui représente les informations de configuration qu'a un PacketSchedulingService pour un des éléments qu'il programme. L'élément programmé est soit un QueuingService, soit un autre PacketSchedulingService.

Parmi les sous classes de cette classe, certaines sont définies d'une façon telle que toutes leurs instances conservent le travail. D'autres sous classes, cependant, peuvent avoir des instances qui conservent ou non le travail. Dans cette classe, la propriété booléenne WorkConserving indique si une instance conserve ou non le travail. La gamme des valeurs pour WorkConserving se restreint à VRAI dans les sous classes qui par nature conservent le travail, car les instances de ces classes ne peuvent pas être autre chose que conservant le travail. La définition de cette classe est la suivante :

NOM : SchedulingElement
DESCRIPTION : classe abstraite représentant les informations de configuration qu'a un PacketSchedulingService pour un des éléments qu'il programme.
DÉRIVÉE DE :ManagedElement
TYPE : Abstrait
PROPRIÉTÉS : WorkConserving

4.3.38.1 Propriété WorkConserving

Cette propriétés booléenne indique si le PacketSchedulingService lié à cette instance par l'agrégation ElementInSchedulingService traite l'entrée liée à cette instance par l'association QueueToSchedule ou SchedulingServiceToSchedule d'une manière qui conserve le travail. Noter que cette propriété est modifiable en écriture, ce qui signifie qu'un administrateur peut changer le comportement du SchedulingElement – mais seulement pour les éléments qui peuvent opérer en mode de non conservation du travail.

4.3.39 Classe AllocationSchedulingElement

Cette classe est une sous classe de la classe abstraite SchedulingElement. Elle introduit cinq nouvelles propriétés pour prendre en charge la programmation fondée sur la bande passante. Comme c'est le cas avec toutes les sous classes de SchedulingElement, l'entrée associée à une instance de AllocationSchedulingElement est d'un des deux types suivants : soit une file d'attente, soit un autre programmateur. La définition de cette classe est la suivante :

NOM : AllocationSchedulingElement
DESCRIPTION : classe concrète contenant des paramètres pour contrôler la programmation fondée sur la bande passante.
DÉRIVÉE DE : SchedulingElement
TYPE : Concret
PROPRIÉTÉS : AllocationUnits, BandwidthAllocation, BurstAllocation, CanShare, WorkFlexible

4.3.39.1 Propriété AllocationUnits

Cette propriété est un entier non signé de 16 bits qui identifie les unités dans lesquelles sont exprimées les propriétés BandwidthAllocation et BurstAllocation. Les valeurs suivantes sont définies :

- o octets (1)
- o paquets (2)
- o cellules (3) -- taille fixe, par exemple, ATM

Note : si la valeur de AllocationUnits n'est pas une de ces trois valeurs, elle DEVRAIT être interprétée comme ayant la valeur '1' (octets).

4.3.39.2.Propriété BandwidthAllocation

Cette propriété est un entier non signé de 32 bits qui définit le nombre d'unités/secondes qui devrait être alloué à l'entrée associée. Les unités sont identifiées par la propriété AllocationUnits.

4.3.39.3 Propriété BurstAllocation

Cette propriété est un entier non signé de 32 bits qui spécifie la quantité de bande passante temporaire ou à court terme (en unités par seconde) qui peut être allouée à une entrée, au delà de la quantité de bande passante allouée au moyen de la propriété BandwidthAllocation. Si l'allocation maximum réelle de bande passante pour l'entrée devait être mesurée, ce serait la somme des propriétés BurstAllocation et BandwidthAllocation. Les unités sont identifiées par la propriété AllocationUnits.

4.3.39.4 Propriété CanShare

C'est une propriété booléenne qui, si elle est VRAIE, permet d'allouer la bande passante inutilisée provenant des entrées associées aux autres entrées desservies par le programmateur.

4.3.39.5 Propriété WorkFlexible

C'est une propriété booléenne qui, si elle est VRAIE, indique que le comportement du programmateur relatif à cette entrée peut être altéré en changeant la valeur de la propriété héritée WorkConserving.

4.3.40 Classe WRRSchedulingElement

Cette classe est une sous classe de la classe abstraite SchedulingElement, représentant une discipline de programmation à pondération comparative (WRR, *weighted round robin*). Elle introduit une nouvelle propriété WeightingFactor, pour donner une plus forte probabilité à certaines entrées d'être servies que d'autres. Elle introduit aussi une propriété Priority, pour servir à départager des entrées qui ont des facteurs de pondération égaux. Comme c'est le cas avec toutes les sous classes de SchedulingElement, l'entrée associée à une instance de WRRSchedulingElement est soit une file d'attente, soit un autre programmateur.

Parce que la programmation de ce type est toujours à conservation de travail, la propriété booléenne héritée WorkConserving est restreinte à la valeur VRAI dans cette classe. La définition de cette classe est la suivante :

NOM : WRRSchedulingElement
DESCRIPTION : cette classe spécialise la classe SchedulingElement pour lui ajouter une pondération par entrée. C'est utilisé par un programmateur de paquet à pondération comparative lorsque il traite ses entrées associées. Cela ajoute aussi une seconde propriété qui sert à départager plusieurs entrées qui ont la même pondération.
DÉRIVÉE DE : SchedulingElement
TYPE : Concret
PROPRIÉTÉS : WeightingFactor, Priority

4.3.40.1 Propriété WeightingFactor

Cette propriété est un entier non signé de 32 bits, qui définit le facteur de pondération qui offre à certaines entrées une plus forte probabilité d'être desservies qu'à d'autres. Cette propriété représente cette probabilité. Son minimum est 0, son maximum est 100 000, et ses unités sont des millièmes.

4.3.40.2 Propriété Priority

Cette propriété est un entier non signé de 16 bits, qui sert au départage, dans les cas où des entrées ont des pondérations égales. Un plus grande valeur représente une priorité plus élevée. Si cette propriété est spécifiée pour un des WRRSchedulingElements associés à un PacketSchedulingService, elle doit alors être spécifiée pour tous les WRRSchedulingElements pour ce PacketSchedulingService, et les valeurs de la propriété pour ces WRRSchedulingElements doivent être toutes différentes.

Bien que la condition puisse ne pas se produire dans certaines mises en œuvre de programmateur à pondération comparative, de nombreuses mises en œuvre exigent une priorité pour résoudre une condition d'égalité de poids. Dans les instances où ce comportement n'est pas nécessaire ou est indésirable, cette propriété peut rester non spécifiée.

4.3.41 Classe PrioritySchedulingElement

Cette classe est une sous classe de la classe abstraite SchedulingElement. Elle indique qu'un programmateur prend des paquets d'un ensemble d'entrées en utilisant la discipline de programmation à priorité. Comme c'est le cas avec toutes les sous classes de SchedulingElement, l'entrée associée à une instance de PrioritySchedulingElement est soit une file d'attente soit un autre programmateur. La propriété Priority dans PrioritySchedulingElement représente la priorité pour une entrée, par rapport aux priorités de toutes les autres entrées auxquelles le programmateur qui agrège ce PrioritySchedulingElement est associé. Les entrées auxquelles se rapporte le programmateur via d'autres disciplines de programmation ne figurent pas dans cette priorité.

