UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

UIT-T Q.816

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

(01/2001)

SÉRIE Q: COMMUTATION ET SIGNALISATION

Interface Q3

Services RGT à architecture CORBA

Recommandation UIT-T Q.816

(Antérieurement Recommandation du CCITT)

RECOMMANDATIONS UIT-T DE LA SÉRIE Q

COMMUTATION ET SIGNALISATION

SIGNALISATION DANS LE SERVICE MANUEL INTERNATIONAL	Q.1–Q.3
EXPLOITATION INTERNATIONALE AUTOMATIQUE ET SEMI-AUTOMATIQUE	Q.4–Q.59
FONCTIONS ET FLUX D'INFORMATION DES SERVICES DU RNIS	Q.60–Q.99
CLAUSES APPLICABLES AUX SYSTÈMES NORMALISÉS DE L'UIT-T	Q.100–Q.119
SPÉCIFICATIONS DES SYSTÈMES DE SIGNALISATION N° 4 ET N° 5	Q.120–Q.249
SPÉCIFICATIONS DU SYSTÈME DE SIGNALISATION N° 6	Q.250–Q.309
SPÉCIFICATIONS DU SYSTÈME DE SIGNALISATION R1	Q.310–Q.399
SPÉCIFICATIONS DU SYSTÈME DE SIGNALISATION R2	Q.400–Q.499
COMMUTATEURS NUMÉRIQUES	Q.500–Q.599
INTERFONCTIONNEMENT DES SYSTÈMES DE SIGNALISATION	Q.600–Q.699
SPÉCIFICATIONS DU SYSTÈME DE SIGNALISATION N° 7	Q.700–Q.799
INTERFACE Q3	**Q.800–Q.849**
SYSTÈME DE SIGNALISATION D'ABONNÉ NUMÉRIQUE N° 1	Q.850–Q.999
RÉSEAUX MOBILES TERRESTRES PUBLICS	Q.1000–Q.1099
INTERFONCTIONNEMENT AVEC LES SYSTÈMES MOBILES À SATELLITES	Q.1100–Q.1199
RÉSEAU INTELLIGENT	Q.1200–Q.1699
PRESCRIPTIONS ET PROTOCOLES DE SIGNALISATION POUR LES IMT-2000	Q.1700–Q.1799
RNIS À LARGE BANDE	Q.2000–Q.2999

Pour plus de détails, voir la Liste des Recommandations de l'UIT-T.

Recommandation UIT-T Q.816

Services RGT à architecture CORBA

Résumé

La présente Recommandation définit un ensemble de services qui forment, en association avec UIT-T X.780, un cadre pour les interfaces RGT en architecture CORBA. Elle spécifie les exigences relatives au protocole, à l'usage du service commun aux objets et aux services supports spécifiques du RGT. Un module de langage IDL définit dans l'architecture CORBA les interfaces avec les services supports spécifiques du RGT.

Source

UIT-T Q.816 de l'UIT-T, élaborée par la Commission d'études 4 (2001-2004) de l'UIT-T, a été approuvée le 19 janvier 2001 selon la procédure définie dans la Résolution 1 de l'AMNT.

Mots clés

Architecture du courtier de requêtes pour objets communs (CORBA), langage de définition d'interface (IDL), services CORBA, traitement réparti, interfaces RGT, objets gérés.

AVANT-PROPOS

L'UIT (Union internationale des télécommunications) est une institution spécialisée des Nations Unies dans le domaine des télécommunications. L'UIT-T (Secteur de la normalisation des télécommunications) est un organe permanent de l'UIT. Il est chargé de l'étude des questions techniques, d'exploitation et de tarification, et émet à ce sujet des Recommandations en vue de la normalisation des télécommunications à l'échelle mondiale.

L'Assemblée mondiale de normalisation des télécommunications (AMNT), qui se réunit tous les quatre ans, détermine les thèmes d'étude à traiter par les Commissions d'études de l'UIT-T, lesquelles élaborent en retour des Recommandations sur ces thèmes.

L'approbation des Recommandations par les Membres de l'UIT-T s'effectue selon la procédure définie dans la Résolution 1 de l'AMNT.

Dans certains secteurs des technologies de l'information qui correspondent à la sphère de compétence de l'UIT-T, les normes nécessaires se préparent en collaboration avec l'ISO et la CEI.

NOTE

Dans la présente Recommandation, l'expression "Administration" est utilisée pour désigner de façon abrégée aussi bien une administration de télécommunications qu'une exploitation reconnue.

DROITS DE PROPRIÉTÉ INTELLECTUELLE

L'UIT attire l'attention sur la possibilité que l'application ou la mise en œuvre de la présente Recommandation puisse donner lieu à l'utilisation d'un droit de propriété intellectuelle. L'UIT ne prend pas position en ce qui concerne l'existence, la validité ou l'applicabilité des droits de propriété intellectuelle, qu'ils soient revendiqués par un Membre de l'UIT ou par une tierce partie étrangère à la procédure d'élaboration des Recommandations.

A la date d'approbation de la présente Recommandation, l'UIT n'avait pas été avisée de l'existence d'une propriété intellectuelle protégée par des brevets à acquérir pour mettre en œuvre la présente Recommandation. Toutefois, comme il ne s'agit peut-être pas de renseignements les plus récents, il est vivement recommandé aux responsables de la mise en œuvre de consulter la base de données des brevets du TSB.

TABLE DES MATIÈRES

Page

Recommandation UIT-T Q.816

Services RGT à architecture CORBA

1 Domaine d'application

L'architecture RGT définie dans UIT-T M.3010 (2000) présente les concepts découlant du traitement réparti et inclut l'utilisation de multiples protocoles de gestion. Les premières spécifications d'interface RGT ont été élaborées pour les interfaces intra-RGT et inter-RGT au moyen de la notation des Directives pour la définition des objets gérés (GDMO) issue de la gestion des systèmes OSI avec le protocole commun de transfert des informations de gestion (CMIP) en tant que protocole. L'interface inter-RGT (X) comportait les deux protocoles – CMIP et GIOP/IIOP de l'architecture CORBA – en tant qu'options au niveau de la couche application.

Etant une technique de traitement réparti, l'utilisation de l'architecture CORBA est actuellement envisagée dans l'architecture de communication RGT en raison surtout de son acceptation par l'industrie informatique. L'on s'attend que cette acceptation renforcera la disponibilité des interfaces en architecture CORBA grâce à de meilleurs outils de développement et à une large extension des compétences spécialisées dans l'élaboration de telles interfaces. Cette technique, mise au point par le Groupe de gestion d'objets (OMG), est également en cours d'examen par de multiples industries. Les spécifications faisant appel à cette technique facilitent la prise en charge d'interfaces de programmation d'application (API) normalisées et de liens linguistiques avec les langages de programmation. Elles facilitent également la portabilité logicielle. Les solutions d'interopérabilité offertes par le gestionnaire de requêtes d'objets, combinées avec les protocoles intergestionnaires ORB, assurent l'interopérabilité entre clients et serveurs. Alors que le protocole CMIP et les modèles d'informations apportent des solutions d'interopérabilité entre systèmes gestionnaires et systèmes agents, l'architecture CORBA définit des interactions entre objets dans lesquelles ceux-ci peuvent être répartis.

1.1 Objet

Plusieurs groupes mettent actuellement au point des spécifications de gestion de réseau faisant appel aux techniques de modélisation CORBA avec la notation IDL et des services CORBA. Le domaine d'application de la présente Recommandation consiste à définir des exigences de protocole et des services communs utilisables dans la spécification d'interfaces interopérables de gestion de réseau en architecture CORBA. Les exigences posées par les interfaces "X" sont différentes de celles des interfaces qui sont utilisées "en interne" dans un RGT, c'est-à-dire les interfaces "Q". Le domaine de la présente Recommandation s'applique à toutes les interfaces du RGT dans lesquelles l'architecture CORBA peut être utilisée. L'on s'attend qu'il ne sera pas nécessaire de faire appel, dans toutes les interfaces RGT, à toutes les capacités et à tous les services ici définis. Cela implique que ce cadre général pourra être utilisé pour des interfaces entre systèmes de gestion à tous les niveaux d'abstraction (administration interdomaniale et intradomaniale) ainsi qu'entre systèmes de gestion et éléments de réseau.

La présente Recommandation est destinée à être utilisée par divers groupes spécifiant des interfaces de gestion de réseau. Un certain nombre de facteurs sont pris en considération: la version de l'architecture CORBA à utiliser, l'ensemble des services communs aux objets CORBA utilisé et les services additionnels. La présente Recommandation, associée aux directives de modélisation des objets qui sont définies dans UIT-T X.780, forme un *cadre général* pour les interfaces RGT en architecture CORBA. L'utilisation d'un cadre commun aux interfaces de gestion des télécommunications présente plusieurs avantages, dont par exemple les suivants:

– réutilisation plus facile de modèles mis au point pour répondre aux exigences génériques des télécommunications;

– configuration des services CORBA pour usage par l'industrie des télécommunications;

– définition plus facile de nouveaux services pour le RGT, réutilisation de la sémantique du riche ensemble existant de modèles; et

– harmonisation de l'approche par modélisation entre groupes faisant appel à une source unique comme UIT-T X.720, X.721 et X.722 pour le protocole CMIP.

La réutilisation d'un cadre commun et d'un modèle d'information générique pour diverses techniques de réseau et diverses applications de gestion de réseau accélérera l'introduction de nouveaux services de réseau tout en minimisant les coûts de développement des systèmes de gestion de réseau.

L'industrie des télécommunications a investi beaucoup de temps et d'énergie dans la mise au point de modèles d'information pour le protocole CMIP de gestion de réseau. Le présent cadre général a comme objectif premier de réutiliser ces modèles d'information en permettant leur traduction en langage de définition d'interface (IDL) dans l'architecture CORBA avec peu de modifications de la sémantique (voir UIT-T X.780). Il en découle que les premiers modèles d'information en langage IDL sont appelés à être déduits des modèles CMIP.

En plus du parti tiré des modèles d'information CMIP, un autre objectif du cadre général consiste à tirer parti de l'architecture CORBA. Le cadre général amplifie les fonctions définies dans les spécifications CORBA, y compris un ensemble de services communs aux objets. De même le cadre général tente de réutiliser les approches et structures de conception CORBA chaque fois qu'elles sont applicables. Finalement, bien que la réutilisation des modèles existants soit importante, il est également important que le cadre prenne en charge le développement de nouveaux modèles. Le présent cadre ne nécessite pas la mise au point d'un modèle de directives GDMO avant le développement d'un modèle en langage IDL. En fait, la mise au point d'un nouveau modèle d'information en langage IDL à utiliser dans le présent cadre ne pose pas de problèmes particuliers et les directives correspondantes sont fournies.

1.2 Application

Au fur et à mesure de l'introduction de l'architecture CORBA dans le RGT, différents scénarios sont possibles, allant de l'utilisation de passerelles effectuant des conversions entre systèmes utilisant différents protocoles de gestion de réseau aux cas où l'architecture CORBA est prise en charge naturellement par les systèmes communicants. L'application du présent cadre général vise les scénarios dans lesquels les deux systèmes – géré et gérant – offrent des interfaces CORBA.

Le cadre ne traite pas les autres scénarios d'interfonctionnement nécessitant des systèmes "passerelles" dans lesquels des conversions de protocole et de modèle d'information sont nécessaires pour réaliser l'interopérabilité. En particulier, ce cadre n'est pas spécifiquement conçu pour prendre en charge la construction de passerelles entre applications de gestion de réseau CORBA et CMIP bien que la sémantique des modèles existants soit conservée par le présent cadre. Un système de gestion peut cependant avoir à prendre en charge de multiples protocoles afin d'interfonctionner dans divers environnements.

Une méthode fondée sur les passerelles a déjà été mise au point et normalisée par le groupe JIDM (*joint inter-domain management*, gestion mixte interdomaniale). Cette méthode des passerelles offre une correspondance univoque entre toutes les créations syntaxiques et toutes les capacités disponibles avec le protocole CMIP et les directives GDMO. Un grand nombre des services et capacités CORBA ne sont cependant pas réutilisés par cette méthode parce que le problème résolu consiste à faciliter l'interfonctionnement avec des systèmes qui ont été déployés au moyen du protocole CMIP. En revanche, le domaine problématique d'application de ce cadre consiste à prendre en charge des interfaces de gestion de réseau initialement fondées sur des normes d'architecture CORBA. Une telle approche tire parti des avantages offerts par l'architecture CORBA en tant que technique utilisée par de multiples industries.

La Recommandation UIT-T X.780 [1] complète la présente Recommandation et définit les directives de modélisation des objets, les hyperclasses pour tous les objets gérés ainsi que les ateliers d'objets

gérés à utiliser dans le présent cadre général, avec un ensemble normalisé de notifications. Ensemble, UIT-T X.780 et la présente Recommandation définissent un *cadre général* pour interfaces RGT en architecture CORBA. De même, UIT-T M.3120 offre une version CORBA en langage IDL du modèle d'information de réseau générique initialement défini dans UIT-T M.3100. La version en langage IDL fait suite aux directives de modélisation d'objets contenues dans UIT-T X.780 et vise à s'adapter aux services définis ci-après.

1.3 Structure de la Recommandation

La présente Recommandation est structurée comme suit:

Paragraphe 1 Introduction, structure de la Recommandation, mises à jour et liste des rappels.

Paragraphe 2 Références.

Paragraphe 3 Définition des termes et abréviations utilisés dans le reste de la Recommandation.

Paragraphe 4 Exigences pour les services RGT en architecture CORBA. Il s'agit des objectifs de conception que les services doivent atteindre.

Paragraphe 5 Exigences relatives à la version du gestionnaire ORB et du service en architecture CORBA. Un aperçu général des interfaces de gestion de réseau est également fourni.

Paragraphe 6 Exigences relatives à l'utilisation de services communs aux objets CORBA pour interfaces de gestion de réseau.

Paragraphe 7 Définition des services supports spécifiques du RGT. Les interfaces IDL pour ces services supports sont définies en Annexe A.

Paragraphe 8 Directives d'observance et de conformité.

Annexe A Spécification en langage IDL des services supports spécifiques du RGT.

Annexe B Le langage de contrainte BNF.

Appendice I Scénarios d'interfonctionnement entre modèles utilisant le cadre UIT et modèles conformes aux Forums ADSL/ATMF.

Appendice II Evénements structurés de filtrage, natifs ou convertis.

Appendice III Bibliographie.

1.4 Conventions de la Recommandation

Un petit nombre de conventions sont appliquées dans la présente Recommandation afin d'informer le lecteur de l'intention du texte. Bien que la plus grande partie de la Recommandation soit normative, les alinéas qui décrivent succinctement les exigences obligatoires qu'un système de gestion (gérant ou géré) doit respecter sont précédés de la lettre "R" en caractère gras entre parenthèses, suivie d'un nom court indiquant le sujet de la prescription et d'un numéro. Par exemple:

(R) EXEMPLE-1 Exemple de prescription obligatoire.

Les exigences dont l'application par un système de gestion est facultative sont précédées de la lettre "O" au lieu de "R". Par exemple:

(O) OPTION-1 Exemple de prescription facultative.

Les déclarations d'exigences servent à créer des profils d'observance et de conformité.

La présente Recommandation donne, dans une annexe normative, de nombreux exemples en langage IDL de l'architecture CORBA et de spécifications IDL des services spécifiques du RGT ainsi que de types de données supports. Les spécifications en langage IDL sont écrites en caractères "courier" de corps 9:

```
// Example IDL
interface foo {
       void operation1 ();
};
```

Les instructions d'extraction et de compilation du langage IDL à partir d'une version électronique de la présente Recommandation sont présentées dans le paragraphe suivant.

1.5 Compilation du langage IDL

Un avantage procuré par l'utilisation du langage IDL pour spécifier les interfaces de gestion de réseau est que ce langage peut être "compilé" en code de programmation par des outils qui accompagnent un gestionnaire ORB. Cela permet en fait d'automatiser la mise au point d'une partie du code nécessaire pour permettre l'interfonctionnement d'applications de gestion de réseau. La présente Recommandation contient, en annexe, un code que les réalisateurs pourront éventuellement extraire et compiler. L'Annexe A étant normative, il convient qu'elle soit utilisée par les développeurs mettant en œuvre des systèmes conformes à la présente norme. Le langage IDL présenté dans la présente Recommandation a été vérifié avec deux compilateurs afin de garantir son exactitude. Il faut utiliser un compilateur prenant en charge la version CORBA spécifiée au § 5.2.

L'Annexe A a été présentée de façon à en faciliter le couper-coller dans un fichier de texte en clair pouvant ensuite être compilé. A cette fin, quelques conseils sont donnés ci-après.

1) Le couper-coller paraît mieux fonctionner à partir de la version Microsoft® Word® de la présente Recommandation. Le couper-coller à partir d'un fichier en format Adobe® Acrobat® semble inclure les en-têtes et pieds de page, qui ne peuvent pas être compilés.

2) Il y a lieu de mémoriser l'ensemble de l'Annexe A, à partir de la ligne "/* Ce code IDL…" jusqu'à la fin, dans un fichier nommé "itut_q816.idl", dans un répertoire où il sera trouvé par le compilateur IDL.

3) Les titres intégrés dans l'Annexe A n'ont pas besoin d'être supprimés. Ils ont été encapsulés dans les commentaires IDL et seront ignorés par le compilateur.

4) Les commentaires commençant par la séquence spéciale "/**" sont reconnus par les compilateurs qui convertissent IDL en HTML. Ces commentaires comportent souvent des instructions de formatage pour ces compilateurs. Ceux qui travailleront avec l'IDL souhaiteront peut-être produire des fichiers HTML car ces derniers possèdent des liens qui permettent une navigation rapide d'un fichier à un autre.

5) L'Annexe A a été formatée avec des tabulations à intervalles de 8 espaces et avec des retours à la ligne à codage fixe qui devraient permettre à presque tous les traitements de texte de travailler avec ce texte.

2 Références

La présente Recommandation se réfère à certaines dispositions des Recommandations UIT-T et textes suivants qui, de ce fait, en sont partie intégrante. Les versions indiquées étaient en vigueur au moment de la publication de la présente Recommandation. Toute Recommandation ou tout texte étant sujet à révision, les utilisateurs de la présente Recommandation sont invités à se reporter, si possible, aux versions les plus récentes des références normatives suivantes. La liste des Recommandations de l'UIT-T en vigueur est régulièrement publiée.

[1] UIT-T X.780 (2001), *Directives pour la définition d'objets gérés CORBA*.

[2] OMG Document formal/99-10-07, *The Common Object Request Broker: Architecture and Specification*, Revision 2.3.1.

[3] OMG Document formal/2000-11-1, *Interoperable Naming Service Specification*.

[4] OMG Document formal/2000-06-20, *Notification Service Specification*, Version 1.0.

[5] OMG Document formal/00-01-04, *Telecom Log Service Specification*, Version 1.0.

[6] OMG Document formal/2000-06-25, *Security Services Specification*, Version 1.5.

[7] OMG Document formal/2000-06-28, *Transaction Service Specification*, Version 1.1.

[8] OMG Document formal/2000-10-58, *CORBA Messaging*.

[9] OMG Document formal/2000-08-01, *Interworking Between CORBA and TMN Systems Specification.*

[10] IETF RFC 2246 (1999), *The TLS Protocol Version 1.0.*

[11] IEEE/ANSI Std 1003.2-1992, *Information Technology – Portable Operating System Interface (POSIX) Part 2: Shell and Utilities.*

[12] IETF RFC 2253 (1997), *Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names.*

[13] ETSI TS 132 106-3 (2000), *Universal Mobile Telecommunications System (UMTS), Telecommunication Management; Configuration Management; Part 3: Notification Integration Reference Point: CORBA solution set version 1.1* (3G TS 32 106-3 Release 1999).