Comme la programmation de ce type conserve toujours le travail, la propriété booléenne héritée WorkConserving se restreint à la valeur VRAI dans cette classe. La définition de cette classe est la suivante :

NOM : PrioritySchedulingElement
DESCRIPTION : classe concrète qui spécialise la classe SchedulingElement pour ajouter une propriété Priority. Cette propriété est utilisée par un SchedulingService qui fait une programmation de priorité pour un ensemble

d'entrées.
DÉRIVÉE DE : SchedulingElement
TYPE : Concret
PROPRIÉTÉS : Priority

4.3.41.1 Propriété Priority

Cette propriété est un entier non signé de 16 bits qui indique le niveau de priorité d'une entrée de programmateur par rapport aux autres entrées desservies par ce PacketSchedulingService. Plus la valeur est grande plus la priorité est élevée.

4.3.42 Classe BoundedPrioritySchedulingElement

Cette classe est une sous classe de la classe PrioritySchedulingElement, qui est elle-même dérivée de la classe abstraite SchedulingElement. Comme c'est le cas avec toutes les sous classes de SchedulingElement, l'entrée associée à une instance de BoundedPrioritySchedulingElement est soit du type file d'attente, soit un autre programmateur. BoundedPrioritySchedulingElement ajoute une limite supérieure (en kilobits par seconde) à la quantité de trafic qui peut être traité à partir d'une entrée. Ces données sont spécifiques de cette entrée. Elle est nécessaire lorsque une programmation à limite stricte de priorité est effectuée.

Cette classe hérite de sa super classe PrioritySchedulingElement la restriction de la propriété booléenne héritée WorkConserving à la valeur VRAI. La définition de cette classe est la suivante :

NOM : BoundedPrioritySchedulingElement
DESCRIPTION : cette classe concrète spécialise la classe PrioritySchedulingElement pour ajouter une propriété BandwidthBound. Cette propriété limite le débit auquel le trafic provenant de l'entrée associée peut être traité.
DÉRIVÉE DE : PrioritySchedulingElement
TYPE : Concret
PROPRIÉTÉS : BandwidthBound

4.3.42.1 Propriété BandwidthBound

Cette propriété est un entier non signé de 32 bits qui définit la limite supérieure de la quantité de trafic qui peut être traité à partir de l'entrée. Ce n'est pas une limite supérieure formatée, car des salves peuvent se produire. C'est une limite stricte, limitant l'impact de l'entrée. Les unités sont des kilobits par seconde.

4.4 Définitions d'associations

Cette section détaille les associations de chemin de données d'appareil de QS, incluant les agrégations, qui ont été montrées aux Figures 4 et 5. Ces associations sont définies comme des classes dans le modèle d'informations. Chacune de ces classes a deux propriétés qui se réfèrent aux instances des deux classes que relie l'association. Certaines des classes d'association ont aussi des propriétés supplémentaires.

4.4.1 Association abstraite Dependency

Cette association abstraite définit deux références d'objet (nommées Antecedent et Dependent) qui établissent une relation de dépendance générale entre différent objets gérés dans le modèle d'informations. La référence Antecedent identifie l'objet indépendant dans l'association, tandis que la référence Dependent identifie l'entité qui dépend de l'IS. La cardinalité de l'association est de plusieurs à plusieurs. L'association est définie dans le modèle de cœur de CIM. Prière de se référer à [CIM] pour la définition complète de cette classe.

4.4.2 Association ServiceSAPDependency

Cette association définit deux références d'objet qui établissent une relation de dépendance générale entre un objet Service et un objet ServiceAccessPoint. Cette relation indique que le Service référencé utilise le ServiceAccessPoint d'un AUTRE Service. Le Service est la référence Dependent, qui s'appuie sur le ServiceAccessPoint pour obtenir l'accès à un autre Service. La cardinalité de l'association est de plusieurs à plusieurs. L'association est définie dans le modèle de cœur de CIM. Prière de se référer à [CIM] pour la définition complète de cette classe.

4.4.3 Association IngressConditioningServiceOnEndpoint

Cette association est dérivée de l'association ServiceSAPDependency, et représente le lien, dans la direction d'entrée, entre un point d'extrémité de protocole et le premier ConditioningService qui traite les paquets reçus via ce point d'extrémité de protocole. Comme il ne peut y avoir qu'un seul "premier" ConditioningService pour un point d'extrémité de protocole, la cardinalité pour la référence d'objet Dependent est rétrécie de 0..n à 0..1. Comme d'un autre côté, un seul ConditioningService peut être le premier à traiter les paquets reçus via plusieurs points d'extrémité de protocole, la cardinalité de la référence d'objet Antecedent reste de 0..n. La définition de cette classe est la suivante :

NOM : IngressConditioningServiceOnEndpoint
DESCRIPTION : association qui établit une relation de dépendance entre un point d'extrémité de protocole et le premier service de conditionnement qui traite le trafic arrivant via ce point d'extrémité de protocole.
DÉRIVÉE DE : ServiceSAPDependency
ABSTRAITE : Faux
PROPRIÉTÉS : Antecedent[ref ProtocolEndpoint[0..n]], Dependent[ref ConditioningService[0..1]]

4.4.4 Association EgressConditioningServiceOnEndpoint

Cette association est dérivée de l'association ServiceSAPDependency, et représente le lien, dans la direction sortie, entre un point d'extrémité de protocole et le dernier ConditioningService qui traite ses paquets avant qu'ils quittent l'appareil réseau via ce point d'extrémité de protocole. (Ce "dernier" ConditioningService est ordinairement un programmateur, mais il ne l'est pas obligatoirement.) Comme il peut y avoir plusieurs "derniers" ConditioningServices pour un point d'extrémité de protocole dans le cas d'un programmateur de secours, la cardinalité pour la référence d'objet Dependent reste 0..n. Cependant comme un seul ConditioningService ne peut pas être le dernier à traiter les paquets pou plusieurs points d'extrémité de protocole, la cardinalité de la référence d'objet Antecedent est rétrécie de 0..n à 0..1. La définition de cette classe est la suivante :

NOM : EgressConditioningServiceOnEndpoint
DESCRIPTION : association qui établit une relation de dépendance entre un point d'extrémité de protocole et le dernier service de conditionnement qui traite le trafic à transmettre via ce point d'extrémité de protocole.
DÉRIVÉE DE : ServiceSAPDependency
ABSTRAITE : Faux
PROPRIÉTÉS : Antecedent[ref ProtocolEndpoint[0..1]], Dependent[ref ConditioningService[0..n]]

4.4.5 Association HeadTailDropQueueBinding

Cette association est une sous classe de Dependency, qui décrit l'association entre un abandonneur de tête ou de queue et une file d'attente qu'il surveille pour déterminer quand éliminer du trafic. La file d'attente référencée est celle dont la profondeur de file d'attente est comparée au seuil de l'abandonneur. La cardinalité est 1..n sur le côté file d'attente, car un abandonneur de tête/queue doit surveiller au moins une file d'attente. Pour les classes HeadTailDropper et HeadTailDropQueueBinding, la règle de combinaison des entrées de plusieurs files d'attente est une simple addition : si la somme des longueurs des files d'attente surveillées excède la valeur de QueueThreshold de l'abandonneur, les paquets sont alors éliminés. Cette règle de combinaison des entrées peut cependant être outrepassées par une règle différente dans des sous classes d'une de ces classes ou des deux. La définition de cette classe est la suivante :

NOM : HeadTailDropQueueBinding
DESCRIPTION : association générique utilisée pour établir une relation de dépendance entre un abandonneur de tête ou de queue et une file d'attente qu'il surveille.
DÉRIVÉE DE : Dependency
ABSTRAITE : Faux
PROPRIÉTÉS : Antecedent[ref QueuingService[1..n]], Dependent[ref HeadTailDropperService [0..n]]

4.4.6 Association CalculationBasedOnQueue

Cette association est une sous classe de Dependency, qui définit deux références d'objet qui établissent une relation de dépendance entre un QueuingService et une instance de la classe DropThresholdCalculationService. La profondeur courante de la file d'attente est utilisée par le service de calcul pour calculer une profondeur moyenne de file d'attente. La définition de cette classe est la suivante :

NOM : CalculationBasedOnQueue
DESCRIPTION : association générique utilisée pour établir une relation de dépendance entre un objet QueuingService et un objet DropThresholdCalculationService.
DÉRIVÉE DE : ServiceServiceDependency

ABSTRAITE : Faux
PROPRIÉTÉS : Antecedent[ref QueuingService[1..1]], Dependent[ref DropThresholdCalculationService [0..n]]

4.4.6.1 Référence Antecedent

Cette propriété est héritée de l'association Dependency, qui est outrepassée pour servir de référence d'objet à un objet QueuingService (au lieu du ManagedElement plus général). Cette référence identifie la file d'attente que le DropThresholdCalculationService va utiliser dans son calcul de profondeur moyenne de file d'attente.

4.4.6.2 Référence Dependent

Cette propriété est héritée de l'association Dependency, qui est outrepassée pour servir de référence d'objet à un objet DropThresholdCalculationService (au lieu du ManagedElement plus général). Cette référence identifie un DropThresholdCalculationService qui utilise la profondeur courante de la file d'attente référencée comme une des entrées de son calcul de profondeur moyenne de file d'attente.

4.4.7 Association ProvidesServiceToElement

Cette association définit deux références d'objet qui établissent une relation de dépendance dans laquelle un ManagedSystemElement dépend de la fonctionnalité d'un ou plusieurs Services. La cardinalité de l'association est de plusieurs à plusieurs. L'association est définie dans le modèle de cœur de CIM. Prière de se référer à [CIM] pour la définition complète de cette classe.

4.4.8 Association ServiceServiceDependency

Cette association définit deux références d'objet qui établissent une relation de dépendance entre deux objets Service. Le type particulier de dépendance est représenté par la propriété TypeOfDependency ; des exemples typiques incluent que la présence d'un Service est requise ou qu'il est exigé qu'il soit terminé pour que l'autre Service fonctionne.

Cette association est très similaire à la relation ServiceSAPDependency. Pour cette dernière, le Service dépend d'un AccessPoint pour avoir un autre Service. Dans cette relation, elle identifie directement sa dépendance au Service. Les deux relations ne devraient pas être instanciées, car leurs informations sont redondantes. La cardinalité de l'association est de plusieurs à plusieurs. L'association est définie dans le modèle de cœur de CIM. Prière de se référer à [CIM] pour la définition complète de cette classe.

4.4.9 Association CalculationServiceForDropper

Cette association est une sous classe de ServiceServiceDependency, qui définit deux références d'objet qui représentent la dépendance d'un REDDropperService à un DropThresholdCalculationService – en calculant une profondeur moyenne de file d'attente fondée sur les profondeurs observées sur une ou plusieurs files d'attente. La définition de cette classe est la suivante :

NOM : CalculationServiceForDropper
DESCRIPTION : association générique utilisée pour établir une relation de dépendance entre un service de calcul et un REDDropperService pour lequel il effectue des calculs de profondeur moyenne de file d'attente.
DÉRIVÉE DE : ServiceServiceDependency
ABSTRAITE : Faux
PROPRIÉTÉS : Antecedent[ref DropThresholdCalculationService[1..n]], Dependent[ref REDDropperService[0..n]]

4.4.9.1 Référence Antecedent

Cette propriété est héritée de l'association ServiceServiceDependency, et est outrepassée pour servir de référence d'objet à un objet DropThresholdCalculationService (à la place de l'objet Service plus général). La cardinalité de la référence d'objet est 1..n, indiquant qu'un abandonneur RED peut être servi par un ou plusieurs services de calcul.

4.4.9.2 Référence Dependent

Cette propriété est héritée de l'association ServiceServiceDependency, et est outrepassée pour servir de référence d'objet à un objet REDDropperService (à la place de l'objet Service plus général). Cette référence identifie un abandonneur RED servi par un DropThresholdCalculationService.

4.4.10 Association QueueAllocation

Cette association est une sous classe de Dependency, qui définit deux références d'objet qui établissent une relation de dépendance entre un QueuingService et un BufferPool qui fournit un espace de mémorisation pour les paquets dans la file d'attente. La définition de cette classe est la suivante :

NOM : QueueAllocation
DESCRIPTION : association générique utilisée pour établir une relation de dépendance entre un objet QueuingService et un objet BufferPool.
DÉRIVÉE DE : Dependency
ABSTRAITE : Faux
PROPRIÉTÉS : Antecedent[ref BufferPool[0..n]], Dependent[ref QueuingService[0..n]], AllocationPercentage

4.4.10.1 Référence Antecedent

Cette propriété est héritée de l'association Dependency, et est outrepassée pour servir de référence d'objet à un objet BufferPool. Cette référence identifie le BufferPool dans lequel les paquets de la file d'attente du QueuingService sont mémorisés.

4.4.10.2 Référence Dependent

Cette propriété est héritée de l'association Dependency, et est outrepassée pour servir de référence d'objet à un objet QueuingService. Cette référence identifie le QueuingService dont les paquets sont mémorisés dans les mémoires tampon du BufferPool.

4.4.10.3 Propriété AllocationPercentage

Cette propriété est un entier non signé de 8 bits avec une valeur minimum de zéro et une valeur maximum de 100. Elle définit le pourcentage du BufferPool qui devrait être alloué au QueuingService référencé. Si on désire une taille absolue, cela sera fait en définissant des BufferPool individuels de la taille spécifiée, avec QueueAllocation.AllocationPercentages réglé à 100.