[14] UIT-T X.701 (1997), *Technologies de l'information – Interconnexion des systèmes ouverts – Aperçu général de la gestion-systèmes.*

[15] UIT-T X.703 (1997), *Technologies de l'information – Architecture de gestion répartie ouverte.*

3 Définitions et abréviations

3.1 Définitions issues de UIT-T X.701

Les termes suivants, utilisés dans la présente Recommandation, sont définis dans UIT-T X.701:

– classe d'objets gérés;

– gestionnaire;

– agent.

3.2 Définitions issues de UIT-T X.703

Le terme suivant, utilisé dans la présente Recommandation, est défini dans UIT-T X.703:

– notification.

3.3 Définitions additionnelles

3.3.1 canal d'événements: objet support dans un système géré avec une ou plusieurs interfaces permettant à un système gérant de recevoir des notifications de ce système géré.

NOTE – Il existe deux modèles pour la remise de notifications par canal d'événements (voir UIT-T X.703):

- dans le modèle de remise d'événement par distribution sélective, le système gérant utilise une ou plusieurs instances d'interface de canal d'événements pour enregistrer une ou plusieurs références d'interface de système gérant pouvant être utilisées par le canal afin d'appeler des opérations de distribution sélective d'événement;
- dans le modèle de remise d'événement par extraction sélective, le système gérant utilise une ou plusieurs références d'interface de canal d'événements pour appeler des opérations renvoyant des informations de notification.

3.4 Abréviations

La présente Recommandation utilise les abréviations suivantes:

AMI invocation de méthode asynchrone (*asynchronous messaging invocation*)

API interface de programmation d'application (*application programming interface*)

ASN.1 notation de syntaxe abstraite numéro un (*abstract syntax notation one*)

ATM mode de transfert asynchrone (*asynchronous transfer mode*)

AVA assertion de valeur d'attribut (*attribute value assertion*)

CMIP protocole commun de transfert d'informations de gestion (*common management information protocol*)

CORBA architecture du courtier de requêtes pour objets communs (*common object request broker architecture*)

COS services communs aux objets (*common object services*)

DN nom distinctif (*distinguished name*)

EMS système de gestion d'élément (*element management system*)

FIFO premier entré, premier sorti (*first in, first out*)

GDMO directives pour la définition des objets gérés (*guidelines for the definition of managed objects*)

GIOP protocole général d'interopérabilité (*general interoperability protocol*)

HTML langage hypertexte (*hypertext markup language*)

ID identificateur

IDL langage de définition d'interface (*interface definition language*)

IEEE Institut des ingénieurs électriciens et électroniciens (*Institute of electrical and electronics engineers*)

IETF Groupe de travail d'ingénierie Internet (*Internet engineering task force*)

IIOP protocole d'interopérabilité Internet (*Internet interoperability protocol*)

IOR référence d'objet interopérable (*interoperable object reference*)

JIDM gestion interdomaniale mixte (*joint inter-domain management*)

MO objet géré (*managed object*)

MOO opération sur objets multiples (*multiple object operation*)

NE élément de réseau (*network element*)

NMS système de gestion de réseau (*network management system*)

OAM&P exploitation, administration, maintenance et fourniture (*operations, administration, maintenance and provisioning*)

OID identificateur d'objet (*object identifier*)

OMG groupe de gestion d'objets (*object management group*)

ORB courtier de demandes d'objets (*object request broker*)

OSI interconnexion des systèmes ouverts (*open systems interconnection*)

PDU unité de données protocolaire (*protocol data unit*)

PM gestion de la performance (*performance management*)

POA adaptateur d'objet portable (*portable object adapter*)

POP point de présence (*point of presence*)

POSIX interface portable de systèmes d'exploitation (*portable operating system interface*)

QS qualité de service

RDN nom distinctif relatif (*relative distinguished name*)

RGT réseau de gestion des télécommunications

SDH hiérarchie numérique synchrone (*synchronous digital hierarchy*)

SONET réseau optique synchrone (*synchronous optical network*)

SSL couche des numéros de connexion logique sécurisés (*secure socket layer*)

TII invocation indépendante du temps (*time-independent invocation*)

TLS sécurité de la couche de transport (*transport layer security*)

TTP point de terminaison de chemin (*trail termination point*)

UID identificateur universel (*universal identifier*)

UIT-T Union internationale des télécommunications – Secteur de la normalisation des télécommunications

UML langage de modélisation unifié (*unified modeling language*)

UTC temps universel coordonné (*universal time code*)

4 Objectifs et exigences des services RGT en architecture CORBA

Le présent paragraphe décrit les principaux objectifs du cadre de services ainsi que les exigences facilitant la prise en charge de ces objectifs par les services RGT en architecture CORBA. Le § 4.1 présente les objectifs du cadre CORBA. Le § 4.2 contient la terminologie et les exigences. Les prescriptions du présent paragraphe sont celles que le cadre général doit observer. Elles sont fondées sur les besoins en gestion des télécommunications. Les § 5, 6, 7 et 8 décrivent ensuite un cadre répondant à ces besoins et définissent la façon de satisfaire aux exigences du présent paragraphe en utilisant d'une certaine façon l'architecture CORBA. Les règles des § 5, 6, 7 et 8 sur la façon d'utiliser l'architecture CORBA sont également considérées comme des exigences.

4.1 Objectifs

La présente Recommandation vise à déterminer un cadre permettant de définir la façon dont il convient de modéliser les interfaces prises en charge par les systèmes de gestion et par les éléments de réseau. Quelques objectifs du cadre couramment visés sont les suivants:

- interopérabilité entre applications;
- usage commun des services communs aux objets;
- transparence du modèle d'information;

Le présent paragraphe développe ces trois objectifs.

4.1.1 Interopérabilité des applications

Un des objectifs essentiels de l'architecture RGT (et en particulier de l'architecture d'information), consiste à promouvoir un cadre normalisé pour assurer l'interopérabilité et l'échange d'informations entre systèmes à partir d'un ensemble de divers fournisseurs de systèmes de gestion de réseau. L'interopérabilité entre systèmes implique de nombreux aspects de développement. Au niveau de la

couche la plus basse, un mécanisme commun de communication doit être en place afin de prendre en charge une syntaxe commune, l'établissement de la connexité et l'échange entre systèmes de demandes/réponses concernant les opérations. Cet aspect de l'interopérabilité est implicitement pris en charge par la spécification CORBA.

Pour le RGT, il est nécessaire d'assurer l'interopérabilité des applications. En d'autres termes, des systèmes de gestion issus de divers fournisseurs seront utilisés dans un même RGT d'administration afin de prendre en charge différentes fonctions nécessaires pour assurer la gestion de ses réseaux. Afin de simplifier l'intégration des systèmes de ces divers fournisseurs, ceux-ci doivent s'entendre sur la sémantique des informations à échanger. C'est à cette fin qu'un modèle d'information est spécifié. Les directives pour la définition de modèles d'information en architecture CORBA sont spécifiées dans UIT-T X.780, mais les services définis ci-après doivent être compatibles avec ces directives.

4.1.2 Usage commun des services communs relatifs aux objets dans l'architecture CORBA

Un deuxième aspect de ce cadre général est la définition de l'usage commun et la configuration de l'environnement de traitement réparti choisi. Cet aspect du cadre devra indiquer les anticipations raisonnables que les fournisseurs de système de gestion de réseau peuvent avoir les uns envers les autres. Plutôt que de redéfinir les capacités d'interface requises afin de prendre en charge des fonctions de gestion de réseau communes à chaque modèle d'information, comme le nommage des objets et le filtrage des notifications, les directives de modélisation de UIT-T X.780 se fondent sur un ensemble de services supports qui permettent de simplifier les modèles d'information tout en améliorant l'interopérabilité.

Lors de la définition de ces services, un effort spécial sera consenti pour faire usage des services communs aux objets de l'architecture CORBA. Plus précisément, la présente Recommandation traitera de l'utilisation du service CORBA de courtier ORB et des services CORBA communs aux objets (COS) qui auront une incidence sur l'interopérabilité des systèmes (c'est-à-dire que les questions impliquant l'utilisation de l'architecture CORBA dans un seul système sont hors du domaine d'application de la présente Recommandation). Lorsque les besoins de la gestion de réseau ne pourront pas être satisfaits par les services COS de l'architecture CORBA, des services additionnels seront définis.

4.1.3 Transparence du modèle d'information

Si l'architecture CORBA est utilisée dans des parties de l'architecture RGT où des modèles d'information existants (comme les directives GDMO) sont bien établis, le cadre doit assurer la réutilisation de ces modèles sans aucune modification majeure.

La présente Recommandation est principalement orientée vers l'ensemble des services requis pour permettre d'utiliser avec un niveau d'efficacité suffisant les modèles existants tels qu'ils ont été initialement prévus.

4.2 Influence de la modélisation informationnelle

Comme décrit dans le paragraphe précédent, la modélisation explicite des ressources gérables de part et d'autre d'une interface est au centre de l'interopérabilité des applications. Les directives définissant les objets gérés CORBA détaillés dans UIT-T X.780 décrivent les règles de modélisation des ressources gérables. Elles impliquent également plusieurs décisions qui doivent être prises en charge par le réseau général de services RGT en architecture CORBA. Le présent paragraphe résume ces points.

4.2.1 Granularité des accès

La *granularité* d'une interface CORBA se rapporte à la relation entre les ressources qui sont modélisées à une interface donnée et les moyens par lesquels on y accède dans l'architecture

CORBA. UIT-T X.780 utilise une approche de modélisation *à granularité d'instance*, ce qui signifie que chaque ressource modélisée est accessible au moyen d'une unique référence d'objet CORBA, appelée *référence d'objet interopérable* (IOR). Les objets qui représentent des ressources gérables sont appelés *objets gérés*.

4.2.2 Représentation des relations de contenance et de nommage

La contenance est une représentation logique de la façon dont des ressources modélisées contiennent d'autres ressources modélisées. Traditionnellement, la contenance est une relation très importante dans les applications de gestion de réseau car c'est un moyen pratique d'identification du grand nombre de ressources qui doivent être gérées. UIT-T X.780 prescrit qu'un nom unique soit attribué à chaque objet géré, fondé d'une part sur le nom de l'objet qui le contient. Les services RGT en architecture CORBA doivent toujours permettre de mémoriser ces noms (et donc les relations de contenance qu'ils représentent) ainsi que de trouver la référence IOR d'un objet d'après son nom.

4.2.3 Création et suppression d'objets

Le gestionnaire ORB de l'architecture CORBA ne donne pas aux clients le moyen de créer des objets dans des systèmes distants. Ceux-ci instancient par contre des objets de type *atelier* et ces objets d'atelier fournissent des opérations qui peuvent être invoquées par des clients afin de créer des objets dans le système distant. Pour un certain nombre d'objectifs de modélisation informationnelle, UIT-T X.780 spécifie qu'une interface IDL avec un objet d'atelier doit être définie pour chaque interface IDL avec un objet géré d'un modèle d'information. La création d'objets sera donc dépendante du modèle et ne constituera donc pas un bon candidat pour un service RGT en architecture CORBA. La présente Recommandation définit cependant un service pour un système gérant qui recherche un atelier afin de demander la création d'un objet géré dans un système géré.

La suppression d'objets est également un domaine qui nécessite une prise en charge. Les objets CORBA sont souvent supprimés par simple invocation d'une certaine opération de suppression d'objet mais ce n'est pas une bonne méthode pour les applications de gestion de réseau en raison de leur référence à des relations de contenance. La suppression d'un objet contenant d'autres objets a des incidences au-delà de l'objet que l'on supprime. De même, comme décrit dans le paragraphe précédent, une prise en charge est requise pour mémoriser les noms des instances d'objet géré et ces données doivent être mises à jour lors de la suppression d'objets. Les services RGT en architecture CORBA doivent donc assurer la prise en charge de la suppression ordonnée des objets gérés.

4.3 Détection de domaine de visibilité et filtrage

L'aptitude à l'exécution de requêtes complexes (c'est-à-dire d'opérations GET), de mises à jour (c'est-à-dire d'opérations SET), et d'opérations de suppression sur un groupe d'entités au moyen d'une unique demande d'opération est une précieuse caractéristique du RGT. Les systèmes de gestion peuvent avoir à gérer jusqu'à 10^7 instances d'objets gérés. Compte tenu des dimensions de la base d'informations de gestion, un système gérant ne peut pas exécuter efficacement des requêtes ad hoc visant des instances individuelles d'objets gérés (c'est-à-dire d'entités). Le système gérant s'attend plutôt à ce que le système géré prenne en charge un certain niveau d'intelligence.

L'intelligence contenue dans le système géré permet au système gérant de sélectionner un groupe d'entités gérées sur lesquelles certaines opérations seront exécutées. La sélection des entités gérées comprend deux phases: la détection du domaine de visibilité et le filtrage. Ce processus de sélection d'entités gérées est assuré par un service qui sera défini plus loin et qui permet à un système gérant de sélectionner un domaine d'objets sur lesquels il pourra agir (ce domaine étant défini au moyen de relations de contenance; voir § 4.2.2). Une fois que le domaine de visibilité des entités est déterminé, l'opération (spécifiée par le domaine et par la demande filtrée) est exécutée mais seulement sur les entités qui répondent à des critères définis par un filtre.

L'utilisation de la détection de domaine de visibilité et du filtrage dans le présent cadre général permet d'effectuer les opérations suivantes:

– obtention détectée en visibilité et filtrée: cette opération renvoie les valeurs issues de chacune des entités qui répondent aux critères de visibilité et de filtrage (pour une liste d'attributs);

– mise à jour détectée en visibilité et filtrée: cette opération remplace une valeur d'attribut ou ajoute/retranche des valeurs à des attributs à valeur d'ensemble, dans le groupe des entités qui répondent aux critères de visibilité et de filtrage, en fonction des valeurs spécifiées dans la demande détectée et filtrée. Cette opération peut servir à mettre à jour un ou plusieurs attributs contenus dans un même objet ou dans plusieurs objets;

– suppression détectée en visibilité et filtrée: cette opération supprime toutes les entités qui répondent aux critères de détection et de filtrage.

4.3.1 Détection de domaine de visibilité

La détection de domaine de visibilité implique l'identification des entités auxquelles un filtre doit être appliqué. Elle est effectuée sur la base de la hiérarchie de contenance définie au § 4.2.2. Le domaine de visibilité est appliqué à partir d'une certaine entité gérée de base en descendant dans l'arbre de contenance jusqu'à une certaine profondeur.

L'entité de base du domaine est définie comme étant la racine de l'arbre de contenance à partir de laquelle la recherche doit commencer. Une demande détectée en visibilité doit toujours spécifier l'entité gérée de base du domaine. La profondeur du niveau de détection peut ensuite être spécifiée de l'une des quatre façons suivantes à l'intérieur de la demande détectée en visibilité:

1) selon l'entité de base;

2) selon les subordonnés du n^e niveau de l'entité de base;

3) selon l'entité de base et tous ses subordonnés jusqu'au $n^{ième}$ niveau inclus;

4) selon l'entité de base et tous ses subordonnés (c'est-à-dire l'ensemble du sous-arbre).

4.3.2 Filtrage

Les filtres permettent de spécifier les critères que les entités doivent observer afin qu'une opération de gestion puisse être exécutée. En association avec la détection de domaine de visibilité, le filtrage permet d'exécuter une même opération sur de multiples objets gérés au moyen d'une seule demande d'opération.

Un paramètre de filtrage est utilisé afin de déterminer s'il y a lieu ou non d'exécuter une opération sur un objet géré. Un paramètre de filtrage applique un test qui est passé correctement ou non par un objet géré particulier. Le filtre est exprimé en termes d'assertions sur la présence ou la valeur de certains attributs de l'objet géré. Le filtrage est correct si et seulement si son résultat est évalué à VRAI.

4.3.2.1 Règles d'homologie d'attribut

Les règles d'homologie suivantes sont définies pour usage dans les assertions de valeur d'attribut (AVA):

– **Egalité**: l'attribut est évalué à VRAI si et seulement si la valeur fournie dans l'assertion AVA est égale à la valeur de l'attribut.

  Pour les attributs à valeur d'ensemble, l'assertion AVA est évaluée à VRAI si et seulement si l'ensemble des membres fourni dans l'assertion AVA est égal à l'ensemble des membres contenu dans l'attribut.

– **Supérieur ou égal**: l'attribut est évalué à VRAI si et seulement si la valeur fournie dans l'assertion AVA est supérieure ou égale à la valeur de l'attribut.

Pour les attributs à valeur d'ensemble, la valeur contenue dans l'assertion AVA doit contenir exactement 1 membre. L'assertion AVA est évaluée à VRAI si et seulement si ce membre est supérieur ou égal à au moins un des membres contenus dans la valeur de l'attribut.

– **Inférieur ou égal**: l'attribut est évalué à VRAI si et seulement si la valeur fournie dans l'assertion AVA est inférieure ou égale à la valeur de l'attribut.

  Pour les attributs à valeur d'ensemble, la valeur contenue dans l'assertion AVA doit contenir exactement 1 membre. L'assertion AVA est évaluée à VRAI si et seulement si ce membre est inférieur ou égal à au moins un des membres contenus dans la valeur de l'attribut.

– **Présent**: l'attribut est évalué à VRAI si et seulement si un tel attribut est présent dans l'objet géré.

– **Sous-chaînes**: l'attribut est évalué à VRAI si et seulement si toutes les sous-chaînes spécifiées dans l'assertion AVA apparaissent dans l'attribut dans l'ordre indiqué sans superposition et sont séparées des extrémités de la valeur de l'attribut et les unes des autres par zéro, ou plus de zéro, éléments de chaîne. Par ailleurs, afin que l'assertion AVA soit évaluée à VRAI:

  • le premier élément de la sous-chaîne initiale doit, si présent, correspondre au premier élément de la valeur d'attribut;

  • les autres sous-chaînes doivent, si présentes, apparaître dans la valeur d'attribut selon l'ordre d'apparition des sous-chaînes dans l'assertion AVA; et

  • le dernier élément de la sous-chaîne finale doit, si présent, correspondre au dernier élément de la valeur d'attribut.

  Pour les attributs à valeur d'ensemble, chaque valeur de l'AVA doit contenir exactement 1 membre. L'AVA est évaluée à VRAI si et seulement si au moins un des membres de la valeur d'attribut contient toutes les sous-chaînes fournies dans l'assertion AVA comme décrit ci-dessus.

(Les trois autres tests de correspondance qui restent ne s'appliquent qu'aux attributs à valeur d'ensemble.)

– **Sous-ensemble de**: l'attribut est évalué à VRAI si et seulement si tous les membres déclarés par l'assertion sont présents dans l'attribut.

– **Sur-ensemble de**: l'attribut est évalué à VRAI si et seulement si tous les membres de l'attribut sont présents dans l'assertion AVA.

– **Intersection d'ensembles non nuls**: l'attribut est évalué à VRAI si et seulement si au moins un des membres déclarés dans l'assertion est présent dans l'attribut.

4.4 Notifications

Le cadre général doit pouvoir prendre en charge les capacités suivantes:

• remise de notifications;

• inscription à des types de notification;

• renvoi de notifications à des destinations multiples;

• filtrage des notifications;

• identification univoque de la ressource émettant la notification.

Le cadre général doit également prendre en charge les exigences relatives au contenu, à la relève et aux algorithmes de corrélation des notifications, qui sont indiquées dans UIT-T X.733 et dans UIT-T Q.821.

5 Aperçu général du cadre et exigences du protocole cadre

Le paragraphe précédent décrivait les fonctions de gestion de réseau que le cadre général doit prendre en charge. Le présent paragraphe et le reste de la Recommandation détaillent la façon dont les services RGT en architecture CORBA rempliront ces fonctions. Les aspects du cadre qui sont associés aux objets de modélisation sont inclus dans UIT-T X.780. L'on présentera d'abord une brève vue générale du cadre puis l'on définira quelques exigences du protocole cadre.