4.4.11 Association ClassifierElementUsesFilterList

Cette association est une sous classe de l'association Dependency. Elle se rapporte à un ou plusieurs ClassifierElements avec une FilterList qui représente les critères de choix des paquets pour chaque ClassifierElements à traiter.

Dans le modèle QDDIM, un classeur est toujours modélisé comme un ClassifierService qui agrège un ensemble de ClassifierElements. Lorsque le ClassifierElements utilise l'association NextServiceAfterClassifierElement pour se lier avec un autre ClassifierService (pour construire un classeur hiérarchique) l'association ClassifierElementUsesFilterList ne doit pas être spécifiée. La définition de cette classe est la suivante :

NOM : ClassifierElementUsesFilterList
DESCRIPTION : association qui met en rapport un ClassifierElement avec la FilterList représentant les critères de choix des paquets à traiter par ce ClassifierElement.
DÉRIVÉE DE : Dependency
ABSTRAITE : Faux
PROPRIÉTÉS : Antecedent[ref FilterList [0..1]], Dependent[ref ClassifierElement [0..n]]

4.4.11.1 Référence Antecedent

Cette propriété est héritée de l'association Dependency, et est outrepassée pour servir de référence d'objet à un objet FilterList, au lieu de l'objet plus général ManagedElement. Aussi, sa cardinalité est restreinte à 0 et 1, indiquant qu'un ClassifierElement utilise soit une FilterList pour choisir les paquets pour elle, soit pas de FilterList lorsque le ClassifierElement utilise l'association NextServiceAfterClassifierElement pour la lier à un autre ClassifierService pour former un classeur hiérarchique.

4.4.11.2 Référence Dependent

Cette propriété est héritée de l'association Dependency, et est outrepassée pour servir de référence d'objet à un objet ClassifierElement, au lieu de l'objet plus général ManagedElement. Cette référence identifie un ClassifierElement qui dépend de l'objet FilterList associé pour représenter ses critères de choix de paquet.

4.4.12 Association AFRelatedServices

Cette association définit deux références d'objet qui établissent une relation de dépendance entre deux objets AFService. Cette dépendance est la préséance des services AF individuels en relation avec l'abandon au sein d'une classe de transmission de paquet IP AF. La définition de cette classe est la suivante :

NOM : AFRelatedServices
DESCRIPTION : association utilisée pour établir une relation de dépendance entre deux objets AFService.
DÉRIVÉE DE : Rien
ABSTRAITE : Faux
PROPRIÉTÉS : AFLowerDropPrecedence[ref AFService[0..1]], AFHigherDropPrecedence[ref AFService[0..n]]

4.4.12.1 Référence AFLowerDropPrecedence

Cette propriété sert de référence d'objet à un objet AFService qui a la plus faible probabilité d'éliminer des paquets.

4.4.12.2 Référence AFHigherDropPrecedence

Cette propriété sert de référence d'objet à un objet AFService qui a la plus forte probabilité d'éliminer des paquets.

4.4.13 Association NextService

Cette association définit deux références d'objet qui établissent une relation de prédécesseur-successeur entre deux objets ConditioningService. Cette association est utilisée pour indiquer la séquence des ConditioningServices requise pour traiter un type particulier de trafic.

Les instances de cette dépendance décrivent les diverses relations entre différents ConditioningServices (comme des classeurs, des mesureurs, abandonneurs, etc.) qui sont utilisés collectivement pour conditionner le trafic. Des relations univoques et plus compliquées de tri interne et/ou externe peuvent être décrites. Les ConditioningServices peuvent se nourrir les uns les autres directement, ou il peuvent être transposés en plusieurs "prochains" Services sur la base des caractéristiques du paquet. La définition de cette classe est la suivante :

NOM : NextService
DESCRIPTION : association utilisée pour établir une relation de prédécesseur-successeur entre deux objets ConditioningService.
DÉRIVÉE DE : rien
ABSTRAITE : Faux
PROPRIÉTÉS : PrecedingService[ref ConditioningService[0..n]], FollowingService[ref ConditioningService[0..n]]

4.4.13.1 Référence PrecedingService

Cette propriété sert de référence d'objet à un objet ConditioningService qui survient plus tôt dans la séquence de traitement pour un certain type de trafic.

4.4.13.2 Référence FollowingService

Cette propriété sert de référence d'objet à un objet ConditioningService qui survient ultérieurement dans la séquence de traitement pour un certain type de trafic, immédiatement après le ConditioningService identifié par la référence d'objet PrecedingService.

4.4.14 Association NextServiceAfterClassifierElement

Cette association précise la définition de sa super classe, l'association NextService, de deux façons :
- elle restreint la référence d'objet PrecedingService à la classe ClassifierElement,
- elle restreint la cardinalité de la référence d'objet FollowingService exactement à 1.

La définition de cette classe est la suivante :

NOM : NextServiceAfterClassifierElement
DESCRIPTION : association utilisée pour établir une relation de prédécesseur-successeur entre un seul ClassifierElement au sein d'un classeur et le prochain objet ConditioningService qui est responsable de la suite du traitement du trafic choisi par ce ClassifierElement.
DÉRIVÉE DE : NextService

ABSTRAITE : Faux
PROPRIÉTÉS : PrecedingService [ref ClassifierElement[0..n]], FollowingService[ref ConditioningService[1..1]

4.4.14.1 Référence PrecedingService

Cette propriété est héritée de l'association NextService. Elle est outrepassée dans cette sous classe pour restreindre la référence d'objet à un ClassifierElement, par opposition au ConditioningService plus général défini dans la super classe NextService. Cette propriété sert de référence d'objet à un ClassifierElement, qui est un composant d'un seul ClassifierService. Les paquets choisis par ce ClassifierElement sont toujours passés au ConditioningService identifié par la référence d'objet FollowingService.

4.4.14.2 Référence FollowingService

Cette propriété est héritée de l'association NextService. Elle est outrepassée dans cette sous classe pour restreindre la cardinalité de la référence à exactement 1. Cela reflète l'exigence que le comportement d'un classeur DiffServ soit déterministe : les paquets choisis par un certain ClassifierElement dans un certain ClassifierService doivent toujours aller dans le seul prochain ConditioningService.

4.4.15 Association NextScheduler

Cette association est une sous classe de NextService, et définit deux références d'objet qui établissent une relation de prédécesseur-successeur entre les PacketSchedulingServices. Dans une configuration de mise en file d'attente hiérarchique où un second programmateur traite le résultat d'un premier programmateur comme une seule entrée agrégée, les deux programmateurs sont en relation via l'association NextScheduler. La définition de cette classe est la suivante :

NOM : NextScheduler
DESCRIPTION : association utilisée pour établir une relation de prédécesseur-successeur entre les objets PacketSchedulingService pour une programmation hiérarchique simple.
DÉRIVÉE DE : NextService
ABSTRAITE : Faux
PROPRIÉTÉS : PrecedingService[ref PacketSchedulingService[0..n]], FollowingService[ref PacketSchedulingService[0..1]]

4.4.15.1 Référence PrecedingService

Cette propriété est héritée de l'association NextService, et est outrepassée pour servir de référence d'objet à un objet PacketSchedulingService (au lieu de l'objet plus général ConditioningService). Cette référence identifie un programmateur dont le résultat est traité comme une seule entrée agrégée par le programmateur identifié par la référence FollowingService. La cardinalité [0..n] indique qu'un seul programmateur FollowingService peut mettre ensemble les résultats agrégés de plusieurs programmateurs antérieurs.