5.1 Aperçu général du cadre

Ce cadre général pour interfaces RGT en architecture CORBA est un ensemble de capacités dont un élément central est un ensemble de services communs aux objets CORBA. Le cadre définit le rôle de ces services dans les interfaces de gestion de réseau ainsi que les conventions relatives à leur utilisation. Le cadre définit également des services supports qui n'ont pas été normalisés en tant que services communs aux objets CORBA mais qui sont appelés à être normalisés dans les interfaces de gestion de réseau conformes à ce cadre. Les interfaces IDL pour ces services sont définies dans l'Annexe A ci-après.

Pour prendre en charge les objets logiciels représentant des ressources gérables, le cadre général exige que ces objets mettent en œuvre certaines capacités communes de base. Deux classes de base sont donc définies dans UIT-T X.780 pour usage lors de la modélisation de ressources de gestion de réseau. Les classes d'objets gérés (ou classes d'objets) contenues dans les modèles d'information doivent toujours hériter et réaliser un ensemble de base de capacités issues de ces classes afin de fonctionner à l'intérieur de ce cadre général. Finalement, certaines règles et conventions sont définies pour les développeurs de modèles d'information à utiliser avec le présent cadre général. Elles consistent en directives de modélisation, règles de conversion des modèles de directives GDMO pour protocole CMIP en définitions IDL d'architecture CORBA et en idiomes de type IDL. Toutes ces fonctions sont décrites graphiquement dans la Figure 1.

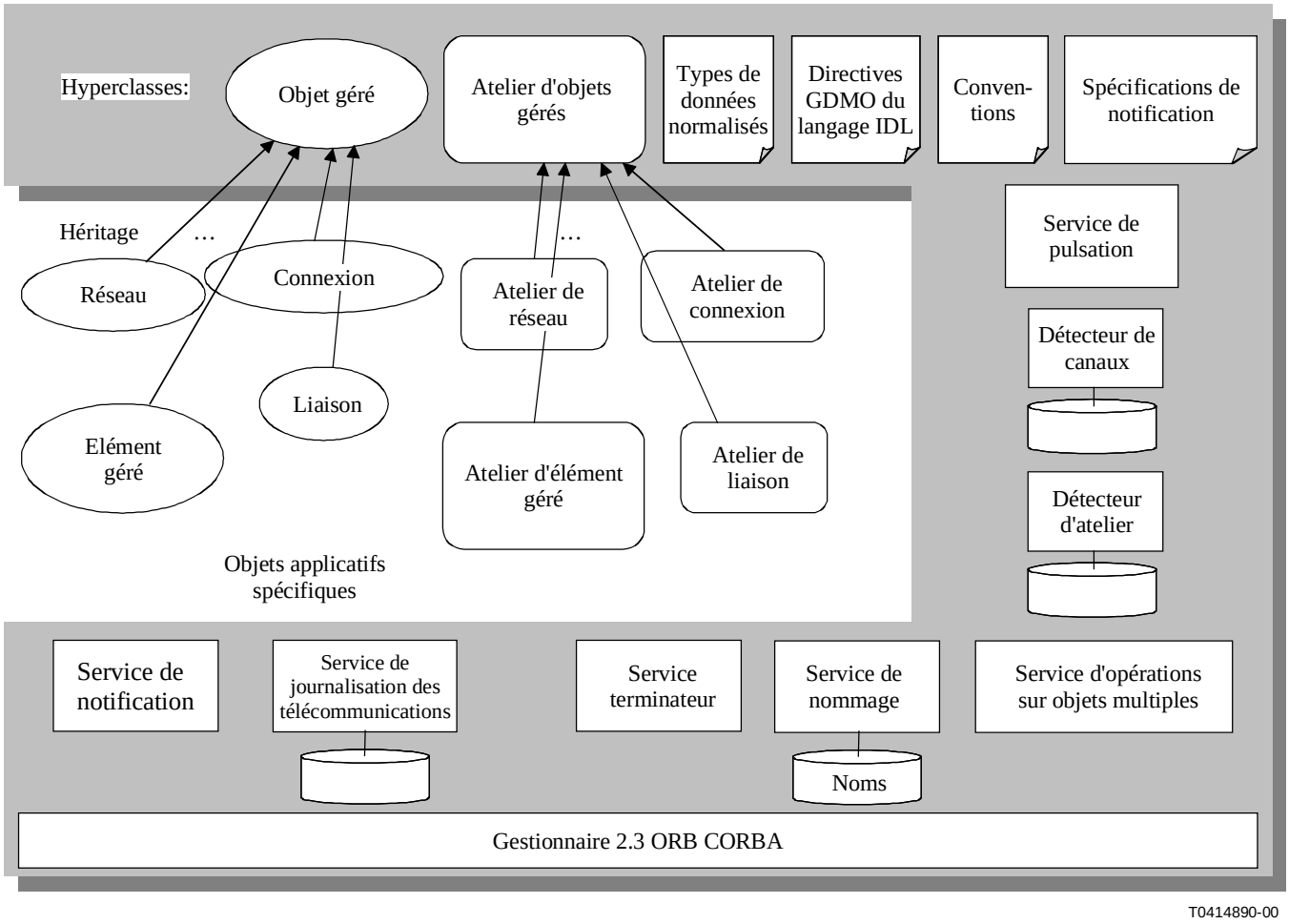

Figure 1/Q.816 – Aperçu général du cadre

Dans la Figure 1, le cadre est représenté en gris. Au centre se trouvent les objets applicatifs spécifiques qui sont pris en charge par le cadre. Dans la rangée du bas, le cadre contient le gestionnaire ORB de l'architecture CORBA. Au-dessus, un certain nombre de cases contenant des noms représentent les services qui composent le cadre général. (Certaines cases contiennent des icônes décrivant les bases de données qu'elles devront conserver afin d'exécuter leurs fonctions.) En haut de la figure, des icônes représentent deux hyperclasses, l'une pour les objets gérés, l'autre pour les ateliers d'objets gérés. Chacun des objets gérés et des ateliers correspondants, pris en charge par ce cadre, doivent finalement hériter de ces deux hyperclasses respectives. La Figure 1 montre également des icônes de pages aux coins retournés, qui représentent des conventions normalisées de modélisation d'objet.

Les services du cadre qui sont représentés sous la forme de cases à coins quadrangulaires sont définis dans la présente Recommandation. Les hyperclasses, les notifications et les conventions de modélisation d'objets sont définies dans UIT-T X.780.

5.2 Exigences du protocole cadre

Ce paragraphe définit les versions des services requis pour prendre en charge le cadre. Les spécifications des services et protocole CORBA sont définies par le Groupe de gestion par objets (OMG). Le Tableau 1 montre quelle est la version de la spécification OMG applicable qui doit être prise en charge afin d'assurer la conformité au présent cadre général. Il indique également le paragraphe dans lequel des spécifications particulières sont définies pour le service. Une version

ultérieure d'un service, qui comprendra toutes les capacités requises de la version indiquée, sera conforme au présent cadre général.

Tableau 1/Q.816 – Versions du service CORBA

Service	Version	Paragraphe
ORB	2.3.1 [2]	5.2
Service de nommage	1.0 [3]	6.1
Service de notification	1.0 [4]	6.2
Service de journalisation des télécommunications	1.0 [5]	6.3
Messagerie asynchrone	(déterminée par le système client)	6.4
Sécurité (si requise)	Soit le "protocole IOP sécurisé" ou "l'interopérabilité SSL de sécurité CORBA" telle que définie en [6]	6.5
Service de transaction	1.1 [7]	6.6

Le choix de la version 2.3.1 pour les capacités ORB de base est important. La version 2.3 de l'architecture CORBA inclut la prise en charge de l'adaptateur d'objet portable (POA) ainsi que de la transmission des objets selon leur valeur. L'adaptateur POA est important pour le cadre général parce qu'il permet à des réalisations fondées sur ce cadre de parvenir à l'échelle de millions d'objets instanciés, ordre de grandeur requis pour les applications de gestion de réseau. Le cadre utilise également l'héritage des types de valeur (qui prend en charge le polymorphisme) afin de conserver la flexibilité mais en réduisant l'utilisation des types CORBA "any", qui peuvent être inefficaces et fastidieux pour les programmeurs.

Les services de nommage, de notification et de journalisation correspondent à toutes les versions initiales qui sont offertes par le groupe OMG.

La messagerie asynchrone ne relève en fait que du côté client. Un gestionnaire ORB possédant les capacités de messagerie asynchrone permet à un client d'utiliser en mode asynchrone des interfaces CORBA synchrones (qui provoqueraient normalement un blocage par le client). Cette capacité est essentielle pour les clients de type monotâche, qui ne peuvent se permettre d'effectuer un blocage au cours d'opérations de gestion de réseau. La disponibilité des capacités de messagerie asynchrone est importante pour le présent cadre général car elle dispense celui-ci de la nécessité de définir les interfaces synchrones comme les interfaces asynchrones. Les clients n'ont pas besoin de faire appel à un gestionnaire ORB avec messagerie asynchrone s'ils sont de type multitâche et peuvent donc se permettre d'effectuer un blocage au cours de communications synchrones dans l'architecture CORBA.

Si la sécurité est requise, ce cadre tolère l'utilisation de gestionnaires ORB utilisant le protocole SSL 3.0 pour la sécurité en attendant la disponibilité de produits prenant en charge la sécurité TLS et en attendant la migration correspondante de la sécurité CORBA par le groupe OMG: tant que la spécification OMG du service de sécurité CORBA ne fait pas référence à la sécurité TLS, le choix du protocole (éventuellement) assuré dans un produit devra faire l'objet d'une négociation entre fournisseurs et utilisateurs individuels. L'utilisation de l'un ou l'autre protocole (ou leur non-utilisation) est donc, pour le moment, conforme au présent cadre général.

6 Exigences cadres des services communs relatifs aux objets

Le gestionnaire ORB de l'architecture CORBA offre des capacités de base d'interaction entre objets [2]. Les capacités supplémentaires sont définies comme étant des "services communs aux objets" distincts. Le services communs aux objets CORBA sont des services d'usage général et indépendant des domaines qui sont essentiels pour la mise au point d'applications CORBA composées d'objets répartis. Ils constituent également les blocs de construction de base pour l'interopérabilité applicative. Ces services sont définis avec les interfaces des objets et peuvent être combinés de différentes manières ou utilisés de nombreuses façons dans différentes applications. Dans un domaine spécifique, les services communs aux objets CORBA peuvent servir à construire des ressources de niveau supérieur et des cadres d'objets pouvant interfonctionner entre de multiples environnements de plate-forme.

Un grand nombre de ces services communs aux objets CORBA ont déjà été mis en œuvre. Ils sont livrables sur le marché des produits logiciels courants. De même, les programmeurs travaillant dans de nombreuses industries sont susceptibles d'en avoir prochainement l'expérience. La réutilisation de ces services communs aux objets au lieu de la définition de nouveaux services uniquement pour l'industrie des télécommunications ou la remise en application de cette capacité dans un code propre à une application se traduira par une adoption plus rapide et plus rentable de l'architecture CORBA pour la gestion de réseau.

Les paragraphes ci-après spécifient des exigences relatives à l'utilisation des services communs aux objets CORBA pour assurer l'interopérabilité entre différents systèmes de gestion de réseau et pour préserver le contexte des télécommunications.

6.1 Service de nommage

Le service de nommage de l'OMG est un service d'annuaire de l'architecture CORBA, dont il constitue les "pages blanches" [3]. Il permet à un client de construire une association nom-objet dénommée *corrélation de noms* que d'autres clients peuvent ensuite utiliser pour trouver l'objet. (Les références d'objet CORBA sont binaires et d'usage difficile par des êtres humains.) Une corrélation de noms est toujours définie par rapport à un *contexte de nommage*. Ce contexte est un objet qui contient un ensemble de corrélations de noms dans lesquelles chaque nom est localement unique. Une corrélation de noms est une structure de données contenant deux chaînes et une référence d'objet (adresse). La chaîne *ID* est l'identificateur de la corrélation. Une deuxième chaîne, appelée *kind (sorte)*, fait également partie de la structure de données. Ensemble, l'identificateur et la sorte désignent de façon univoque un objet relatif à un contexte. Différents noms peuvent être liés en même temps à un objet dans le même contexte ou dans des contextes différents. Le contexte de nommage peut aussi être lié à un nom contenu dans un autre contexte de nommage. Le fait de lier des contextes à d'autres contextes crée un *graphe de nommage*, qui est un graphe orienté comportant des bordures étiquetées et des nœuds qui représentent des contextes. En fonction d'un contexte contenu dans un graphe de nommage, une séquence de composants nominatifs (paires *ID-sorte*) peut faire référence à un objet. Cette séquence de structures, appelée *nom composé*, définit un trajet dans le graphe de nommage qui peut faire l'objet d'une navigation afin de résoudre le nom et de trouver l'objet.

Il n'est pas exigé que les corrélations de noms CORBA représentent une relation de contenance entre objets mais le concept de contenance est important en gestion de réseau et doit être communiqué de part et d'autre des interfaces de gestion de réseau. Le service de nommage CORBA est le meilleur moyen d'effectuer cette communication. Les alinéas suivants définissent une série d'exigences relatives à l'utilisation du service de nommage CORBA afin de représenter les relations de contenance entre instances d'objets gérés.

(R) NAME-1 Chaque objet géré doit avoir un nom (DN) et un seul. Les composants du nom peuvent être obtenus de multiples serveurs fédérés. Bien que le service de nommage OMG prenne en charge les noms multiples d'un même objet, le présent cadre général limite un objet géré à

l'utilisation d'un seul nom. La prise en charge de noms multiples est hors du domaine d'application du présent cadre général.

(R) NAME-2 Etant donné qu'une simple corrélation de noms ne peut pas identifier un objet, contenu ou non, chaque objet géré doit en fait posséder un contexte de nommage propre. Une corrélation de noms spéciale reliera, dans chacun de ces contextes, la valeur *ID* "d'objet" à une référence visant l'objet géré proprement dit. (Le champ *kind (sorte)* de cette corrélation sera vide.) D'autres contextes de nommage, représentant des objets gérés contenus, pourront également être associés à des noms dans ce contexte.

(R) NAME-3 Le champ *ID* d'une corrélation de noms pour un contexte de nommage représentant un objet géré sera dépendant de l'application et pourra en fait avoir une valeur sémantique allant au-delà de l'identification univoque d'un objet géré, dans une classe d'objets particulière. Par exemple, une valeur *ID* de "7" pour un objet de maintenance d'équipement, représentant une alvéole dans un châssis, peut indiquer que cet objet représente la 7e alvéole du châssis. Une valeur sémantique spécialement rattachée aux identificateurs sera documentée pour chaque classe d'objets gérés dans le cadre de la spécification d'interface d'objets gérés. Noter que le champ *ID* est une chaîne.

(R) NAME-4 Le champ *kind (sorte)* d'une corrélation de noms pour un contexte de nommage représentant un objet géré doit être déterminé par *des informations de corrélation de noms d'objets gérés*. Il s'agit d'informations définies sous la forme de constantes dans des modules de langage IDL afin spécifiquement de représenter d'éventuelles relations de contenance. On trouvera dans UIT-T X.780 des détails sur la représentation des informations de corrélation de noms d'objets gérés. En bref, un module de corrélation de noms contiendra cependant une chaîne constante nommée "sorte" qui servira de valeur dans le champ *kind (sorte)* des corrélations de noms CORBA. La valeur de cette chaîne sera habituellement le nom de classe non détecté de l'objet géré. Cela présente l'intérêt supplémentaire de faciliter l'identification du type d'un objet et de diminuer la probabilité de collisions entre noms. Un facteur de complication de ce processus est la parution de nouvelles versions d'un objet, par exemple un objet *equipmentR1* complétant un objet *equipment*. Lorsque la nouvelle classe développe seulement les capacités d'une classe existante sans modifier sa finalité (c'est-à-dire qu'elle continue à représenter la même ressource gérée), le champ *kind (sorte)* sera habituellement le nom original de la classe de base. Mais cela relève finalement du modéliseur d'objet qui définit le module IDL de corrélation de noms. L'utilisation de la valeur originale de la classe de base permettra aux applications existantes de continuer à utiliser la nouvelle classe comme si l'ancienne version était toujours applicable.

La Figure 2 donne un exemple de corrélations de noms conformément aux exigences ci-dessus. Dans cette figure, les contextes de nommage CORBA sont représentés comme des dossiers dont le contenu est constitué par les corrélations de noms. La convention de représentation d'un composant nominatif est une chaîne de format <ID>.<sorte>. (Certaines corrélations de noms de l'exemple ne possèdent pas de pointeur représenté dans le schéma afin de réduire la complexité de celui-ci.) Le graphe représente un objet Réseau nommé "CentralNet", qui contient un objet Elément géré nommé "Element9" et un objet Connexion nommé "R5698".

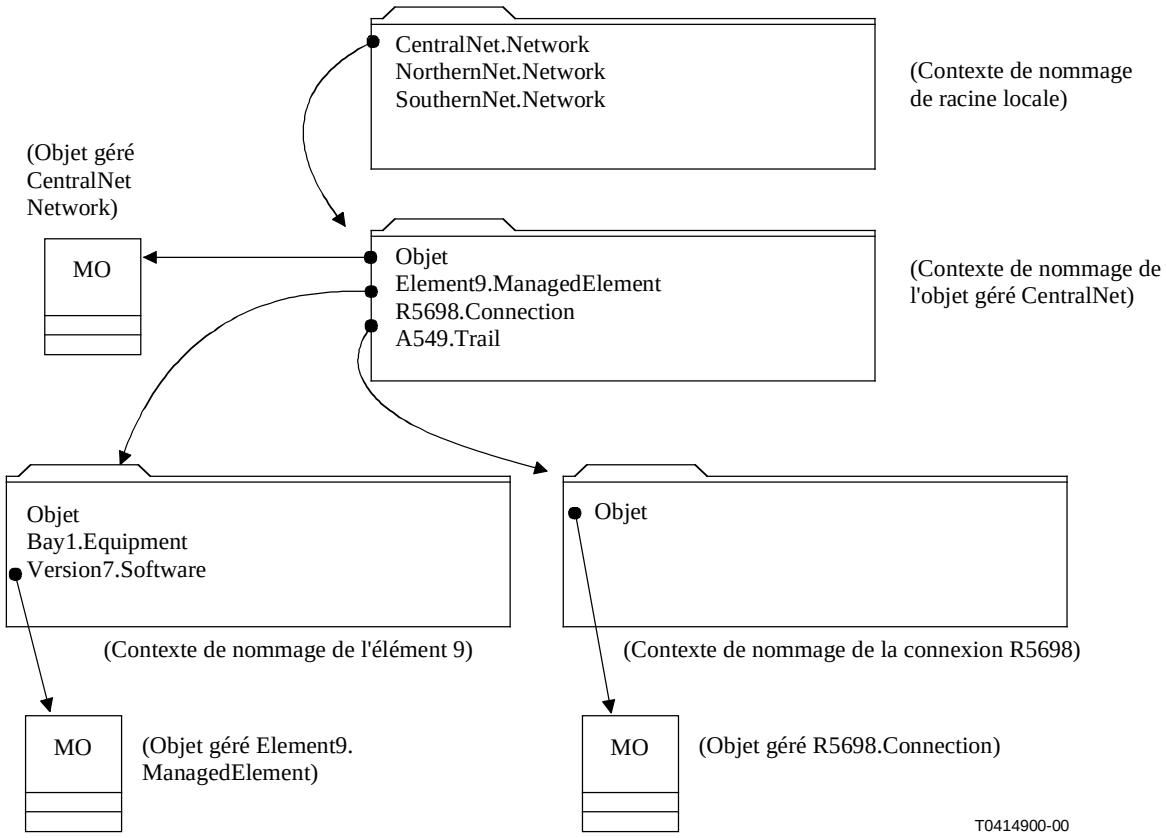

Figure 2/Q.816 – Graphe de nommage d'objets gérés

(R) NAME-5 Chaque système géré doit fournir au moins un contexte de nommage de racine locale. On notera que, dans la figure ci-dessus, le contexte de nommage situé le plus haut est appelé "contexte de nommage de racine locale": il s'agit du contexte de nommage dans lequel les noms des objets gérés situés le plus haut dans le système seront corrélés, ainsi que les noms de certains objets de service support.