4.4.15.2 Référence FollowingService

Cette propriété est héritée de l'association NextService, et est outrepassée pour servir de référence d'objet à un objet PacketSchedulingService (au lieu de l'objet plus général ConditioningService). Cette référence identifie un programmateur qui inclut parmi ses entrées les résultats agrégés d'un ou plusieurs programmateurs PrecedingService.

4.4.16 Association FailNextScheduler

Cette association est une sous classe de l'association NextScheduler. FailNextScheduler représente la relation entre deux programmateurs lorsque le premier programmateur renonce à une opportunité de programmation (se comportant par là d'une manière non conservatrice du travail) et rend la bande passante résultante disponible à l'utilisation du second programmateur. Voir aux paragraphes 3.11.3 et 3.11.4 des exemples d'utilisation de ces associations. La définition de cette classe est la suivante :

NOM : FailNextScheduler
DESCRIPTION : cette association spécialise l'association NextScheduler. Elle établit une relation entre un programmateur qui ne conserve pas le travail et un second programmateur auquel il rend disponible la bande passante qu'il choisit de ne pas utiliser.
DÉRIVÉE DE : NextScheduler
ABSTRAITE : Faux
PROPRIÉTÉS : PrecedingService[ref NonWorkConservingSchedulingService[0..n]]

4.4.16.1 Référence PrecedingService

Cette propriété est héritée de l'association NextSchedulerassociation, et est outrepassée pour servir de référence d'objet à un objet NonWorkConservingSchedulingService (au lieu de l'objet plus général PacketSchedulingService). Cette référence identifie un programmateur qui ne conserve pas le travail dont l'excès de bande passante est mis à la disposition du programmateur identifié par la référence FollowingService. La cardinalité [0..n] indique qu'un seul programmateur FollowingService peut avoir l'opportunité d'utiliser la bande passante inutilisée de plusieurs programmateurs antérieurs qui ne conservent pas le travail.

4.4.17 Association NextServiceAfterMeter

Cette association décrit une relation de prédécesseur-successeur entre un MeterService et un ou plusieurs objets ConditioningServicequi traitent le trafic provenant du mesureur. Par exemple, pour les appareils qui mettent en œuvre le marquage de préambule, la référence FollowingService (après le mesureur) est un PreambleMarkerService, pour enregistrer le résultat de la mesure dans le préambule.

On pourrait s'attendre à ce que l'association NextServiceAfterMeter soit une sous classe de NextService. Cependant, les mesureurs sont des éléments de déploiement 1:n, et requièrent un mécanisme pour distinguer les différents résultats du mesureur. Donc, cette association définit un nouvelle propriété clé, MeterResult, qui est utilisée pour enregistrer le résultat et identifier la sortie par laquelle ce trafic quitte le mesureur. À cause de cette clé supplémentaire, NextServiceAfterMeter ne peut pas être une sous classe de NextService. La définition de cette classe est la suivante :

NOM : NextServiceAfterMeter
DESCRIPTION : association utilisée pour établir une relation de prédécesseur-successeur entre un certain résultat d'un MeterService et le prochain objet ConditioningService qui est charge de la suite du traitement du trafic.
DÉRIVÉE DE : rien
ABSTRAITE : Faux
PROPRIÉTÉS : PrecedingService[ref MeterService[0..n]], FollowingService[ref ConditioningService[0..n]], MeterResult

4.4.17.1 Référence PrecedingService

C'est le MeterService précédant, 'plus tôt' dans la séquence de traitements d'un paquet. Comme les mesureurs sont des appareils de diffusion de 1:n, cette relation associe un résultat particulier d'un MeterService (identifié par la propriété MeterResult) au prochain ConditioningService qui est utilisé pour la suite du traitement du trafic.

4.4.17.2 Référence FollowingService

C'est le 'prochain' ConditioningService ou le suivant.

4.4.17.3 Propriété MeterResult

Cette propriété est un entier non signé de 16 bits, et représente des informations qui décrivent le résultat de la mesure. Le trafic est distingué comme conformé, non conforme, ou partiellement conforme. Des mesures plus compliquées peuvent être construites en étendant l'énumération ou en mettant les mesureurs en cascade. Les valeurs admises sont : "inconnu" (0), "Conforme" (1), "PartiellementConforme" (2), "NonConforme" (3).

4.4.18 Association QueueToSchedule

C'est une association de niveau supérieur, qui représente la relation entre une file d'attente (QueuingService) et un SchedulingElement. Le SchedulingElement, à son tour, représente les informations dans un service de programmation de paquets qui est spécifique de cette file d'attente, comme une priorité relative ou la bande passante allouée.

Elle ne peut pas être exprimée formellement avec les cardinalités d'association, mais il y a une contrainte supplémentaire sur la participation à l'association. Une instance particulière de (une sous classe de) SchedulingElement participe toujours soit à exactement une instance de cette association, soit à exactement une instance de l'association SchedulingServiceToSchedule. La définition de cette classe est la suivante :

NOM : QueueToSchedule
DESCRIPTION : Cette association met en relation une file d'attente avec le SchedulingElement qui contient les informations spécifiques de la file d'attente.

DÉRIVÉE DE : rien
ABSTRAITE : Faux
PROPRIÉTÉS : Queue[ref QueuingService[0..1]], SchedElement[ref SchedulingElement[0..n]]

4.4.18.1 Référence Queue

Cette propriété sert de référence d'objet pour un objet QueuingService. Un objet QueuingService peut être associé à 0, un ou plusieurs objets SchedulingElement.

4.4.18.2 Référence SchedElement

Cette propriété sert de référence d'objet pour un objet SchedulingElement. Un SchedulingElement est toujours associé à exactement un QueuingService ou à exactement un programmateur amont (PacketSchedulingService).

4.4.19 Association SchedulingServiceToSchedule

C'est une association de niveau supérieur, qui représente les relations entre un programmateur (PacketSchedulingService) et un SchedulingElement, dans une configuration qui implique des programmateurs en cascade. Le SchedulingElement, à son tour, représente les informations dans un service de programmation de paquets suivant qui est spécifique de ce programmateur, comme une priorité relative ou la bande passante allouée.