Un système géré peut avoir plusieurs contextes de nommage de racine locale. Etant donné que les objets gérés ne peuvent pas avoir de noms multiples, ces objets peuvent être corrélés au-dessous d'une seule racine locale. Les objets de service support peuvent cependant avoir des noms corrélés sous de multiples contextes de nommage de racine dans le même système. Un facteur à considérer lors de la détermination du nombre de contextes de nommage de racine locale qu'un système géré peut avoir est la possibilité que certains des objets gérés puissent parfois devoir être transférés dans un autre système. Le transfert d'un arbre entier d'objets gérés, y compris le contexte de nommage de racine locale, sera plus simple que le transfert d'un sous-arbre d'objets.

(R) NAME-6 Un système géré doit offrir une procédure administrative locale pour attribuer un nom CORBA à chaque contexte de nommage de racine locale dans le système. Tous les noms échangés de part et d'autre de l'interface gérée comprendront, sauf indication contraire, le nom du contexte de racine locale, ce qui inclut les paramètres d'exploitation et les notifications.

Cette caractéristique vise à permettre à une administration de rendre des noms mondialement uniques. Etant donné que le système géré doit toujours faire en sorte que tous les noms soient uniques par rapport au contexte de nommage de la racine locale, une administration peut, à cette fin, attribuer un nom mondialement unique au contexte de nommage de la racine locale. Le mécanisme utilisé pour choisir un nom mondialement unique pour le contexte de nommage de racine locale relève de l'administration. Le format du nom sera celui du service de nommage CORBA, c'est-à-dire *CosNaming::Name*. De multiples composants sont autorisés mais les administrations préféreront sans doute des noms courts dans le contexte de racine locale afin de réduire le surdébit.

L'attribution d'un nom au contexte de nommage de la racine locale, en plus de rendre les noms uniques, facilite la résolution des noms par un système gérant car celui-ci peut associer les contextes de nommage de racine locale pour tous les systèmes qu'il gère afin de les introduire dans son propre service de nommage local. Le nom qu'il utilise pour cette association sera celui qui est attribué au contexte de nommage radical du système géré. On en trouvera un exemple à la Figure 3.

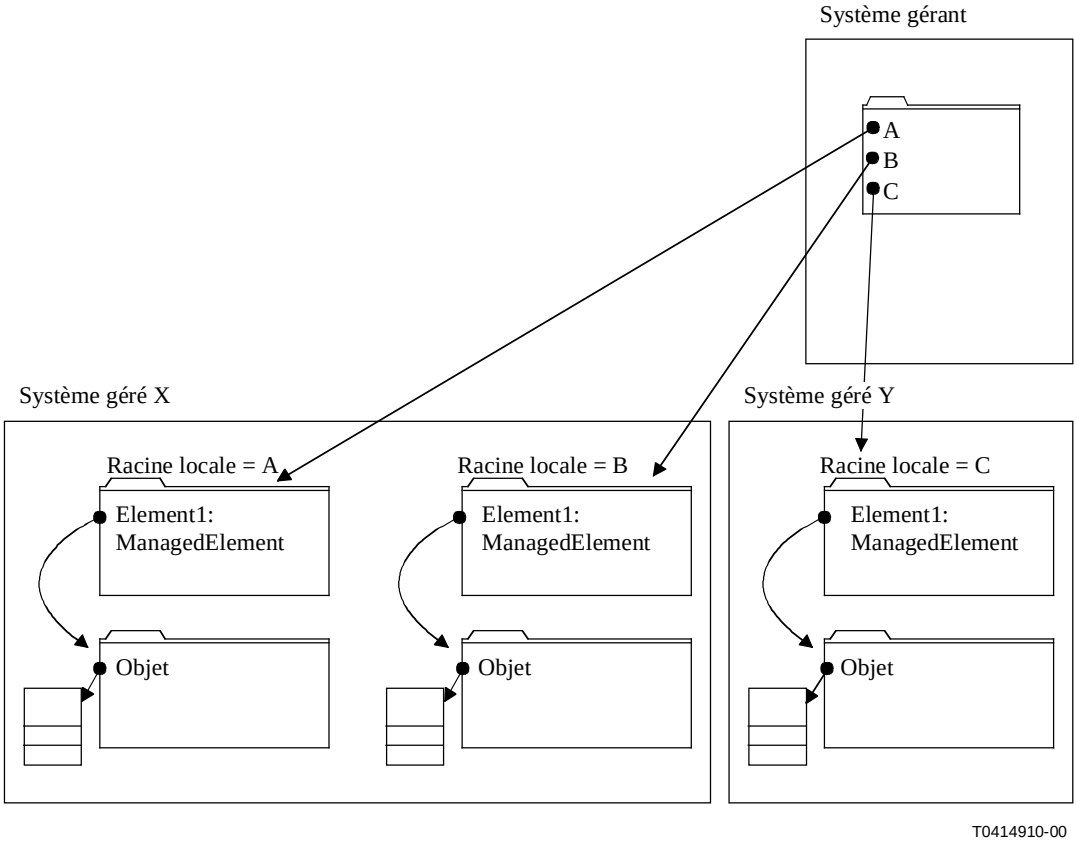

Figure 3/Q.816 – Attribution de noms à des contextes de nommage radicaux

La Figure 3 montre deux systèmes de gestion d'élément de base: le système "X", qui possède deux objets de type *ManagedElement* et le système "Y", qui en possède 1. Chaque objet de type *ManagedElement* appartient à son propre contexte de nommage de racine locale. Autrement dit, le système X possède deux racines locales et le système Y une seule. Il y a également un système de gestion de réseau et les contextes de racine locale des deux systèmes EMS ont été associés au service de nommage de ce système. Cette administration a choisi d'attribuer les noms uniques "A" et "B" aux contextes de racine locale du système X et le nom "C" au contexte de racine locale du système Y. Les références aux contextes de nommage de racine locale ont été associées à ces noms dans le système de gestion de réseau.

Admettons que le système Y émet une notification concernant son objet *ManagedElement*. Le nom complet de cet objet (contenu dans la notification) sera "C/Element1.ManagedElement". Admettons ensuite que le système NMS souhaite extraire d'autres données de cet objet. A cette fin, il devra résoudre le nom afin de l'introduire dans une référence d'objet CORBA. Le système NMS peut remplir cette tâche en effectuant simplement une opération de résolution au moyen du nom complet contenu dans le contexte local auquel les contextes de racine locale du système EMS sont associés. Etant donné que le service de nommage du système NMS est fédéré avec les services de nommage du système EMS, ce service peut faire suivre automatiquement l'opération de résolution au service

de nommage du système EMS approprié puis renvoyer à l'application NMS la référence d'objet correspondante.

L'on s'attend que le nom contenu dans le contexte de nommage de la racine locale sera attribué au cours de l'initialisation d'un nouveau système. Une fois l'opération engagée, il sera très difficile sinon impossible d'effectuer un changement.

La référence d'objet interopérable (IOR) CORBA du contexte de la racine locale doit, une fois qu'un nom lui a été attribué, être associée à un contexte de nommage du système gérant car jusqu'à ce point elle n'est pas informée de l'existence du nouveau système. Autrement dit, le système géré devra également permettre l'accès à la référence IOR "concaténée" du contexte de nommage de la racine locale. Cette valeur sera ensuite transférée au système gérant par une autre voie que l'interface de gestion (courrier électronique, transfert de fichier, etc.). Le système gérant devra pouvoir accepter cette référence IOR concaténée puis l'associer à un nom contenu dans ce système. Dès que la référence IOR du contexte de racine locale est associée à un nom dans le système gérant, celui-ci peut commencer à rechercher les objets du nouveau système (au moyen du service d'opération sur objets multiples décrit ci-après) puis commencer à les gérer.

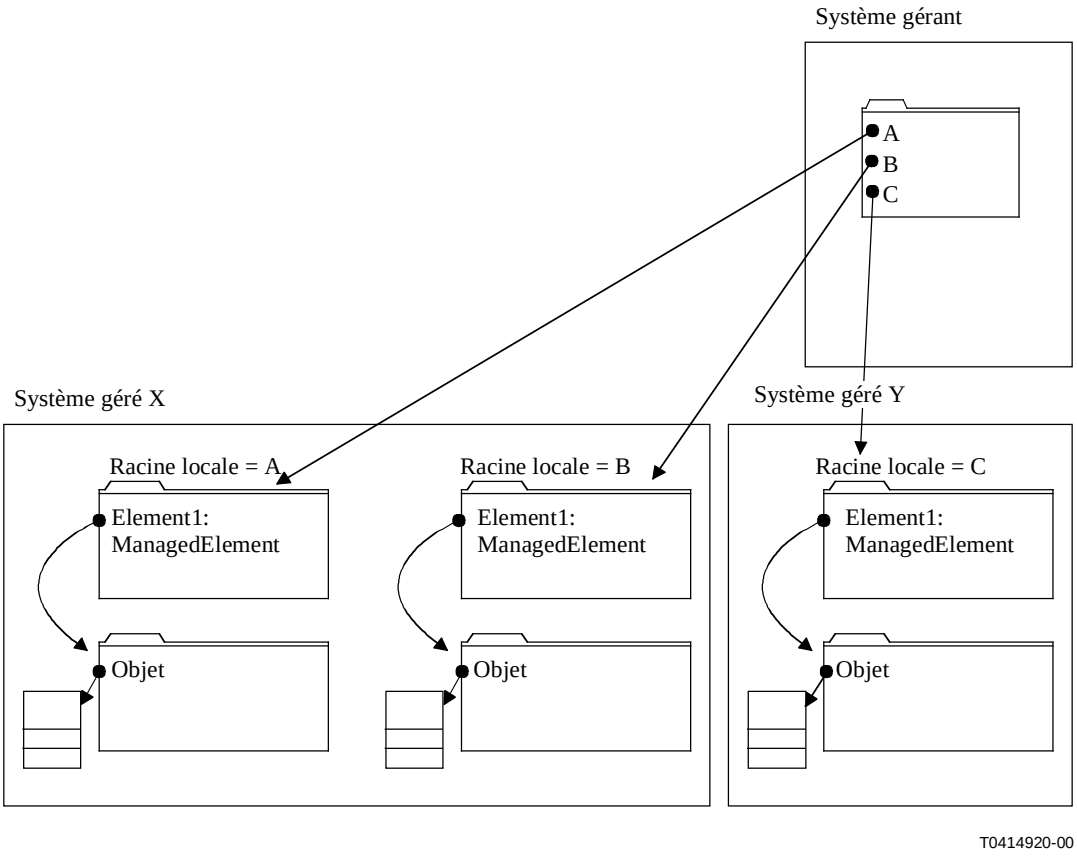

Figure 4/Q.816 – Transfert d'un contexte de nommage radical et des objets qu'il contient

La Figure 4 montre comment l'on peut transférer vers un autre système un contexte de nommage de racine locale et tous les objets qui y sont contenus hiérarchiquement, sans modifier les noms de ces objets. Le seul changement qui pourra être nécessaire sera celui de la référence d'objet associée au nom dans le ou les systèmes de gestion de réseau. Par ailleurs, il faudra mettre à jour toutes les références pouvant encore viser des objets transférés. Le transfert de la seule partie d'un arbre contenu hiérarchiquement par un contexte de nommage de racine locale nécessitera le renommage de ces objets.

6.1.1 Conversion en chaînes des noms d'objet géré

Dans le présent cadre général, les noms d'objet géré sont représentés par des structures de données. Il peut cependant être parfois nécessaire de représenter des noms d'objet géré sous forme de chaînes. La spécification du service de nommage interopérable [Ref] de l'architecture CORBA définit des règles de conversion en chaînes des noms du service de nommage CORBA (utilisés dans le présent cadre général). Les systèmes de gestion de réseau pourront cependant juger utile de mémoriser des noms d'objet géré dans des serveurs en protocole léger d'accès à l'annuaire (LDAP, *lightweight directory access protocol*). Le présent paragraphe définit les règles de conversions des noms d'objet géré en chaînes pouvant être utilisées avec le protocole LDAP.

Un nom d'objet géré est une structure de données contenant des séquences de données structurées en deux chaînes nommées *kind* (sorte) et *ID* (identificateur). Les noms d'objet géré sont convertis en chaînes de nom distinctif par concaténation des chaînes provenant de chacune des données structurées de la séquence, assorties de signes d'égalité (=) et de virgules (,) dans l'ordre indiqué ci-dessous:

Chaîne DN = "<*kind*>$_0$=<*ID*>$_0$,<*kind*>$_1$=<*ID*>$_1$,<*kind*>$_2$=<*ID*>$_2$…"

Le terme <*kind*>$_0$ se rapporte à la valeur de la chaîne *kind* dans le premier élément de la séquence. Le terme <*kind*>$_1$ se rapporte à la valeur de la chaîne *kind* dans le deuxième élément de la séquence, etc. Lorsqu'elles sont mises sous forme de chaînes de noms distinctifs LDAP, les chaînes *kind* et *ID* peuvent avoir à contenir des séquences d'échappement. Pour les détails, voir [12].

6.2 Service de notification

Le service de notification CORBA assure l'échange asynchrone de messages événementiels entre clients utilisant un paradigme de type inscription-publication [4]. Le service de notification présente des canaux d'événements qui arbitrent des messages événementiels, des fournisseurs de notification qui remettent des messages événementiels et des consommateurs de notification qui consomment des messages événementiels. Le service de notification CORBA conserve toute la sémantique spécifiée pour le service d'événement CORBA et permet la rétrocompatibilité avec les clients du service d'événements. La capacité étendue qui est importante pour le domaine de gestion de réseau comporte le service d'événement structuré, le service de filtrage d'événements et la QS (qualité de service). La Figure 5 ci-dessous décrit l'architecture générale du service de notification.

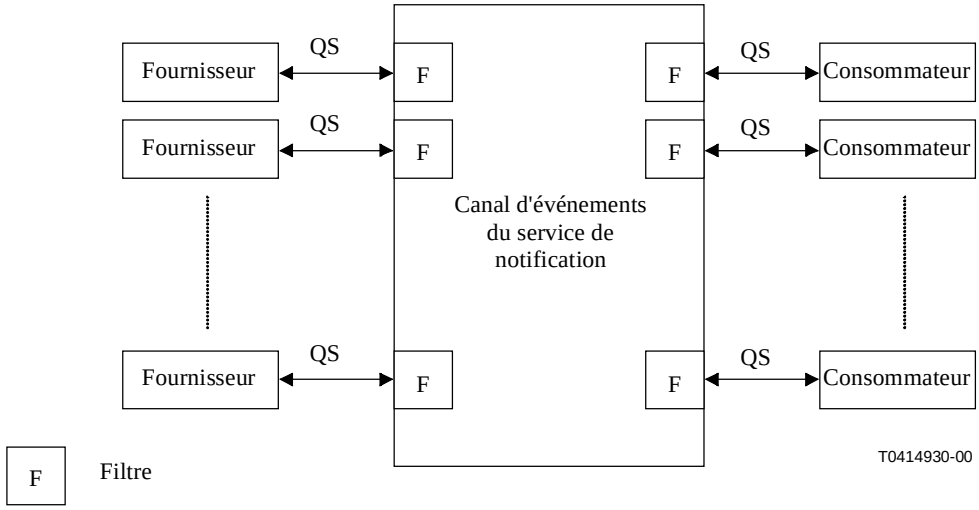

Figure 5/Q.816 – Architecture du service de notification

(R) NOTIF-1 Le service de notification doit prendre en charge le modèle d'interface de distribution sélective. L'interface d'objet géré avec le canal d'événements doit être un fournisseur de distribution sélective.

(R) NOTIF-2 Le système géré doit instancier l'objet (les objets) de canal d'événements qu'il utilisera. Un système géré doit toujours instancier au moins un canal et peut en instancier plusieurs. (Ces canaux peuvent être soit des canaux OMG d'événements de notification [4] soit des canaux OMG d'événements de journalisation des télécommunications [5] ou un objet prenant en charge l'interface NotificationIRPOperation du groupe 3GPP [13].) Le cadre général ne prend pas en charge la création ou la suppression de canaux d'événements de part et d'autre de l'interface de gestion. Des procédures administratives locales peuvent être fournies à cette fin. (Les canaux d'événements prennent bien en charge, cependant, la création et la suppression de filtres de part et d'autre de l'interface de gestion.)

(R) NOTIF-3 Chaque canal d'événements doit être enregistré dans le service détecteur de canaux, qui est un service support défini au § 7.2 du présent cadre général. Au cours de cet enregistrement, le canal doit être associé à un ou plusieurs objets gérés qui forment chacun la base d'un arbre d'objets gérés expédiant leurs événements vers le canal. Plusieurs canaux peuvent être associés au même objet géré de base. Un usage probable de cette notification est d'avoir différents canaux pour différents types d'événement. Par exemple, un canal peut traiter les événements de gestion de la qualité de fonctionnement tandis qu'un autre canal traitera des alarmes. Lorsque le canal est enregistré dans le service détecteur de canaux, il est également lié à un ensemble de types d'événements qu'il traite et à un ensemble de types d'objets gérés qui lui envoient leurs événements. Chaque notification de chaque objet géré doit aller vers au moins un canal.

Bien que cette méthode soit tout à fait flexible et permette des dispositions complexes des canaux, cette complexité dépend de la réalisation du système géré car les canaux ne peuvent pas être créés de part et d'autre de l'interface de gestion. La disposition peut se réduire à un seul canal surveillant tous les objets gérés du système. (Prière de noter encore que si les canaux ne peuvent être créés de part et d'autre de l'interface, les canaux individuels prennent bien en charge la création et la suppression de filtres de part et d'autre de l'interface de gestion: un nombre quelconque de clients peut donc être enregistré pour les événements que ces clients souhaitent recevoir.)

(R) NOTIF-4 Le service de notification doit prendre en charge les événements structurés.

(O) NOTIF-5 L'utilisation de séquences d'événements structurés est facultative. Les séquences d'événements structurés sont définies en [4]. Elles servent à envoyer plusieurs événements dans un même message.

(O) NOTIF-6 L'utilisation d'événements typés est facultative.

NOTE – Si le système géré prend en charge les événements typés, il doit toujours permettre aux systèmes gérants de recevoir les événements structurés, si le gestionnaire en décide ainsi. A cette fin, l'on peut utiliser un canal d'événement de notification assurant la conversion des événements typés en événements structurés comme défini dans la spécification du service de notification de l'OMG [4].

L'interface de messagerie entre fournisseurs et consommateurs doit être définie en langage IDL comme s'ils utilisaient toujours des événements typés, ce qui permet de capturer la notification en langage IDL (ce qui est impossible pour des événements structurés à moins qu'ils ne soient assortis de commentaires) ainsi que de prendre en charge les notifications typées pour les applications qui souhaitent les utiliser.

On trouvera ci-après les règles de création des notifications structurées sur la base de ces opérations typées.

La définition du service de notification OMG définit effectivement des règles afin que les canaux convertissent automatiquement les notifications typées en notifications structurées. Si les objets gérés créent par défaut des notifications typées mais que le client souhaite recevoir des notifications structurées, les règles de conversion de l'OMG doivent être suivies par le canal. Noter cependant que

cette disposition est sans doute moins efficace que l'utilisation d'événements typés par les deux systèmes. Si les objets gérés créent par défaut des notifications structurées, ils doivent le faire conformément aux règles ci-après.