Elle ne peut pas être exprimée formellement par des cardinalités d'association, mais il y a une contrainte supplémentaire à la participation à cette association. Une instance particulière de (une sous classe de) SchedulingElement participe toujours soit à exactement une instance de cette association, soit à exactement une instance de l'association QueueToSchedule. La définition de cette classe est la suivante :

NOM : SchedulingServiceToSchedule
DESCRIPTION : Cette association met en rapport un programmateur avec le SchedulingElement dans un programmateur suivant qui contient des informations spécifiques de ce programmateur.
DÉRIVÉE DE : rien
ABSTRAITE : Faux
PROPRIÉTÉS : SchedService[ref PacketSchedulingService[0..1]], SchedElement[ref SchedulingElement[0..n]]

4.4.19.1 Référence SchedService

Cette propriété sert de référence d'objet pour un objet PacketSchedulingService. Un objet PacketSchedulingService peut être associé à 0, un ou plusieurs objets SchedulingElement.

4.4.19.2 Référence SchedElement

Cette propriété sert de référence d'objet pour un objet SchedulingElement. Un SchedulingElement est toujours associé soit à exactement un QueuingService, soit à exactement un programmateur amont (PacketSchedulingService).

4.4.20 Agrégation MemberOfCollection

Cette agrégation est une relation générique utilisée pour modéliser l'agrégation d'un ensemble de ManagedElements dans un objet Collection généralisé. La cardinalité de l'agrégation est de plusieurs à plusieurs. MemberOfCollection est définie dans le modèle de cœur de CIM. Prière de se référer à [CIM] pour la définition complète de cette classe.

4.4.21 Agrégation CollectedBufferPool

Cette agrégation modélise la capacité à traiter un ensemble de mémoires tampon comme un réservoir, ou collection, qui peut à son tour être contenu dans un réservoir de mémoires tampon de "niveau supérieur". Cette classe outrepasse l'agrégation plus générique MemberOfCollection pour restreindre les deux objets agrégés et la partie composant à être des instances de la seule classe BufferPool. La définition de classe pour l'agrégation est la suivante :

NOM : CollectedBufferPool
DESCRIPTION : association générique utilisée pour agréger un ensemble de mémoires tampon en rapports dans un réservoir de mémoires tampon de niveau supérieur.
DÉRIVÉE DE : MemberOfCollection

ABSTRAITE : Faux
PROPRIÉTÉS : Collection[ref BufferPool[0..1]], Member[ref BufferPool[0..n]]

4.4.21.1 Référence Collection

Cette propriété représente l'objet parent, ou agrégation, dans la relation. C'est un objet BufferPool.

4.4.21.2 Référence Member

Cette propriété représente l'enfant, ou le réservoir de niveau inférieur, dans la relation. C'est un élément de l'ensemble de BufferPools qui constitue le réservoir de niveau supérieur.

4.4.22 Agrégation abstraite Component

Cette agrégation abstraite est une relation générique utilisée pour établir une relation "d'appartenance" entre des objets gérés (nommés GroupComponent et PartComponent). La cardinalité de l'association est de plusieurs à plusieurs. L'association est définie dans le modèle de cœur de CIM. Prière de se référer à [CIM] pour la définition complète de cette classe.

4.4.23 Agrégation ServiceComponent

Cette agrégation est utilisée pour modéliser un ensemble de services subordonnés qui sont agrégés ensemble pour former un service de niveau supérieur. Cette agrégation est dérivée de la super classe plus générique Component pour restreindre les types d'objets qui peuvent participer à cette relation. La cardinalité de l'association est de plusieurs à plusieurs. L'association est définie dans le modèle de cœur de CIM. Prière de se référer à [CIM] pour la définition complète de cette classe.

4.4.24 Agrégation QoSSubService

Cette agrégation représente un ensemble de d'objets QoSService subordonnés (c'est-à-dire, un ensemble d'instances de sous classes de la classe QoSService) qui sont agrégées ensemble pour former un QoSService de niveau supérieur. Un QoSService est un type de Service spécifique qui conceptualise les fonctions de QS comme un ensemble de sous services coordonnés.

Cette agrégation est dérivée de la super classe plus générique ServiceComponent pour restreindre les types d'objets qui peuvent participer à cette relation aux objets QoSService, à la place d'un objet plus générique Service. Elle restreint aussi la cardinalité de l'agrégat à 0 ou 1 (au lieu du plus générique 0 ou plus). La définition de classe pour l'agrégation est la suivante :

NOM : QoSSubService
DESCRIPTION : association générique utilisée pour établir une relation "d'appartenance" entre un objet QoSService de niveau supérieur et l'ensemble des QoSServices de niveau inférieur sont agrégés pour le créer/former.
DÉRIVÉE DE : ServiceComponent
ABSTRAITE : Faux
PROPRIÉTÉS : GroupComponent[ref QoSService[0..1]], PartComponent[ref QoSService[0..n]]

4.4.24.1 Référence GroupComponent

Cette propriété est outrepassée dans cette agrégation pour représenter une référence d'objet à un objet QoSService (au lieu de l'objet plus générique Service défini dans sa super classe). Cet objet représente l'objet parent, ou agrégat, dans la relation.

4.4.24.2 Référence PartComponent

Cette propriété est outrepassée dans cette agrégation pour représenter une référence d'objet à un objet QoSService (au lieu de l'objet plus générique Service défini dans sa super classe). Cet objet représente l'objet enfant, ou "composant", dans la relation.

4.4.25 Agrégation QoSConditioningSubService

Cette agrégation identifie l'ensemble des services de conditionnement qui ensemble conditionnent le trafic pour un service de QS particulier. Cette agrégation est dérivée de la super classe plus générique ServiceComponent ; elle restreint les types d'objets qui peuvent y participer aux objets ConditioningService et QoSService, au lieu des objets plus génériques Service. La définition de classe pour l'agrégation est la suivante :

NOM : QoSConditioningSubService

DESCRIPTION : agrégation générique utilisée pour établir une relation "d'appartenance" entre un ensemble d'objets ConditioningService et le ou les objets QoSService particuliers pour lesquels ils fournissent le conditionnement de trafic.
DÉRIVÉE DE : ServiceComponent
ABSTRAITE : Faux
PROPRIÉTÉS : GroupComponent[ref QoSService[0..n]], PartComponent[ref ConditioningService[0..n]]

4.4.25.1 Référence GroupComponent

Cette propriété est outrepassée dans cette agrégation pour représenter une référence d'objet à un objet QoSService (au lieu de l'objet plus générique Service défini dans sa super classe). La cardinalité de la référence reste 0..n, pour indiquer qu'un certain ConditioningService peut fournir le conditionnement de trafic pour 0, 1, ou plus de un objet QoSService. Cet objet représente l'objet parent, ou agrégat, dans l'association. Dans ce cas, cet objet représente le QoSService qui agrège un ou plusieurs objets ConditioningService pour mettre en œuvre le conditionnement de trafic approprié pour son trafic.

4.4.25.2 Référence PartComponent

Cette propriété est outrepassée dans cette agrégation pour représenter une référence d'objet à un objet ConditioningService (au lieu de l'objet plus générique Service défini dans sa super classe). Cet objet représente l'objet enfant, ou "composant", dans la relation. Dans ce cas, cet objet représente un ou plusieurs objets ConditioningService qui ensemble indiquent comment le trafic est conditionné pour un QoSService spécifique.