Les notifications structurées qui sont créées par défaut par un système géré différeront légèrement des notifications structurées qui sont créées par conversion automatique à partir de notifications typées, cela afin qu'un système gérant puisse signaler cette différence et puisse accepter des notifications typées si celles-ci sont prises en charge par le système géré. Une autre raison en est l'utilisation plus efficace des notifications structurées. Les systèmes gérés qui créent par défaut des notifications structurées peuvent en exclure certains paramètres facultatifs. Comme une notification typée est créée à partir d'une invocation de méthode fortement typée, un canal de notification du commerce qui convertit cette invocation en notification structurée inclut d'éventuelles valeurs néant en tant que paires nom-valeur dans le corps de l'événement structuré au lieu d'exclure ces valeurs. Noter que le fait de permettre à des systèmes gérés, qui créent par défaut des notifications structurées, d'exclure des paramètres facultatifs rend improbable que les canaux de notification du commerce soient en mesure de prendre en charge la conversion automatique d'événements structurés en événements typés.

En résumé, si les objets gérés créent par défaut des événements structurés, ils doivent le faire conformément aux règles ci-après. Etant donné que, pour des raisons d'efficacité, les présentes règles permettent à des systèmes gérés d'exclure des paramètres facultatifs de notifications structurées, l'on ne s'attend pas à la prise en charge de la conversion automatique de ces notifications structurées en notifications typées par des canaux de notification du commerce. Le système gérant doit donc toujours accepter les événements structurés. Si le système géré crée par défaut des événements typés, le système gérant peut se fonder sur le canal de notification pour les convertir automatiquement en événements structurés conformément aux règles du service de notification de l'OMG. Les notifications structurées se fondent sur une utilisation importante des types de données CORBA "any", ce qui peut cependant être inefficace. Dans ce cas, le système gérant préférera probablement accepter des notifications typées.

(R) NOTIF-7 Les fournisseurs et les consommateurs d'événements structurés doivent suivre ces règles pour construire et recevoir ces événements. (Voir dans la Figure 6 la description de la structure de notification et la façon dont les éléments issus de la notification IDL doivent y être appliqués.)

- La chaîne `domain_type` de l'en-tête fixe de l'événement structuré doit être mise à la valeur "télécommunications".
- La chaîne `type_name` de l'en-tête fixe de l'événement structuré doit être mise à la valeur du nom détecté de l'opération définissant la notification en langage IDL. Par exemple: "`itut_x780::Notifications::attributeValueChange`".
- La chaîne `event_name` de l'en-tête fixe de l'événement structuré n'est pas utilisée par leprésent cadre général.
- Les champs facultatifs de l'en-tête peuvent, le cas échéant, être inclus afin de prendre en charge des caractéristiques comme la qualité de service.
- Chaque paramètre de l'opération doit être placé dans une paire nom-valeur de la partie corps filtrable de l'événement structuré. La chaîne `fd_name` de cette paire doit être mise au nom du paramètre et le type inséré dans la valeur `fd_value` associée sera celui qui est spécifié pour le paramètre. Si l'on prend comme exemple la notification `equipmentAlarm` issue du module IDL présenté plus loin, la première chaîne `fd_name` sera mise à "`eventTime`" et la première valeur `fd_value` contiendra un type de données `ExternalTimeType`. Bien que tous les paramètres de notification entrent dans le corps filtrable de la structure de notification, il peut être difficile ou impossible, selon le type de données du paramètre, de créer un filtre utile au moyen de ce paramètre. Les "règles d'homologie" du filtre sont fondées sur les capacités du canal.

- Les paramètres qui sont qualifiés de "facultatifs" peuvent éventuellement être exclus de la structure de notification. Si des notifications typées sont utilisées, ces paramètres sont inclus mais auront généralement une valeur néant spéciale s'ils ne sont pas pris en charge. Dans le cas des types pour lesquels il n'existe aucune valeur néant spéciale (comme les entiers) l'on définit habituellement un type spécial composé par réunion logique du type de base (comme un entier) et du type néant. Ces types obtenus par réunion d'ensembles peuvent être exclus des notifications structurées lorsqu'ils ont une valeur néant mais, s'ils sont inclus, le type doit être obtenu par réunion d'ensembles, ce qui permet d'utiliser les mêmes filtres pour les notifications structurées et pour les notifications typées.
- Le reste du corps de l'événement structuré (partie non filtrable) doit être vide.
- Les paramètres nommés "opération" doivent être évités dans les opérations de notification afin de pouvoir prendre en charge l'utilisation de notifications typées. (Lors de la conversion de notifications typées en notifications structurées, les paramètres d'une opération sont automatiquement placés par le canal d'événements dans une structure de notification. Malheureusement, les règles élaborées à cette fin indiquent que le nom de l'opération utilisée pour émettre la notification ne doit pas être inséré dans l'en-tête de l'événement mais dans le corps de la structure, en tant que première paire nom-valeur. La chaîne `fd_name` est mise à la valeur "opération" et la valeur `fd_value` est mise à une chaîne contenant le nom de l'opération. L'utilisation d'un paramètre nommé "opération" se traduirait en effet par une deuxième paire nom-valeur également nommée "opération", ce qui pourrait provoquer une confusion entre ces deux noms.)

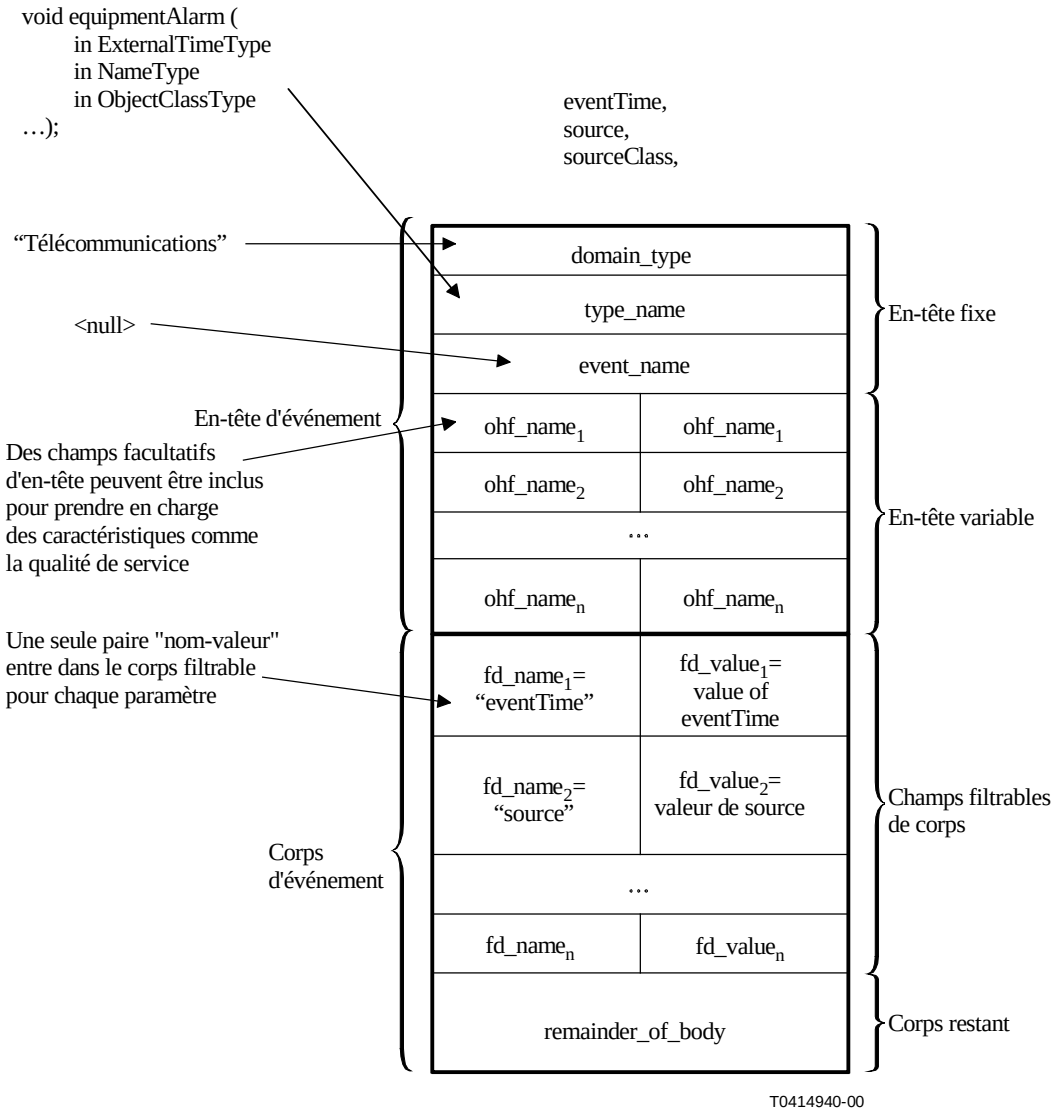

Figure 6/Q.816 – Application des notifications sur les événements structurés

(R) NOTIF-8 La spécification du service de notification prend en charge les expressions de filtrage qui sont utilisées afin de déterminer si l'événement doit être réexpédié. Elle prend également en charge les expressions de filtrage qui "appliquent" des valeurs de la notification sur des paramètres utilisés pour configurer le fonctionnement du canal d'événements, comme le paramètre QS utilisé lors de l'acheminement de l'événement. Par exemple, un filtre applicateur peut servir à appliquer un champ de valeur "sévérité=majeure" issu d'un événement (ce qui n'a pas de signification pour un canal d'événements) sur un paramètre QS "priorité=1" (ce qui a effectivement une signification pour un canal d'événements). Le service de notification doit assurer le filtrage des événements avec des objets de filtrage prenant en charge les contraintes exprimées dans la grammaire de contrainte par défaut spécifiée par le groupe OMG. Le service de notification doit également prendre en charge les filtres applicateurs.

(R) NOTIF-9 Le paramètre QS de fiabilité doit, dans le service de notification, prendre en charge les relations "EventReliability = Persistent & ConnectionReliability = Persistent".

Il est garanti que chaque événement sera acheminé vers tous les consommateurs inscrits à sa réception au moment où cet événement aura été remis au canal et dans le cadre des limites d'expiration. Si la connexion entre le canal et un consommateur est interrompue pour une raison ou une autre, le canal mémorisera continuellement les éventuels événements destinés à ce consommateur jusqu'à ce que chaque événement arrive à expiration en raison de ses limites de

temporisation ou jusqu'à ce que le consommateur redevienne disponible et que le canal soit donc en mesure d'acheminer les événements vers tous les consommateurs inscrits. Par ailleurs, dès la reprise sur défaillance, le canal de notification réétablira automatiquement les connexions vers tous les clients qui lui étaient connectés au moment de la défaillance [4].

(R) NOTIF-10 Le paramètre QS de la politique d'ordonnancement du service de notification doit permettre l'acheminement des événements dans l'ordre de leur arrivée, c'est-à-dire en mode FIFO. Le service de notification peut également (en option) prendre en charge un paramètre QS d'ordre de priorité dans lequel les événements peuvent être mis en mémoire tampon dans l'ordre de priorité, de façon que les événements de priorité supérieure soient acheminés avant les événements de priorité inférieure.

(R) NOTIF-11 La réalisation du service de notification déployé doit observer les règles de conformité de la spécification OMG du service de notification, à l'exception du modèle d'interface d'extraction sélective.

6.3 Service de journalisation des télécommunications

Le service CORBA de journalisation des télécommunications [5] est un service de journalisation en architecture CORBA qui prend entièrement en charge UIT-T X.735. La journalisation est réalisée en tant que canal d'événements du service d'événements ou du service de notification. Le service de journalisation prend en charge les capacités suivantes:

- rédaction du journal: les événements fournis au journal sont continuellement mémorisés sous la forme de journaux d'exploitation;
- réexpédition à partir du journal: les événements journalisés sont également réexpédiés vers d'autres journaux ou vers d'éventuelles applications souhaitant les recevoir;
- événements de journalisation: le journal produit lui-même des événements.

Le service de journalisation fournit également des fonctions de commande et de gestion des journaux, de manipulation des journaux et de gestion de la durée de vie des journaux. La Figure 7 ci-après donne une représentation graphique du service de journalisation.

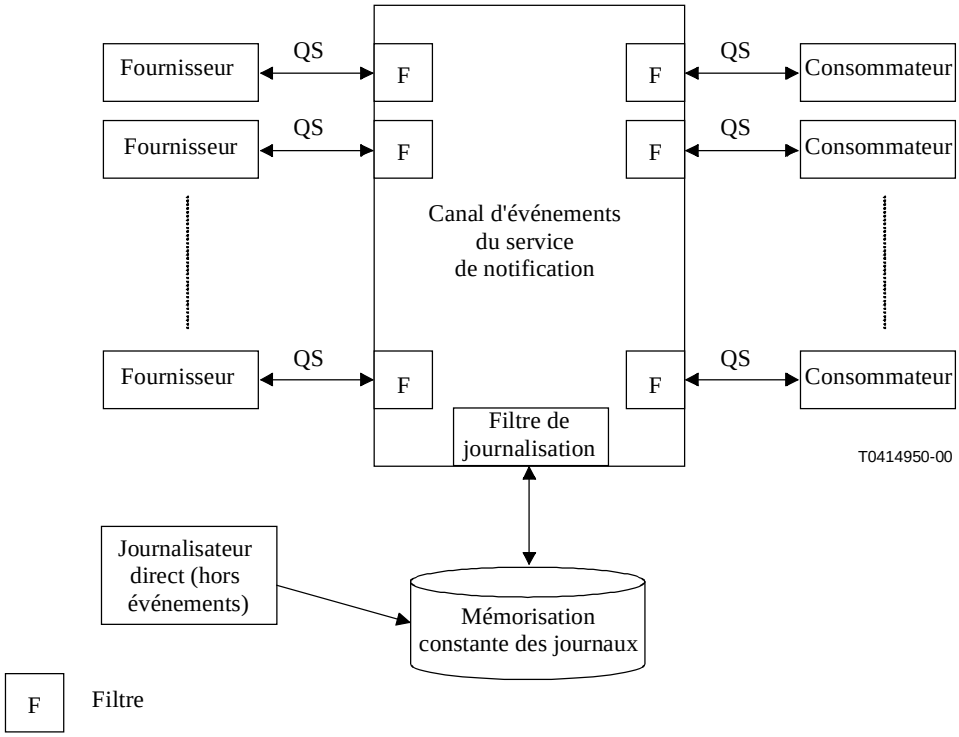

Figure 7/Q.816 – Service de journalisation des télécommunications

En manipulant le filtre de journalisation, un système gérant est en mesure de déterminer les événements qui seront journalisés et ceux qui ne le seront pas, exactement de la même façon qu'il peut déterminer les événements qui seront réexpédiés et ceux qui ne le seront pas. La seule exception est le "journalisateur direct", qui est une application inscrivant directement les données dans le journal.

On notera que la définition du service OMG de journalisation des télécommunications est antérieure à la rédaction du présent cadre général. Les notifications issues du journal des télécommunications ont une structure différente des autres notifications de ce cadre général, bien que certains noms des notifications et des paramètres soient sémantiquement identiques.

(R) LOG-1 Le service de journalisation doit prendre en charge toutes les exigences du service de notification. Les canaux d'événements de journalisation doivent toujours être inscrits au service détecteur de canaux.

(R) LOG-2 Le journal pris en charge par le service de journalisation doit être du type `struct LogRecord` normal. La prise en charge du type `struct TypedLogRecord` est facultative.

(R) LOG-3 La mise en œuvre du service de journalisation doit observer la règle de conformité contenue dans la spécification OMG du service de journalisation des télécommunications, à l'exception du modèle d'interface d'extraction sélective.

6.4 Service de messagerie

Le service de messagerie CORBA couvre trois secteurs: l'invocation de méthode asynchrone (AMI), l'invocation indépendante du temps (TII) et la qualité du service (QS) de messagerie [8]. De ces trois secteurs, l'invocation AMI joue un rôle déterminant dans le domaine de la gestion de réseau car elle permet aux clients de formuler des requêtes d'objet CORBA non bloquantes.

On notera que l'architecture CORBA est conçue de façon qu'une application puisse invoquer une méthode relative à un objet comme si celui-ci était local dans l'application, quel que soit

l'emplacement réel de cet objet. Normalement, lorsqu'une méthode est invoquée au sujet d'un objet dans une application non répartie, la commande passe à cet objet et la routine d'appel effectue un blocage jusqu'à ce que la méthode soit complète et que la commande soit rétrocédée. Ces règles sémantiques sont maintenues en architecture CORBA. Dans une application répartie, la latence du réseau peut toutefois se traduire par une mauvaise qualité. Il y a cinq solutions possibles à envisager:

1) les applications pourraient simplement s'accommoder des retards;

2) les interfaces RGT pourraient par contre être définies en langage IDL comme étant asynchrones, c'est-à-dire que les opérations de gestion y seraient toujours définies de façon à ne pas renvoyer de résultats. Les invocations pourront alors être formulées sans blocage (ce qui est compatible avec l'architecture CORBA). Les résultats seront renvoyés ultérieurement, lorsque le système géré effectuera un "rappel" auprès du système gérant;

3) les interfaces RGT définies en langage IDL possèderaient toujours deux ensembles d'opérations: l'un asynchrone, l'autre synchrone;

4) les interfaces RGT seraient définies en langage IDL comme étant synchrones: les applications de gestion amélioreraient la qualité de fonctionnement du fait qu'elles seraient multitâches et capables de bloquer en cas de multiples requêtes en instance tout en continuant à traiter d'autres tâches;

5) certaines interfaces RGT seraient définies en langage IDL comme étant synchrones: les applications de gestion amélioreraient la qualité de fonctionnement du fait qu'elles utiliseraient le service d'invocation de méthode asynchrone et un gestionnaire ORB qui le prend en charge.

Le présent cadre choisit une combinaison des solutions 4) et 5). Des interfaces RGT sont définies en langage IDL comme étant synchrones. Les applications de gestionnaire qui sont multitâches peuvent utiliser directement ces interfaces et bénéficier d'une bonne qualité de fonctionnement. Les applications de gestionnaire qui ne peuvent pas être multitâches doivent utiliser le service d'invocation AMI afin d'améliorer la qualité de fonctionnement. Etant donné que les gestionnaires multitâches n'ont pas besoin du service d'invocation AMI, son utilisation est facultative.

L'invocation AMI n'est manipulée que comme un élément d'application linguistique du côté client. Dans la plupart des cas, les réalisations du côté serveur n'ont pas besoin d'être modifiées. Dans certaines situations, comme dans le cas d'un serveur transactionnel, l'asynchronisme d'un client pose effectivement un problème et nécessite des changements du côté serveur si celui-ci est censé traiter des requêtes transactionnelles asynchrones. Les requêtes asynchrones ne seront cependant pas étudiées dans la présente Recommandation. La Figure 8 suivante décrit le concept fondamental du modèle OMG d'invocation AMI.