4.4.26 Agrégation ClassifierElementInClassifierService

Cette agrégation représente la relation entre un classeur et les éléments de classeur qui fournissent la fonction de ventilation pour le classeur. Un classeur agrège normalement plusieurs éléments de classeur. Cependant, un élément de classeur n'est agrégé que par un seul classeur. Voir dans les [RFC3290] et [RFC3289] plus d'informations sur les classeurs et les éléments de classeur. La définition de classe pour l'agrégation est la suivante :

NOM : ClassifierElementInClassifierService
DESCRIPTION : agrégation qui représente la relation entre un classeur et ses éléments de classeur.
DÉRIVÉE DE : ServiceComponent
ABSTRAITE : Faux
PROPRIÉTÉS : GroupComponent[ref ClassifierService[1..1]], PartComponent[ref ClassifierElement[0..n], ClassifierOrder

4.4.26.1 Référence GroupComponent

Cette propriété est outrepassée dans cette agrégation pour représenter une référence d'objet à un objet ClassifierService (au lieu de l'objet plus générique Service défini dans sa super classe). Elle restreint aussi la cardinalité de l'agrégat à 1..1 (au lieu du plus générique 0 ou plus) représentant le fait qu'un ClassifierElement existe toujours dans le contexte d'exactement un ClassifierService.

4.4.26.2 Référence PartComponent

Cette propriété est outrepassée dans cette agrégation pour représenter une référence d'objet à un objet ClassifierElement (au lieu de l'objet plus générique Service défini dans sa super classe). Cet objet représente un seul sélecteur de trafic pour le classeur. Un ClassifierElement a généralement une association à une FilterList qui donne les critères de choix des paquets provenant du flux de trafic qui entre dans le classeur, et à un ConditioningService auquel les paquets choisis selon ces critères sont ensuite transmis.

4.4.26.3 Propriété ClassifierOrder

Comme les filtres pour un classeur peuvent se chevaucher, il est nécessaire de spécifier l'ordre dans lequel les ClassifierElements agrégés par un ClassifierService sont présentés aux paquets entrant dans le classeur. Cette propriété est un entier non signé de 32 bits représentant cet ordre. Les valeurs sont représentées en ordre ascendant : d'abord '1', puis '2', et ainsi de suite. Des valeurs différentes DOIVENT être allouées pour chacun des ClassifierElements agrégés par un certain ClassifierService.

4.4.27 Agrégation EntriesInFilterList

Cette agrégation est une spécialisation de l'agrégation Component ; elle est utilisée pour définir un ensemble d'entrées de filtre (sous classes de FilterEntryBase) qui sont agrégées par une FilterList. La cardinalité de l'agrégation elle-même est 0..1 sur l'extrémité FilterList, et 0..n sur l'extrémité FilterEntryBase. Donc dans le cas général, une entrée de filtre peut exister sans être agrégée dans une FilterList. Cependant, la seule façon dont une entrée de filtre peut figurer dans le modèle d'appareil de QS est d'être agrégée dans une FilterList par cette agrégation. Voir la définition de cette agrégation dans la [RFC3460].

4.4.28 Agrégation ElementInSchedulingService

Cette agrégation concrète représente la relation entre un PacketSchedulingService et l'ensemble de SchedulingElements qui le lie à ses entrées. La définition de classe pour l'agrégation est la suivante :

NOM : ElementInSchedulingService
DESCRIPTION : agrégation utilisée pour lier un PacketSchedlingService aux informations de configuration pour un des éléments (QueuingService ou un autre PacketSchedulingService) qu'il programme.
DÉRIVÉE DE : Component
ABSTRAITE : Faux
PROPRIÉTÉS : GroupComponent[ref PacketSchedulingService[0..1]], PartComponent[ref SchedulingElement[1..n]

4.4.28.1 Référence GroupComponent

Cette propriété est outrepassée dans cette agrégation pour représenter une référence d'objet à un objet PacketSchedulingService (au lieu de l'objet plus générique Service défini dans sa super classe). Il restreint aussi la cardinalité de l'agrégat à 0..1 (au lieu du plus générique 0 ou plus) représentant le fait qu'un SchedulingElement existe dans le contexte d'au plus un PacketSchedulingService.

4.4.28.2 Référence PartComponent

Cette propriété est outrepassée dans cette agrégation pour représenter une référence d'objet à un objet SchedulingElement (au lieu de l'objet plus générique Service défini dans sa super classe). Cet objet représente un seul élément de programmation pour le programmateur. Il restreint aussi la cardinalité de SchedulingElement à 1..n (au lieu du plus générique 0 ou plus) représentant le fait qu'un PacketSchedulingService inclut toujours au moins un SchedulingElement.

5. Déclaration de propriété intellectuelle

L'IETF ne prend pas position sur la validité et la portée de tout droit de propriété intellectuelle ou autres droits qui pourrait être revendiqués au titre de la mise en œuvre ou l'utilisation de la technologie décrite dans le présent document ou sur la mesure dans laquelle toute licence sur de tels droits pourrait être ou n'être pas disponible ; pas plus qu'elle ne prétend avoir accompli aucun effort pour identifier de tels droits. Les informations sur les procédures de l'ISOC au sujet des droits dans les documents de l'ISOC figurent dans les BCP 78 et BCP 79.

Des copies des dépôts d'IPR faites au secrétariat de l'IETF et toutes assurances de disponibilité de licences, ou le résultat de tentatives faites pour obtenir une licence ou permission générale d'utilisation de tels droits de propriété par ceux qui mettent en œuvre ou utilisent la présente spécification peuvent être obtenues sur répertoire en ligne des IPR de l'IETF à http://www.ietf.org/ipr .

L'IETF invite toute partie intéressée à porter son attention sur tous copyrights, licences ou applications de licence, ou autres droits de propriété qui pourraient couvrir les technologies qui peuvent être nécessaires pour mettre en œuvre la présente norme. Prière d'adresser les informations à l'IETF à ietf-ipr@ietf.org .

6. Remerciements

Les auteurs tiennent à remercier les participants aux groupes de travail Cadre de politique et Services différenciés pour leurs nombreux commentaires et suggestions utiles. Des remerciements particuliers à Joel Halpern, qui a fourni des indications techniques clés durant les dernières étapes du développement de ce document.

7. Considérations sur la sécurité

Comme les [RFC3060] et [RFC3460], le présent document définit un modèle d'informations qui ne peut pas être mis en œuvre directement. Par conséquent, les questions de sécurité ne se posent pas tant qu'il n'est pas transposé en un modèle de données applicable, comme un schéma de MIB, de PIB, ou LDAP. Voir dans la [RFC3060] une discussion générale des considérations de sécurité pour les modèles d'informations. Voir aussi dans la [RFC3289] (qui est en fait un modèle de données qui correspond dans une large mesure au modèle d'informations QDDIM) la discussion des implications pour la sécurité des objets spécifiques du modèle.

8. Références

8.1 Références normatives

[CIM] "Common information model (CIM) Schema, version 2.5". Distributed Management Task Force, Inc., disponible à http://www.dmtf.org/standards/cim_schema_v25.php .

[IEEE802Q] ANSI/IEEE std 802.1Q, "Virtual Bridged Local Area Networks", Approuvée le 8 décembre 1998.

[RFC0791] J. Postel, éd., "Protocole Internet - Spécification du protocole du programme Internet", STD 5, septembre 1981.

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997.

[RFC2474] K. Nichols, S. Blake, F. Baker et D. Black, "Définition du champ Services différenciés (DS) dans les en-têtes IPv4 et IPv6", décembre 1998. (*MàJ par* RFC3168, RFC3260) *(P.S.)*

[RFC2597] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, "Groupe PHB Transmission assurée", juin 1999. (*MàJ par* RFC3260)*PS*

[RFC3060] B. Moore et autres, "Spécification du modèle d'information de cœur de politique -- version 1", février 2001. (*MàJ par* RFC3460) *(P.S.)*

[RFC3140] D. Black et autres, "Codes d'identification de comportement par bond", juin 2001. *(P.S.)*

[RFC3460] B. Moore, éd., "Extensions au modèle d'information de cœur de politique (PCIM)", janvier 2003. *(P.S.)*

8.2 Références pour information

[RED] Voir http://www.aciri.org/floyd/red.html

[RFC1633] R. Braden, D. Clark et S. Shenker, "Intégration de services dans l'architecture de l'Internet : généralités", juin 1994. *(Info.)*

[RFC2475] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang et W. Weiss, "Architecture pour services différenciés", décembre 1998. *(MàJ par* RFC3260)

[RFC3198] A. Westerinen et autres, "Terminologie pour la gestion fondée sur la politique", novembre 2001. *(Information)*

[RFC3246] B. Davie et autres, "Comportement par bond de transmission accélérée", mars 2002. *(P.S.)*

[RFC3289] F. Baker, K. Chan, A. Smith, "Base de données d'informations de gestion pour l'architecture de services différenciés", mai 2002. *(P.S.)*

[RFC3290] Y. Bernet et autres, "Modèle informel de gestion pour routeurs Diffserv", mai 2002. *(Information)*

[RFC3317] K. Chan, "Base de données d'informations de politique de qualité de service pour services différenciés", mars 2003. *(Information)*

[RFC3644] Y. Snir et autres, "Modèle d'informations de politique de qualité de service (QS)", novembre 2003. *(P.S.)*

9. Appendice A : Instances de dénomination dans une mise en œuvre native de CIM

Suivant le précédent établi par la [RFC3060], le présent document a placé les détails de la façon de désigner les instances de ses classes dans une mise en œuvre CIM native dans un appendice. Comme l'Appendice A de la [RFC3060] a une longue discussion des principes généraux de la dénomination dans CIM, le présent appendice ne répète pas ces informations. Le lecteur intéressé par une discussion plus générale sur la façon dont sont nommées les instances dans une mise en œuvre CIM native devrait se référer à la [RFC3060].

9.1 Instances de dénomination des classes dérivées de Service

La plupart des classes définies dans ce modèle sont dérivées de la classe Service de CIL. Bien que Service soit une classe abstraite, elle a néanmoins des propriétés clés incluses au titre de sa définition. L'objet de l'inclusion de propriétés clés dans une classe abstraite est d'avoir des instances de toutes ses sous classes instantiables nommées de la même façon. Donc, la majorité des classes dans ce modèle désigne ses instances exactement de la même façon : avec les deux propriétés clés CreationClassName et Name dont elles héritent de Service.

9.2 Instances de dénomination des sous classes de FilterEntryBase

Comme Service, FilterEntryBase (définie dans la [RFC3460]) est une classe abstraite qui inclut des propriétés clés dans sa définition. FilterEntryBase a quatre propriétés clés. Deux d'entre elles, SystemCreationClassName et SystemName, lui sont propagées via l'association faible FilterEntryInSystem. Les deux auores, CreationClassName et Name, sont natives de FilterEntryBase.

Donc, les instances de toutes les sous classes de FilterEntryBase, incluant la classe PreambleFilter définie ici, sont nommées de la même manière : avec les quatre propriétés clés qu'elles héritent de FilterEntryBase.

9.3 Instances de dénomination de ProtocolEndpoint

La classe ProtocolEndpoint hérite ses propriétés clés de sa superclasse, ServiceAccessPoint. Ces propriétés clés donnent la même structure de désignation que celle vue auparavant : deux propriétés clés propagées de SystemCreationClassName et SystemName, plus deux propriétés clés natives CreationClassName et Name.

9.4 Instances de dénomination de BufferPool

À la différence des autres classes de ce modèle, BufferPool n'est pas dérivée de Service. Par conséquent, elle n'hérite pas ses propriétés clés de Service. À la place, elle hérite une de ses propriétés clés, CollectionID, de sa superclasse Collection, et ajoute son autre propriété clé, CreationClassName, dans sa propre définition.

9.4.1 Propriété CollectionID

CollectionID est une propriété de chaîne d'une longueur maximum de 256 caractères. Elle identifie le réservoir de mémoire tampon. Noter que cette propriété est définie dans la superclasse de la classe BufferPool, CollectionOfMSEs, mais pas comme une propriété clé. Elle est outrepassée dans BufferPool, pour en faire une partie de la clé composite de cette classe.

9.4.2 Propriété CreationClassName

Cette propriété est une propriété de chaîne d'une longueur maximum de 256 caractères. Elle est réglée à "CIM_BufferPool" si cette classe est directement instanciée, ou au nom de classe de la sous classe BufferPool qui est créée.

9.5 Instances de dénomination de SchedulingElement

Cette classe n'a pas encore été incorporée dans le modèle CIM, de sorte qu'elle n'a pas encore de propriété de désignation CIM. Cependant, si le schéma normal est suivi, les instances seront nommées avec deux propriétés CreationClassName et Name.

10. Adresse des auteurs

Bob Moore
P. O. Box 12195, BRQA/B501/G206
3039 Cornwallis Rd.
Research Triangle Park, NC 27709-2195
téléphone : (919) 254-4436
mél : remoore@us.ibm.com

David Durham
Intel
2111 NE 25th Avenue
Hillsboro, OR 97124
téléphone : (503) 264-6232
mél : david.durham@intel.com

John Strassner
INTELLIDEN, Inc.
90 South Cascade Avenue
Colorado Springs, CO 80903
téléphone : (719) 785-0648
mél : john.strassner@intelliden.com

Andrea Westerinen
Cisco Systems, Bldg 20
725 Alder Drive
Milpitas, CA 95035
mél : andreaw@cisco.com

Walter Weiss
Ellacoya Networks
7 Henry Clay Dr.
Merrimack, NH 03054
téléphone : (603) 879-7364
mél : walterweiss@attbi.com

11. Déclaration complète de droits de reproduction

Remerciement
Le financement de la fonction d'édition des RFC est actuellement fourni par l'Internet Society.