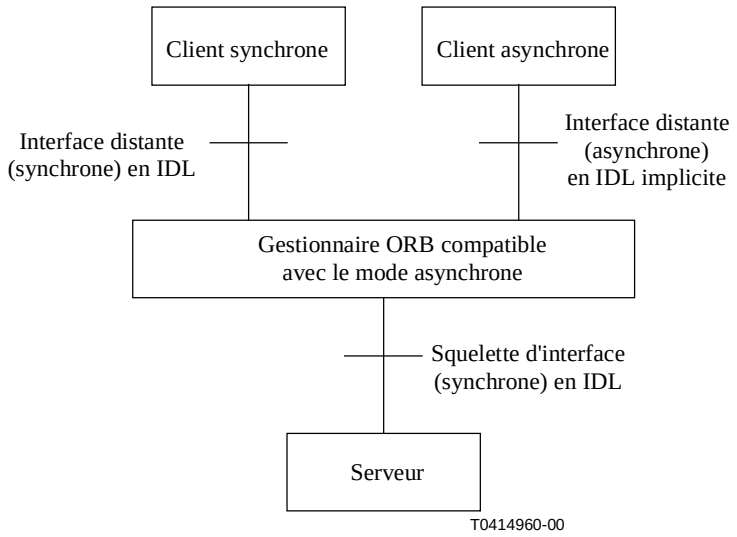

Figure 8/Q.816 – Gestionnaire ORB compatible avec le mode asynchrone

La spécification d'invocation AMI offre deux modèles de requêtes asynchrones: la *requête à rappel* et la *requête à interrogation*. Dans le modèle de *requête à rappel*, le client transmet une référence d'objet de type `ReplyHandler` sous la forme d'un paramètre lorsqu'il invoque une opération asynchrone bilatérale auprès d'un serveur. Lorsque celui-ci répond, le gestionnaire ORB du client reçoit cette réponse et la fait suivre selon la méthode appropriée au serveur manipulateur de réponses (`ReplyHandler`) de façon que le client puisse traiter cette réponse. En d'autres termes, le gestionnaire ORB convertit la réponse en requête dans le manipulateur de réponses du client. Ce manipulateur est un objet CORBA normal qui est mis en œuvre par le programmeur comme dans une mise en œuvre quelconque d'objet. Dans le modèle de *requête à interrogation*, le client fait passer la requête par tous les paramètres nécessaires pour l'invocation et reçoit en retour un objet interrogateur (`Poller`) qui peut être interrogé pour obtenir les résultats de l'invocation. Cet interrogateur est une instance d'un type de valeur (`valueType`) qui est un nouveau type IDL introduit par la nouvelle spécification d'*objets par valeur*. Un type de valeur possède à la fois des membres d'un ensemble de données et des méthodes qui, lorsqu'elles sont invoquées, ne sont que des appels de fonction locaux et non pas des invocations d'opérations CORBA réparties.

La valeur de la capacité d'invocation de méthode asynchrone dans les applications de gestion de réseau est telle qu'elle permet aux systèmes gérants, qui souhaitent utiliser des appels de méthode asynchrone, d'interfonctionner avec des systèmes gérés au moyen des mêmes définitions d'interface que celles des clients synchrones. Aucune modification n'est requise dans la définition d'interface ou dans la mise en œuvre du système géré. Les exigences suivantes sont proposées pour les réalisations qui souhaitent, (en option), prendre en charge les invocations de méthode asynchrone (mais non les transactions avec capacités "ACID" (atomicité, cohérence, isolation et durabilité).

(O) AMI-1 La réalisation CORBA compatible avec les invocations AMI doit prendre en charge au moins le modèle de programmation à rappel.

(O) AMI-2 Pour chaque opération dans une interface IDL, la réalisation CORBA compatible avec les invocations AMI doit produire les signatures correspondantes de la méthode de rappel asynchrone. Ces signatures sont décrites en langage IDL implicite, qui est utilisé pour produire les signatures d'opération linguistiquement spécifiques.

(O) AMI-3 Lorsque des réponses contenant des exceptions sont renvoyées par l'objet CORBA, le gestionnaire ORB compatible avec les invocations AMI dans l'architecture CORBA doit transmettre au manipulateur de réponses une instance de valeur `ExceptionHolder` propre à chaque type, contenant les exceptions ordonnancées en tant qu'états individuels de ce gestionnaire. Le compilateur IDL

compatible avec les invocations AMI produira pour chaque interface IDL un conteneur d'exceptions (`ExceptionHolder`).

(O) AMI-4 Le compilateur IDL compatible avec les invocations AMI doit produire un manipulateur de réponses propre à chaque type et à chaque interface IDL. Le client mettra en œuvre et inscrira un manipulateur de réponses dans chaque réponse asynchrone puis recevra un rappel lorsqu'une réponse sera donnée à une requête. Ce manipulateur de réponses est déduit du manipulateur générique `Messaging::ReplyHandler`.

6.5 Service de sécurité

Le service de sécurité de l'architecture CORBA se compose des capacités de sécurité suivantes: authentification des principaux (utilisateurs humains et objets), autorisation d'accès aux objets par les principaux, analyse de sécurité, sûreté des communications, non-répudiation, et administration [6]. Toutes ces capacités peuvent cependant être trop puissantes pour de nombreuses applications. Celles-ci peuvent n'avoir besoin au contraire que des capacités de sûreté des communications et d'authentification au niveau du système, sur la base de la technique de sécurité de couche de transport (TLS) (et de son précurseur, la couche SSL) pour des raisons de disponibilité et de simplicité. Finalement certaines applications pourront n'exiger aucune sécurité. Les prescriptions facultatives ci-dessous reflètent donc trois choix possibles:

1) l'absence de sécurité;
2) l'utilisation par les gestionnaires ORB de la couche SSL pour assurer la sûreté des communications et l'authentification au niveau du système, ce qui revient pratiquement à une sécurité "de session";
3) l'utilisation par les gestionnaires ORB du service de sécurité CORBA pour assurer la sûreté des communications, l'authentification, la non-répudiation, les listes de contrôles d'accès pour groupes ou individus accédant à des objets individuels et à des opérations, etc.

Le niveau réel de service à fournir dans une interface est laissé au titre d'un point de négociation entre les parties fournissant les systèmes gérés et gérants.

(O) SEC-1 L'interface CORBA peut (en option) prendre en charge soit le "protocole IOP sécurisé" soit "l'interopérabilité CORBA avec la couche SSL de sécurité", comme défini dans la spécification du service de sécurité CORBA [6].

(O) SEC-2 Le service de sécurité CORBA peut être utilisé afin d'assurer son vaste ensemble de capacités.

(O) SEC-3 La prise en charge de l'échange de certificats d'authentification doit être une option offerte à l'administration.

6.6 Service de transaction

Dans un environnement de calcul réparti comme l'architecture CORBA, il est possible que des mises à jour provenant de certains clients soient effacées par recouvrement de mises à jour concurrentes (ou presque concurrentes) provenant d'autres clients, à moins que des protections appropriées ne soient mises en place. Bien que le service de notification et le service de journalisation des télécommunications constituent une base pour informer un client du fait que sa mise à jour a été recouverte, ces services n'offrent pas de mécanisme de blocage afin d'empêcher l'apparition de tels recouvrements. Le service OMG de transaction [7] offre un mécanisme de blocage élaboré permettant d'empêcher le recouvrement de la mise à jour d'un client par une mise à jour concurrente provenant d'un autre client. Cette solution vise à assurer une fiabilité élevée. Le service OMG de transaction n'est cependant pas requis par toutes les applications et le surdébit ainsi ajouté n'est pas toujours justifié. S'il faut assurer la cohérence des données après des mises à jour concurrentes, il y a lieu d'envisager le service de transaction du groupe OMG.

(O) TRANS-1 L'interface CORBA peut (en option) prendre en charge le service OMG de transaction afin de garantir la cohérence des données.

7 Services supports cadres

Le présent paragraphe définit les services supports communs qui sont inclus dans le cadre général mais qui nesont pas des services communs aux objets CORBA normalisés par le groupe OMG. De nombreuses applications de gestion de réseau remplissent des fonctions qui ne sont pas couramment requises par les applications bureautiques générales, de sorte qu'il n'est pas logique de s'attendre à une fourniture, par le cadre CORBA normalisé, de tous les services nécessaires à la gestion de réseau. Le présent paragraphe définit les services qui seront largement utilisés par les applications de gestion de réseau mais qui ne sont pas susceptibles d'être utilisés par un grand nombre d'autres types d'applications et donc de devenir des services communs aux objets CORBA. Ces services offrent également les capacités requises pour permettre la réutilisation des modèles d'information existants sans modification notable de la sémantique.

Les avantages présentés par l'insertion de ces capacités dans les services supports communs est que cela dispense les réalisations d'objets gérés de la fourniture de ces services et permet à ceux-ci d'évoluer de façon à offrir un plus grand nombre de capacités fonctionnelles sans modification des interfaces avec les objets gérés ou des réalisations correspondantes. La description en langage IDL des interfaces avec ces services est donnée dans l'Annexe A.

7.1 Service détecteur d'atelier

Le service détecteur d'atelier permet aux clients de trouver des ateliers. A cette fin, un client recherche ce service au moyen du nom de classe d'un atelier. Le service renvoie en réponse la référence d'un atelier de ce type. Noter que le nom de classe fourni par le client désigne la classe d'atelier et non la classe de l'objet à créer.

L'on trouve le service détecteur d'atelier en le recherchant dans le service de nommage.

Avant que des ateliers puissent être trouvés dans le service, celui-ci doit en être informé. Le service détecteur d'atelier permet donc aux ateliers de s'inscrire et de se désinscrire auprès du service. Bien qu'il ne soit pas nécessaire de décrire cette capacité de part et d'autre de l'interface de gestion, ces opérations sont définies de façon à permettre la mise en œuvre du service détecteur d'atelier en tant que composant distinct des objets constituants d'un modèle d'information spécifique. Une fois réalisé, le service détecteur d'atelier n'aura donc pas à être modifié lors de la définition de nouveaux modèles d'information comportant de nouveaux ateliers. Le service détecteur d'atelier pourra même être acquis en provenance d'une tierce partie.

Les opérations utilisées pour inscrire et désinscrire des canaux se trouvent dans une interface distincte, appartenant à une sous-classe de l'interface du détecteur d'atelier, qui est la seule interface à mettre en œuvre pour assurer la conformité au présent cadre général. L'interface sous-classée est fournie pour les réalisations qui souhaitent l'utiliser afin de construire le service détecteur d'atelier en tant que composant distinct.

La définition en langage IDL de l'interface avec le service détecteur d'atelier est donnée ci-après (sans commentaires):

```
interface FactoryFinder {

        itut_x780::ManagedObjectFactory find (
                in ObjectClassType factoryClass)
                raises (FactoryNotFound, itut_x780::ApplicationError);

        FactoryInfoSetType list()
                raises (itut_x780::ApplicationError);
}; // fin de l'interface FactoryFinder
```

```
interface FactoryFinderComponent : FactoryFinder {

        void register (in ObjectClassType factoryClass,
                in itut_x780::ManagedObjectFactory factoryRef)
                raises (itut_x780::ApplicationError);

        void unregister (in ObjectClassType factoryClass,
                in itut_x780::ManagedObjectFactory factoryRef)
                raises (FactoryNotFound, itut_x780::ApplicationError);

}; // fin de l'interface FactoryFinderComponent
```

L'opération *find* de l'interface *FactoryFinder* est utilisée par un client pour trouver un atelier d'un type particulier. L'opération *list* renvoie la liste de tous les ateliers inscrits auprès du détecteur d'atelier. L'opération *register* de l'interface *FactoryFinderComponent* est utilisée par un atelier pour s'inscrire au service, tandis que l'opération *unregister* est utilisée par un atelier pour supprimer son inscription. Ces deux dernières opérations ne doivent pas être utilisées par les systèmes gérants.

(R) FACTORY_FINDER-1 Un système géré doit instancier au moins un objet de service détecteur d'atelier. De même, chaque contexte de nommage de racine locale doit, dans un système, avoir au moins une corrélation de noms pour un objet de service détecteur d'atelier. La valeur de la chaîne *ID* doit simplement désigner le serveur dans cette corrélation, éventuellement avec une valeur similaire à "FactoryFinder1". La chaîne *kind* de la corrélation doit désigner la classe de l'objet ("itut_q816::FactoryFinder").

(R) FACTORY_FINDER-2 L'objet (les objets) de service détecteur d'atelier doit (doivent) prendre en charge l'interface de détecteur d'atelier décrite ci-dessus et définie dans l'Annexe A en langage IDL de l'architecture CORBA. L'objet (les objets) de service détecteur d'atelier peut (peuvent) prendre en charge les interfaces de composant de détecteur d'atelier définies ci-dessus. Les capacités fonctionnelles décrites ci-dessus doivent être prises en charge.

7.2 Service détecteur de canaux

L'on s'attend qu'un grand système de gestion de réseau pourra avoir de multiples canaux d'événements qui pourront traiter différents types d'événement ou des événements issus de différents ensembles d'objets. Afin de garantir qu'un client ne manquera aucun des événements auxquels il s'intéresse, il faut trouver un moyen pour identifier les canaux présents dans un système géré ainsi que les événements qui y sont manipulés. C'est le service détecteur de canaux qui remplit cette fonction. Un client peut invoquer une opération de ce service pour énumérer tous les canaux d'événement présents dans un système géré, de même que les événements qui y sont manipulés. Une fois que le client est informé des canaux, il peut interagir avec eux pour configurer la réception des notifications.

Un client trouve un objet détecteur de canaux en le recherchant dans le service de nommage.

Avant que les canaux puissent être énumérés par ce service, celui-ci doit en être informé. Le service détecteur de canaux permet donc d'inscrire et de désinscrire des canaux auprès de ce service. Bien qu'il ne soit pas nécessaire de décrire cette capacité de part et d'autre de l'interface de gestion, ces opérations sont définies afin de permettre la mise en œuvre du service détecteur de canaux en tant que composant, distinct des objets constituant un modèle d'information spécifique. Une fois mis en œuvre, le service détecteur de canaux n'aura donc pas à être modifié lorsque de nouveaux modèles d'information seront définis. Le service détecteur de canaux peut même être acquis en provenance d'une tierce partie.

Les opérations utilisées pour inscrire et désinscrire des canaux se trouvent dans une interface distincte, appartenant à une sous-classe de l'interface du détecteur de canaux, qui est la seule interface à mettre en œuvre pour assurer la conformité au présent cadre générale. L'interface

sous-classée est fournie pour les réalisations qui souhaitent l'utiliser afin de construire le service détecteur de canaux en tant que composant distinct.

7.2.1 Interface avec le détecteur de canaux

La définition en langage IDL de l'interface avec le service détecteur d'atelier est donnée ci-après (sans commentaires):

```
interface ChannelFinder {

        ChannelInfoSetType list()
                raises (itut_x780::ApplicationError);

}; // fin de l'interface ChannelFinder


interface ChannelFinderComponent : ChannelFinder {
        void register (in ChannelIDType channelID,
                in ObjectClassType channelClass,
                in BaseAndScopeSetType baseAndScopes,
                in EventSetType eventTypes,
                in EventSetType excludedEventTypes,
                in ScopedNameSetType sourceClasses,
                in ScopedNameSetType excludedSourceClasses,
                in EventChannel channel)
                raises (ChannelAlreadyRegistered,
                        itut_x780::ApplicationError);

        void unregister (in ChannelIDType channelID)
                raises (ChannelNotFound, itut_x780::ApplicationError);

}; // fin de l'interface ChannelFinderComponent
```

L'opération *list* de l'interface *ChannelFinder* peut être utilisée par un système gérant pour trouver tous les canaux présents dans un système géré. Les informations renvoyées sont celles qui sont incluses lorsqu'un canal est inscrit, ce qui est examiné ci-après. Les deux opérations à l'interface *ChannelFinderComponent* sont utilisées par le système géré pour inscrire et désinscrire ses canaux auprès du service. Ces opérations ne doivent pas être utilisées par les systèmes gérants.

7.2.1.1 Données du détecteur de canaux

Lorsqu'un canal est inscrit, sept éléments de données sont associés à la référence du canal. D'abord, un identificateur de chaîne est spécifié pour le canal. Ensuite, le type du canal est identifié (le présent cadre possède deux types de canaux: de notification et de journalisation).

Ensuite, l'ensemble des objets couverts par le canal est spécifié par un ensemble de noms d'objet géré et de "domaines de visibilité" d'objets confinés au-dessous d'eux. Voir au § 7.4.1.1 de plus amples détails sur les domaines de visibilité. Il y a un domaine de visibilité pour chaque objet géré de base contenu dans l'ensemble. Ces objets de base et les objets contenus dans leurs domaines de visibilité associés sont les ensembles d'objets qui envoient des événements dans le canal. Si plusieurs canaux ont des domaines de visibilité qui se superposent, c'est le canal dont l'objet de base est le plus proche ascendant de l'objet source qui manipulera les événements issus de cet objet, conformément à certaines restrictions examinées ci-après. Si plusieurs canaux ont le même objet de base et que c'est le plus proche ascendant d'un objet source, tous ces canaux traitent les événements issus de cet objet source. Un ensemble vide a la signification spéciale que tous les objets gérés directement liés au contexte de nommage de la racine locale, qui contient également ce détecteur de canaux, sont des objets de base et que le domaine de visibilité des objets qui envoient des événements au canal est l'ensemble des sous-arborescences contenues dans ces objets.

L'on identifie ensuite les types d'événement envoyés dans le canal considéré. A cette fin, l'on utilise deux ensembles de chaînes. L'ensemble *eventTypes* comprend explicitement le nom de chaque type d'événement envoyé au canal considéré. Un exemple de type d'événement est

"itut_x780::Notifications::equipmentAlarm". Un ensemble vide a la signification spéciale que tous les types d'événement sont envoyés dans le canal considéré. L'ensemble *excludedEventTypes* énumère les types d'événement qui ne sont pas envoyés au canal considéré. Si l'ensemble de chaînes *eventTypes* n'est pas vide, tous les événements sont envoyés au canal à l'exception de ceux qui sont énumérés dans l'ensemble *excludedEventTypes*. Si l'ensemble de chaînes *eventTypes* n'est pas vide, le paramètre *excludedEventTypes* doit être vide et est négligé.

Finalement, les types d'objet envoyant des événements dans le canal sont identifiés, ce qui est également effectué avec deux ensembles de chaînes: l'ensemble *sourceClasses* inclut explicitement le nom de chaque classe d'objets gérés qui envoie des événements dans le canal considéré. Exemple de nom de classe d'objet: "itut_m3120::Equipment". Un ensemble vide a la signification spéciale que tous les types d'événement sont envoyés dans le canal considéré. L'ensemble *excludedSourceClasses* énumère les noms des classes d'objets gérés qui n'envoient pas d'événements dans le canal. Si l'ensemble de chaînes *sourceClasses* est vide, toutes les classes d'objets gérés envoient des événements au canal, sauf celles qui sont énumérées dans l'ensemble *excludedSourceClasses*. Si l'ensemble de chaînes *sourceClasses* n'est pas vide, le paramètre *excludedSourceClasses* doit être vide et est négligé.

7.2.1.2 Domaine d'application d'un canal

La combinaison des objets de base, de la liste de types d'événement et des classes de source détermine les événements manipulés par un canal, ce qui peut conduire à certaines combinaisons prêtant à confusion. Bien que toutes ces combinaisons ne soient pas censées être mises en œuvre dans les systèmes, elles sont expliquées ci-dessous à toutes fins utiles:

1) un canal peut être inscrit avec plusieurs objets de base et plusieurs domaines de visibilité;

2) plusieurs canaux peuvent être inscrits avec le même objet de base. Si les listes d'événements et les types d'objet de source associés à ces canaux se superposent aussi, les événements issus des objets contenus dans les domaines de visibilité superposés doivent être mis à la disposition de tous les canaux énumérant les types correspondants d'événement et d'objet source;

3) une tentative de réinscription d'un canal se traduira par une mise à jour de l'inscription de ce canal. Les nouvelles informations seront associées au canal et les anciennes informations seront supprimées. Une notification signalant ce changement sera envoyée;

4) Un objet géré envoie toujours un événement au canal dont l'objet de base associé:
   - possède un objet de base et un domaine de visibilité qui contient l'objet;
   - possède l'objet de base qui est le plus proche ascendant de l'objet géré (si d'autres canaux ont également des objets de base et des domaines de visibilité qui contiennent l'objet);
   - manipule ce type d'événement;
   - et manipule cette classe d'objets.

   Si plusieurs canaux sont associés, l'objet géré envoie l'événement à tous ces canaux.

La Figure 9 décrit également, sous forme graphique, quelques exemples pour plus de clarté.

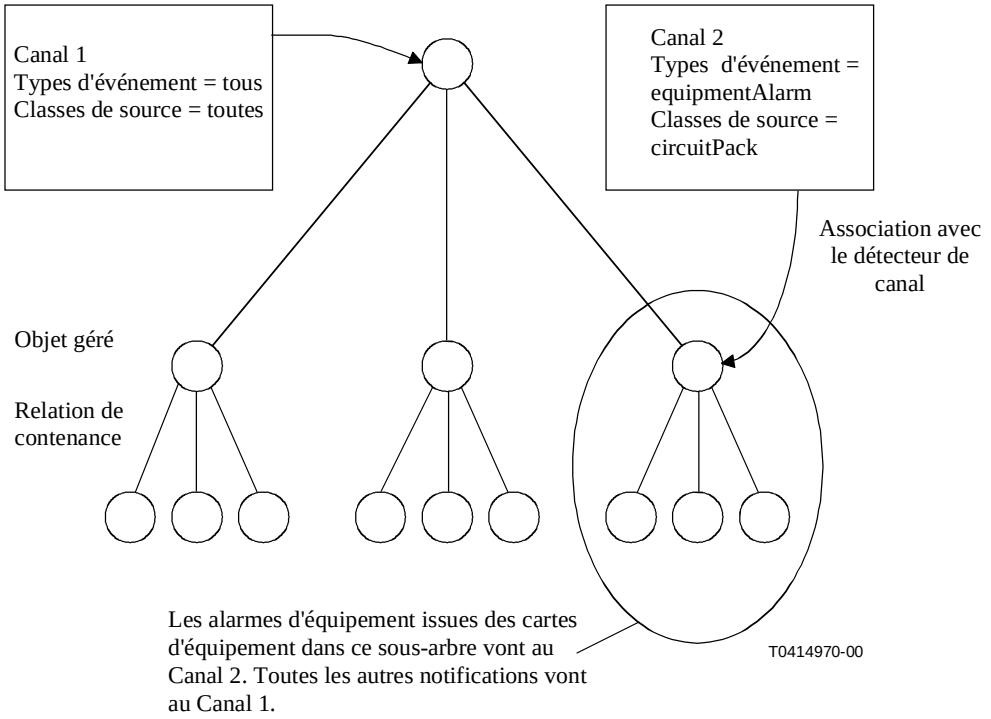

Figure 9/Q.816 – Exemple de canal d'événements

7.2.1.3 Notification de détecteur de canaux

Une notification spéciale a été définie, qui doit être envoyée chaque fois qu'une inscription de canal est ajoutée, supprimée ou modifiée. Cette notification doit être envoyée dans tous les canaux inscrits, immédiatement avant que la modification se soit produite. Etant donné que cette notification est envoyée à tous les canaux, le détecteur de canaux ne doit pas énumérer les notifications de changement de canal en même temps que les autres notifications manipulées par un canal. La description de cette notification en langage IDL est la suivante:

```
void channelChange (
       in ChannelModificationType   channelModification,
       in ChannelInfoType           channelInfo
);
```

Le paramètre de modification de canal indique si un canal a été ajouté, supprimé ou modifié (en termes des types d'événement qu'il manipule). Le paramètre d'informations de canal fournit une chaîne décrivant le type de canal, une référence à ce canal, l'objet ou les objets de base auxquels il est associé, ainsi que les classes d'objet de source et les types d'événement qu'il manipule.

Il convient de noter que, bien que cette méthode de prise en charge de multiples canaux d'événements soit tout à fait flexible et donc compliquée, le degré de complication dépend de la mise en œuvre du système géré. La création et l'inscription de canaux d'événements de part et d'autre de l'interface de gestion ne sont pas prises en charge. Un système géré complexe pourrait prendre en charge une procédure administrative locale afin d'améliorer la qualité de fonctionnement en ajoutant, en modifiant ou en supprimant des canaux. Il pourrait également se limiter à mettre à jour les canaux en fonction des versions logicielles. Un système simple aura sans doute un ou deux canaux (un canal pour des événements de haute priorité comme les alarmes d'équipement et les autres canaux pour tous les autres signaux) associés à l'objet géré racine. De même, il convient de noter que le fait d'empêcher un système gérant de créer des canaux d'événements ne l'empêche pas de créer des filtres et des "fournisseurs intermédiaires" dans les canaux existants, ce qui offre au client des possibilités

équivalentes à la création de discriminateurs de retransmission d'événements dans les systèmes OSI de gestion de réseau.

7.2.2 Exigences du détecteur de canaux

(R) CHANNEL_FINDER-1 Un système géré doit instancier au moins un objet de service détecteur de canaux. De même, chaque contexte de nommage de racine locale doit avoir au moins, dans un système donné, une corrélation de noms pour un objet de service détecteur de canaux. La valeur de la chaîne *ID* doit, dans cette corrélation, désigner simplement le serveur, éventuellement avec une valeur comme "ChannelFinder1". La chaîne *kind* (sorte) de la corrélation doit désigner la classe de l'objet ("itut_q816::ChannelFinder").

(R) CHANNEL_FINDER-2 L'objet ou les objets de service détecteur de canaux doit (doivent) prendre en charge l'interface de détecteur de canaux décrite ci-dessus et définie dans l'Annexe A en langage IDL de l'architecture CORBA. L'objet ou les objets de service détecteur de canaux peut ou peuvent prendre en charge les interfaces de composant de détecteur de canaux définies ci-dessus. La capacité fonctionnelle décrite ci-dessus doit être prise en charge.

(R) CHANNEL_FINDER-3 Chaque fois qu'une modification est apportée aux inscriptions de canal, le détecteur de canaux doit envoyer une notification de changement de canal à tous les canaux qui étaient inscrits immédiatement avant le changement.

(R) CHANNEL_FINDER-4 Le réseau de canaux d'événements signalés par un système géré doit manipuler toutes les alarmes provenant de tous les objets gérés du système. Un système qui énumère un ensemble de canaux ne contenant pas tous les événements issus de tous les objets gérés du système n'est pas conforme au présent cadre général.

7.3 Service terminateur

L'objet du service terminateur est de constituer, dans le cadre général, un lieu de mise en œuvre d'une capacité fonctionnelle commune qu'il faudrait autrement mettre en œuvre dans les objets gérés. Le service terminateur est utilisé par les systèmes gérants pour supprimer des objets gérés conformément à la politique de suppression de ces objets. (UIT-T X.780 prescrit que chaque objet géré doit avoir un attribut *deletePolicy* ayant l'une des trois valeurs suivantes: *notDeletable*, *deleteOnlyIfNoContainedObjects* et *deleteContainedObjects*.) Si la politique de suppression de l'objet géré est de type *notDeletable*, le service terminateur ne supprime pas l'objet mais lève une exception. Si la politique de suppression est de type *deleteOnlyIfNoContainedObjects* et que l'objet ne contienne aucun objet, le service terminateur supprime cet objet. Sinon, il lève une exception. Finalement, si la politique de suppression et de type *deleteContainedObjects*, le service terminateur supprime l'objet ainsi que tous les objets qu'il contient, en vertu des quelques règles définies ci-dessous.

La Recommandation UIT-T X.780 définit également une opération destroy qui doit être prise en charge par tous les objets gérés et qui est destinée à être utilisée par le service terminateur pour supprimer réellement les objets gérés et libérer leurs ressources. Le service terminateur ne suit pas seulement les politiques de suppression et ne supprime pas seulement les objets gérés mais il constitue un lieu approprié à la mise en œuvre d'un code visant à maintenir l'intégrité de l'arbre de nommage en supprimant les corrélations de noms relatives aux objets gérés qui vont être supprimés. Les réalisations peuvent choisir d'implanter ailleurs cette fonction mais un des objectifs du cadre général est de permettre des mises en œuvre d'objets gérés représentant essentiellement des ressources de réseau. L'on estime qu'un service comme celui-là simplifiera la mise en œuvre d'objets gérés.

La description en langage IDL du service terminateur offre deux méthodes pour supprimer un objet géré: l'une identifie celui-ci par son nom, l'autre par une référence. La définition IDL de l'interface du service de suppression est la suivante:

```
interface TerminatorService {

void deleteByName (in NameType name)
        raises (itut_x780::ApplicationError,
                itut_x780::DeleteError);

void deleteByRef (in itut_x780::ManagedObject mo)
        raises (itut_x780::ApplicationError,
                itut_x780::DeleteError);

        }; // fin de l'interface TerminatorService
```

(R) TERM-1 Un système géré doit instancier au moins un objet de service terminateur. De même, chaque contexte de nommage de racine locale doit avoir au moins, dans un système donné, une corrélation de noms pour un objet de service terminateur. La valeur de la chaîne *ID* doit, dans cette corrélation, désigner simplement le serveur, éventuellement avec une valeur comme "Terminator1". La chaîne *kind* (sorte) de la corrélation doit désigner la classe de l'objet ("itut_q816::TerminatorService").

(R) TERM-2 L'interface prise en charge par l'objet ou les objets de service terminateur doit être l'interface de terminateur qui est décrite ci-dessus et définie en Annexe A en langage IDL de l'architecture CORBA. La capacité fonctionnelle décrite ci-dessus doit être prise en charge.

(R) TERM-3 Le service terminateur doit supprimer les objets conformément à l'attribut de politique de suppression des objets. Cet attribut est posé lors de la création et ne peut pas être modifié. Noter que le service terminateur n'est pas un service détecté dans un domaine de visibilité. Ce service peut supprimer effectivement de multiples objets en réponse à une requête isolée mais il vise essentiellement le seul objet dont la suppression est demandée. Le service terminateur doit mettre en œuvre les règles suivantes lors de la suppression d'un objet:

1) aucun objet ne doit jamais être "laissé à l'abandon". En d'autres termes, aucun objet ne peut être supprimé sans que ses subordonnés le soient également;

2) si l'objet a comme politique de suppression *notDeletable*, cet objet ne doit pas être supprimé ni aucun de ses éventuels subordonnés. L'exception *DeleteError* doit être levée avec l'identificateur d'erreur mis à la valeur *notDeletable*;

3) si l'objet a comme politique de suppression *deleteOnlyIfNoContainedObjects* et qu'il ne possède pas de subordonnés, l'objet doit être supprimé. Si l'objet a des subordonnés, quelle que soit leur politique de suppression, il ne doit pas être supprimé ni aucun de ses subordonnés. L'exception *DeleteError* doit être levée avec l'identificateur d'erreur mis à la valeur *containsObjects*;

4) si l'objet a comme politique de suppression *deleteContainedObjects* et qu'il ne possède pas de subordonnés, l'objet doit être supprimé. Si l'objet a des subordonnés, le service terminateur doit vérifier les politiques de suppression de tous les subordonnés. Si un subordonné quelconque a comme politique de suppression *notDeletable*, cet objet n'est pas supprimé. Si certains objets ont la politique *deleteOnlyIfNoContained* et qu'ils contiennent des subordonnés, ces objets ne sont pas supprimés. Sinon, l'objet et ses subordonnés sont supprimés;

5) le service terminateur doit supprimer les objets contenus de bas en haut. Si un quelconque objet contenu lève une exception en cours de suppression, le service terminateur ne doit supprimer ni le nom de cet objet ni aucun de ses ascendants. Le service terminateur doit cependant continuer à essayer de supprimer les autres objets contenus. Lorsque tous les objets qui peuvent être supprimés l'ont été, le service terminateur doit lever une exception *DeleteError* avec l'identificateur d'erreur mis à la valeur *undeletableContainedObject*. Cette méthode de suppression au mieux des objets contenus est requise pour rendre les résultats déterministes. Le client peut identifier les objets non supprimables parce qu'ils seront dans les feuilles de l'arbre restant au-dessous de l'objet cible;

6) si l'objet de base lève une exception *DeleteError*, le service terminateur doit renvoyer la même exception (et les données incluses). L'objet n'est pas supprimé et son nom n'est pas supprimé de l'arbre de nommage.

7.4 Service d'opérations sur objets multiples

Comme il peut y avoir des millions d'entités à gérer, il est nécessaire que le cadre prenne en charge les opérations sur objets multiples au moyen d'un seule invocation de méthode (ou peut-être un petit nombre d'invocations). Le service d'opérations sur objets multiples (MOO) offre cette capacité.

L'on s'attend que chaque plate-forme de gestion de réseau prenant en charge une interface CORBA offrira au moins une instance du service d'opérations MOO. (Pour des raisons liées à la qualité de fonctionnement, il est recommandé que le service d'opérations MOO, le service de nommage et les objets gérés soient situés dans la même plate-forme de calcul.) Les gestionnaires entreront en interaction avec ce service au moyen d'un petit nombre d'interactions exigeant une largeur de bande relativement petite. Le service entrera à son tour en interaction avec les objets gérés par l'intermédiaire de leur interface CORBA publiée ou par un moyen non normalisé. Ce grand nombre d'interactions est censé utiliser une plus grande largeur de bande, ce qui conduit à copositionner le service et les objets gérés.

On notera que le service d'opérations MOO est un exemple de granularité d'accès propre à l'application (voir paragraphe 4).

7.4.1 Interface avec le service d'opérations MOO

L'interface avec le service d'opérations MOO, définie dans l'Annexe A, est faiblement typée. Elle offre un ensemble de capacités génériques qui peuvent être invoquées au sujet d'ensembles de sortes quelconques d'objets gérés, même de types différents. Les opérations prises en charge sont les suivantes:

- Opération d'obtention détectée: renvoie les valeurs de chacun des objets pour une liste d'attributs.
- Opération de mise à jour détectée: remplace une valeur d'attribut ou ajoute/supprime des valeurs dans des attributs à valeur d'ensemble. Cette opération peut servir à mettre à jour un ou plusieurs attributs dans un seul objet ou dans plusieurs objets.
- Opération de suppression détectée: supprime plusieurs objets.

L'opération d'obtention détectée est définie à l'interface *BasicMooService*. Les opérations de mise à jour et de suppression détectées sont définies à l'interface *AdvancedMooService*, qui hérite l'opération d'obtention détectée de l'interface du service de base. Cela permet une certaine flexibilité dans la mise en œuvre de services d'opération sur objets multiples. Un service de base n'a besoin de mettre en œuvre que l'opération d'obtention détectée.

7.4.1.1 Paramètres communs aux opérations du service d'opérations MOO

Chacune des opérations détectées exige quatre paramètres pour définir l'ensemble des objets sur lesquels l'opération sera effectuée:

- Nom d'objet de base: nom de l'objet situé à la racine d'un arbre d'objets sur lequel l'opération pourra être effectuée.
- Domaine de visibilité: réunion d'ensembles discriminés définissant les objets contenus par l'objet de base sur lequel l'opération pourra être effectuée. Cette réunion d'ensembles comporte quatre cases, dont deux contiennent un entier spécifiant un niveau d'objets contenus au-dessous de l'objet de base. Les quatre options sont les suivantes:
  - objet de base seulement. Si le domaine de visibilité est *baseObjectOnly*, seul l'objet cible (de base) nommé est inclus dans ce domaine;

– sous-arborescence entière. Si le domaine de visibilité est *wholeSubtree*, le domaine de visibilité englobe tous les objets contenus au-dessous de l'objet de base, y compris celui-ci;
– niveau individuel. Si le domaine de visibilité est *individualLevel*, ce domaine comprendra également un niveau *level* à valeur d'entier. Tous les objets contenus à un niveau inférieur à l'objet de base égal à cette valeur sont dans le domaine de visibilité. Les objets directement contenus par l'objet de base sont au niveau 1. Si la valeur du niveau est égale à zéro, le domaine est l'objet de base;
– base à niveau. Si le domaine de visibilité est *baseToLevel*, ce domaine comprendra également un niveau *level* à valeur d'entier. Il inclura tous les objets en descendant de la base jusqu'au niveau indiqué, y compris l'objet de base et l'objet du niveau indiqué. Si la valeur du niveau est égale à zéro, le domaine est seulement l'objet de base.

Noter qu'en raison de l'utilisation de conventions de nommage uniques par le présent cadre général, le service doit effectuer un petit travail afin de déterminer la profondeur réelle des domaines de visibilité fondés sur les contenances. L'objectif est de rendre ce travail aussi simple que possible pour le client. Le nom de l'objet de base sera d'abord le nom composé entier, y compris le composant final ayant une valeur d'identification égale à "Object". Le service devra revenir au contexte de nommage contenant cette corrélation et commencer à compter à partir de ce point. De même, le service doit suivre automatiquement les corrélations identifiées par "Object" dans les contextes de nommage d'objets gérés contenus dans le domaine de visibilité. Cette dernière opération n'effectuera pas de comptage de niveaux de profondeur.

- Filtre: expression écrite dans un langage de contrainte et utilisée pour évaluer les attributs d'un objet. L'opération est appliquée aux objets contenus dans le domaine de visibilité pour lesquels l'expression de filtrage donne la valeur *true*.
- Langue: chaîne indiquant la langue dans laquelle l'expression de filtrage est écrite.

Les noms d'objet sont les mêmes que ceux du type *Name* défini par le service de nommage CORBA. Le domaine de visibilité est une réunion d'ensembles dont les valeurs sont décrites ci-dessus. Finalement, les paramètres de filtre et de langue sont des chaînes.

Chacune des opérations peut lever une des exceptions suivantes:

- une exception de type *InvalidParameter* si l'un des paramètres a une valeur invalide. Le nom du paramètre invalide est renvoyé. Certaines des conditions dans lesquelles cette exception sera levée sont les suivantes:
  – le nom de l'objet de base est invalide;
  – une valeur non reconnue de langue de filtrage est fournie.
- une exception de type *InvalidFilter* si la syntaxe du filtre est incorrecte. Cette exception renvoie le texte situé près de l'erreur de syntaxe afin de permettre le dépannage;
- une exception de type *FilterComplexityLimit* si la syntaxe du filtre est correcte mais que le filtre soit trop complexe pour être traité par le système géré;
- une exception de type *ApplicationError* afin de signaler d'autres problèmes dans le serveur (comme un manque de ressources) rendant impossible l'exécution de l'opération requise.

L'on notera que si une expression ne peut pas être évaluée pour un objet particulier parce que les types de ses attributs ne correspondent pas à cette expression, le filtre n'est pas invalide: cet objet peut simplement ne pas réussir à traverser le filtre.

Les autres paramètres des opérations, ainsi que les types de renvoi, sont propres à chaque opération. Par exemple, l'opération d'obtention détectée prend une liste de noms d'attribut et renvoie une séquence de résultats issus chacun d'un objet, le nom de celui-ci étant associé aux résultats qui en proviennent.

7.4.1.2 Itérateurs de service d'opérations MOO

Etant donné que chacune des opérations peut renvoyer de grandes quantités de données, le modèle théorique d'itérateur est utilisé pour renvoyer les résultats. Un itérateur est un objet qui est créé afin de contenir les résultats d'une opération de façon qu'ils puissent être renvoyés au client à une fréquence déterminée par celui-ci. Le client reçoit une référence à l'itérateur dans le cadre des informations renvoyées par la méthode. Le client peut ensuite invoquer des opérations au sujet de l'itérateur afin de recevoir des lots de résultats dimensionnés par le client. L'itérateur garde trace du point jusqu'où le client a progressé dans les résultats.

Chaque fois qu'un client invoque une opération d'obtention *get* auprès d'un itérateur, il fournit le nombre maximal de réponses qu'il souhaite recevoir. L'itérateur ne doit pas renvoyer une série de taille supérieure à celle qui a été demandée, dans une structure de données de type séquence *sequence*. L'itérateur renvoie également une indication booléenne s'il faut extraire un plus grand nombre de résultats. L'itérateur peut renvoyer une série de taille inférieure à celle qui a été demandée afin de compenser en efficacité le renvoi de résultats en grande série, qui peut nécessiter un blocage dans l'attente d'un plus grand nombre de résultats. L'itérateur ne doit pas renvoyer de séquence vide à moins qu'il n'y ait plus de résultats à renvoyer, car cela obligerait le client à interroger l'itérateur.

Le système géré régit le cycle de vie de l'itérateur. Une opération de destruction est cependant prévue dans le cas où le gestionnaire souhaiterait mettre fin à l'extraction de résultats avant que la dernière itération soit atteinte. Dès le dernier résultat renvoyé, l'itérateur doit se détruire lui-même. Il peut également être détruit par le système géré s'il est inutilisé pendant une période exagérément longue.

Noter que les itérateurs ne servent qu'à régler la fréquence de retour des informations provenant des opérations et qu'ils ne devraient pas agir sur le moment où les opérations sont effectivement invoquées au sujet d'objets individuels. Il y a lieu d'invoquer une opération détectée au sujet des objets et de mettre les résultats en file d'attente dès que possible. Il est parfois plus efficace de retarder l'invocation de l'opération au sujet des objets gérés individuels en attendant que les résultats soient demandés par l'intermédiaire de l'itérateur, mais cela peut conduire à des résultats incorrects ou à des conditions critiques.

Les paragraphes qui suivent donnent de plus amples détails sur chacune des opérations détectées.

7.4.1.3 Opération d'obtention détectée

La signature IDL de l'opération d'obtention détectée est la suivante dans le service d'opération MOO de base:

```
GetResultsSetType scopedGet (
        in NameType baseName,
        in ScopeType scope,
        in FilterType filter,
        in LanguageType language,
        in StringSetType attributes,
        in unsigned short howMany,
        out GetResultsIterator resultsIterator)
        raises (InvalidParameter,
                InvalidFilter,
                FilterComplexityLimit,
                Itut_x780::ApplicationError);
```

Comme décrit ci-dessus, les quatre premiers paramètres servent à sélectionner un ensemble d'objets sur lesquels l'opération d'obtention sera effectuée. Pour chacun de ces objets, le service essaiera de renvoyer une valeur relative à chacun des attributs nommés dans le paramètre "attributs", qui n'est qu'une liste de chaînes. Une liste d'attributs vides a, lorsqu'elle est soumise, la signification spéciale que *toutes* les valeurs d'attribut doivent être renvoyées pour les objets qui satisfont au filtre. Les types contenus dans la valeur de retour sont les suivants:

```
struct AttributeValueType {     // from itut_x780 framework
        string  attributeName;
        any     value;          // type will depend on the attribute
};

typedef sequence <AttributeValueType> AttributeSetType; // framework

struct GetResultsType {
        NameType                name;
        boolean                 notFilterable;
        AttributeSetType        attributes;
        StringSetType           failedAttributes;
};

typedef sequence <GetResultsType> GetResultsSetType;
```

Les deux premiers types forment une liste de paires nom-valeur (*name-value*). Le type de retour est une séquence de structures individuelles pour chaque objet qui a satisfait au filtre. Une telle structure contient un nom d'objet, un fanion qui sera vrai si l'objet n'a pas pu être évalué pour voir s'il a satisfait au filtre, la liste des valeurs d'attribut issues de cet objet, et les noms des éventuels attributs qui n'ont pas pu être extraits de cet objet. Les objets qui n'ont pas pu être filtrés sont étiquetés comme des cas spéciaux parce qu'ils peuvent être des objets auxquels le client ne s'intéresse peut-être même pas. Si l'objet n'a pas pu être filtré, le client saura que le serveur d'opérations MOO n'a pas pu extraire d'attributs pour cet objet, de sorte que les deux autres éléments de données doivent être vides. Si un objet satisfait au filtre mais qu'une valeur d'attribut nepuisse pas être extraite, soit parce que l'objet n'a pas d'attribut correspondant ou parce qu'une exception a été levée au sujet de l'accès, le nom de cet attribut doit être extrait de la liste pour cet objet.

Le paramètre *howMany* indique au service combien de résultats d'objet il y a lieu d'inclure dans le premier lot de réponses. (La valeur zéro est admise, ce qui force le retour de tous les résultats par l'intermédiaire de l'itérateur.) Le paramètre de sortie *resultsIterator* est une référence à un objet itérateur qui peut être utilisé pour trouver des résultats additionnels dans des lots. Si tous les résultats ont été renvoyés par l'opération *scopedGet*, cette référence sera vide. Le client doit détruire cet objet lorsqu'il n'en a plus besoin. Il peut le supprimer avant que tous les résultats soient extraits. La capacité de l'opération CMIP d'annulation d'obtention est fournie dans le présent cadre par destruction de l'itérateur de résultats.

7.4.1.4 Opération de mise à jour détectée

La signature IDL de l'opération de mise à jour détectée est la suivante dans le service d'opérations MOO évolué:

```
UpdateResultsSetType scopedUpdate (
        in NameType baseName,
        in ScopeType scope,
        in FilterType filter,
        in LanguageType language,
        in ModificationSeqType modifications,
        in boolean failuresOnly,
        in unsigned short howMany,
        out UpdateResultsIterator resultsIterator)
        raises (InvalidParameter,
                InvalidFilter,
                filterComplexityLimit,
                itut_x780::ApplicationError);
```

Ici encore, les quatre premiers paramètres servent à sélectionner l'ensemble des objets dont la mise à jour doit être effectuée. La liste de modifications énumère des structures accompagnées chacune du nom d'un attribut, d'une valeur pour cet attribut et d'une valeur énumérée indiquant si la valeur doit remplacer la valeur actuelle de l'attribut, doit être ajoutée à la valeur actuelle de l'attribut ou doit en être retranchée. Les options d'addition et de soustraction ne sont valides que si le type de l'attribut est une séquence CORBA et que l'interface possède des opérations d'addition et de soustraction pour

l'attribut. Les valeurs contenues dans les structures de liste de modification sont acheminées sous forme de type CORBA *any*. Si le type de l'attribut est une séquence CORBA, il y a lieu d'insérer une séquence du type approprié dans le champ *any*, même s'il ne contient qu'une seule valeur. La description en langage IDL de la liste de modifications est la suivante:

```
enum ModificationOpType {set, add, remove};

struct ModificationType {
        string                attribute;
        ModificationOpType    op;
        any                   value;
};

typedef sequence <ModificationType> ModificationSeqType;
```

Le fanion *failuresOnly* sert à indiquer si le client souhaite que le service renvoie des résultats pour tous les objets correspondant au domaine de visibilité et au filtre ou seulement pour les objets auxquels au moins une des modifications n'a pas pu être exécutée bien que le domaine et le filtre aient été satisfaits.

La valeur de retour est une liste de structures contenant chacune un nom d'objet accompagnée d'une valeur booléenne indiquant si l'objet n'a pas pu être évalué pour voir s'il a satisfait au filtre, ainsi qu'une liste des éventuels attributs qui n'ont pas pu être modifiés. Si le service ne peut pas interagir avec l'objet afin de déterminer s'il satisfait au filtre, les résultats relatifs à cet objet auront le fanion *notFilterable* mis à vrai et l'élément de données *failedAttributes* sera vide. (Ce cas est considéré comme spécial parce que l'objet peut faire partie de ceux auxquels le client n'est pas même intéressé.) Le service essaiera d'exécuter dans l'ordre toutes les modifications indiquées dans la liste et continuera cet essai même si une modification échoue au sujet d'un attribut, auquel cas le nom de celui-ci est ajouté à la liste des échecs. Si le fanion *notFilterable* est faux et que l'élément de données *failedAttributes* est vide, le client sait que toutes les mises à jour ont été effectuées sur cet objet. Les nouveaux types contenus dans la valeur de retour sont les suivants:

```
struct UpdateResultsType {
        NameType        name;
        boolean         notFilterable;
        StringSetType   failedAttributes;
};

typedef sequence <UpdateResultsType> UpdateResultsSetType;
```

Le paramètre *howMany* indique au service combien de résultats d'objet il y a lieu d'inclure dans le premier lot de réponses. (La valeur zéro est admise, ce qui force le retour de tous les résultats par l'intermédiaire de l'itérateur.) Le paramètre de sortie *resultsIterator* est une référence à un objet itérateur qui peut être utilisé pour trouver des résultats additionnels dans des lots. Si tous les résultats ont été renvoyés par l'opération *scopedUpdate*, cette référence sera vide. Le client doit détruire cet objet lorsqu'il n'en a plus besoin. Il peut le supprimer avant que tous les résultats soient extraits.

7.4.1.5 Opération de suppression détectée

La signature IDL de l'opération de suppression détectée est la suivante dans le service d'opérations MOO évolué:

```
DeleteResultsSetType scopedDelete (
        in NameType baseName,
        in ScopeType scope,
        in FilterType filter,
        in LanguageType language,
        in boolean failuresOnly,
        in unsigned short howMany,
        out DeleteResultsIterator resultsIterator)
        raises (InvalidParameter,
```

```
                    InvalidFilter,
                    FilterComplexityLimit,
                    Itut_x780::ApplicationError);
```

Plutôt que d'accéder à des valeurs d'attribut, cette opération tente simplement de supprimer chaque objet du domaine qui satisfait au filtre.

Le fanion *failuresOnly* sert à indiquer si le client souhaite que le service renvoie des résultats pour tous les objets satisfaisant au domaine de visibilité et au filtre ou seulement pour les objets qui n'ont pas pu être supprimés. Etant donné que des notifications de suppression d'objet sont normalement envoyées, les clients peuvent souvent décider de ne recevoir de résultats que pour les objets qui n'ont pas pu être supprimés.

La valeur de retour énumère les noms de tous les objets avec deux fanions qui peuvent indiquer que l'objet n'a pas pu être supprimé. Le fanion *notFilterable* doit être vrai si le service d'opérations MOO a pu interagir avec l'objet pour déterminer s'il a satisfait au filtre. Le fanion *notDeletable* doit être vrai si l'objet a satisfait au filtre mais n'a pas pu être supprimé, soit en raison de sa politique de suppression soit parce qu'il a levé une exception. Un objet qui ne peut pas être évalué en fonction du filtre est étiqueté comme un cas spécial afin d'informer le client que c'est peut-être un objet dont il ne prévoit même pas la suppression.

```
struct DeleteResultsType {
        NameType                name;
        boolean                 notFilterable;
        boolean                 notDeletable;
};

typedef sequence <DeleteResultsType> DeleteResultsSetType;
```

Le paramètre *howMany* indique au service combien de résultats d'objet il y a lieu d'inclure dans le premier lot de réponses. (La valeur zéro est admise, ce qui force le retour de tous les résultats par l'intermédiaire de l'itérateur.) Le paramètre de sortie *resultsIterator* est une référence à un objet itérateur qui peut être utilisé pour trouver des résultats additionnels dans des lots. Si tous les résultats ont été renvoyés par l'opération *scopedDelete*, cette référence sera vide. Le client doit détruire cet objet lorsqu'il n'en a plus besoin. Il peut le supprimer avant que tous les résultats soient extraits.

Etant donné qu'un grand nombre d'objets ne peuvent pas être supprimés s'ils contiennent d'autres objets, le service doit commencer à supprimer les objets "feuilles" contenus dans le domaine de visibilité et à progresser vers l'objet "racine" dans les domaines fondés sur des relations de contenance. Lors de la suppression d'objets, le service d'opérations MOO doit toujours suivre les règles de suppression d'objet fondées sur la politique de suppression d'objet, comme décrit au § 7.3. Comme ces règles vont être appliquées à chacun des objets contenus dans le domaine de visibilité, l'effet sera toutefois différent, en allant de bas en haut, qu'un simple essai de suppression d'objet à la racine d'une sous-arborescence. De même, le service d'opérations MOO est de type "au mieux". Il est donc possible que certains des objets d'un sous-arbre détecté soient supprimés alors que d'autres ne le sont pas. Les règles qui doivent toujours être appliquées aux opérations de suppression détectée sont les suivantes:

1) aucun objet ne peut être laissé à l'abandon. Autrement dit, un objet ne peut pas être supprimé sans suppression de tous les objets (subordonnés) qu'il contient;

2) l'exécution d'une opération de suppression détectée sur un sous-arbre entier a comme résultat la suppression de tous les objets contenus dans ce sous-arbre, à moins qu'un objet n'ait une politique de suppression de type *notDeletable*, que l'objet ait levé une exception relative à l'opération de destruction, ou qu'un objet ait un subordonné qui n'est pas supprimable;

3) l'exécution d'une opération de suppression détectée sur une partie d'un sous-arbre nécessite l'évaluation de chacun des objets situés au plus bas niveau détecté, au moyen des règles de suppression du § 7.3. Si un objet situé au plus bas niveau détecté peut être supprimé conformément à ces règles, cet objet est supprimé ainsi que ses éventuels subordonnés. Si un

objet situé au plus bas niveau ne peut pas être supprimé, il ne l'est pas, non plus que l'un quelconque de ses objets supérieurs. D'autres objets contenus dans le domaine de visibilité peuvent cependant être supprimés si les règles de suppression le permettent. Le service remonte ensuite à la couche suivante, et ainsi de suite.

7.4.2 Langage de filtrage par défaut

Ce paragraphe décrit le langage de contraintes de filtrage par défaut qui doit être pris en charge par toutes les réalisations conformes du service d'opérations MOO. Ces réalisations peuvent prendre en charge d'autres grammaires de contrainte en plus de la grammaire décrite ici. Une opération est prévue à l'interface du service d'opérations MOO de base afin de permettre à un client d'extraire les langages pris en charge par un service. La grammaire utilisée dans une requête est indiquée par un paramètre à valeur de chaîne nommé "langage" pour chaque opération détectée. Une valeur "MOO 1.0" (avec un seul espace entre "MOO" et "1.0") doit indiquer qu'il s'agit de la grammaire décrite ici. (Une constante nommée *defaultLanguage*, ayant cette valeur, est fournie dans le module IDL.)

La grammaire par défaut prise en charge par chaque réalisation conforme doit être la grammaire de contrainte par défaut définie pour la version 1.0 du service de notification [4] avec les modifications décrites dans les paragraphes suivants. Noter qu'en suivant cette méthode, le cadre général règle la prise en charge des règles de comparaison (ou "règles d'homologie") par la grammaire de filtrage. De nouvelles règles (par exemple une correspondance de chaînes insensibles à la casse) ne pourront pas être ajoutées en même temps qu'un nouveau type de données ou d'attribut. En revanche, la grammaire devra être mise à jour si de nouvelles capacités sont requises.

7.4.2.1 Application du langage de contrainte aux attributs d'objet

La grammaire de contrainte par défaut du service de notification a introduit le jeton spécial "$" afin de désigner les variables actuelles d'événement et de durée d'exécution. Pour des opérations sur objets multiples, le jeton "$" doit désigner l'objet "actuel" ainsi que les variables de durée d'exécution. En d'autres termes, l'on peut considérer les services d'opérations MOO comme une sélection d'un ensemble d'objets sur la base du nom de base fourni et du paramètre de domaine fourni, puis comme l'application de l'expression de filtrage individuellement à chacun des objets de cet ensemble. L'objet "actuel" est celui pour lequel l'on évalue l'expression. Les exemples suivants décrivent l'utilisation du jeton "$":

$.administrativeState	Attribut d'état administratif de l'objet actuel.
$curtime	Variable "interne" nommée "curtime".

Les identificateurs qui suivent le caractère "$." (signe du dollar suivi d'un point) sont les noms des attributs de l'objet actuel tels qu'ils se trouvent dans l'objet de valeur défini pour le retour des attributs d'un objet. (Voir UIT-T X.780 les détails des attributs d'objet géré.) En d'autres termes, l'expression "$.administrativeState" se rapporte au membre nommé "administrativeState" dans le type de valeur renvoyé par un appel à l'opération *getAttributes*() relative à l'objet actuel. (L'on part du principe qu'un grand nombre de mises en œuvre du service d'opérations MOO utiliseront l'opération *getAttributes*() pour extraire les attributs d'un objet avant d'appliquer la valeur du filtre.)

Le langage de contraintes du service de notification, sur lequel le langage de contraintes du service d'opérations MOO est construit, possède un opérateur "dot" (".") qui peut servir à accéder aux membres individuels d'une structure de données, celle-ci pouvant être imbriquée dans une autre. Un identificateur comme celui-ci peut donc être utilisé pour accéder à une valeur contenue dans un attribut dont la structure de données est la suivante:

$.systemTimingSource.primaryTimingSource

Une comparaison avec un nom d'attribut qui n'est pas présent dans un objet donne toujours la valeur "faux". Par exemple, dans l'expression "(A == 0) || (A != 0)", s'il n'y a pas d'attribut nommé "A" dans l'objet, les deux comparaisons donneront la valeur "faux" et l'expression sera effectivement évaluée à

"faux". Le langage par défaut du service de notification prend effectivement en charge une opération "existe" qui peut servir à vérifier l'existence d'un attribut avant de l'inclure dans une comparaison. De même, une comparaison est toujours évaluée à "faux" si les types des opérandes ne correspondent pas. Dans l'exemple ci-dessus, si "A" est une chaîne, l'expression sera fausse.

On notera que la grammaire de contraintes par défaut du service de notification définit un ensemble de variables de durée d'exécution (qui peuvent plutôt être considérées comme des variables "internes" ou "prédéfinies") mais n'autorise pas l'insertion, dans des expressions de filtrage, de variables définies par l'utilisateur. Aucune variable interne n'est actuellement définie pour le service de détection et de filtrage et les variables définies par l'utilisateur ne sont pas prises en charge.

NOTE – Etant donné que le service de notification évalue les objets sur la base du nom de leurs attributs, il faut prendre des précautions lors de la définition de noms d'attribut (qui sont les noms des membres de l'ensemble d'objets de valeur d'attribut défini pour une interface). Un attribut de type *AdministrativeStateType* nommé "adminState" avec la valeur "débloqué" ne traversera pas un filtre d'expression "administrativeState == unlocked" parce que les noms ne concordent pas.

7.4.2.2 Prise en charge des expressions régulières

Le langage de contraintes par défaut du service de notification définit un opérateur de sous-chaîne fonctionnant comme suit: l'expression "A ~ B" est vraie si A est une sous-chaîne de B. Le langage de contraintes par défaut du service d'opérations MOO étend ce langage en définissant un opérateur de mise en correspondance d'expressions régulières. Le caractère "#" est désigné pour cet opérateur. De cette façon, l'expression "A # B" devient vraie si la structure d'expression régulière définie en A trouve une correspondance en B. Dans le présent cadre général, les expressions régulières sont de type "moderne" comme défini au § 2.8 de POSIX 1003.2 [11].

Une expression régulière est une structure qui décrit un ensemble de chaînes. L'inclusion de caractères spéciaux, appelés "métacaractères", permet de décrire un ensemble de chaînes au moyen d'une seule chaîne. La page du manuel relatif à la commande "grep" *(global regular expression print)* donne, dans la plupart des systèmes conformes à l'interface POSIX, une description complète des expressions régulières et de leur utilisation.

Une correspondance avec une expression régulière est ajoutée au langage de contraintes afin de satisfaire l'exigence de correspondance avec des sous-chaînes au début, au milieu ou à la fin d'une chaîne. Les expressions régulières de l'interface POSIX prennent en charge cette capacité au moyen de métacaractères qui représentent le début ou la fin d'une chaîne ("^" et "$"). La correspondance au milieu d'une chaîne est obtenue par exclusion de ces caractères de l'expression régulière. Il est certain que cette exigence pourrait avoir été satisfaite par simple adjonction d'une paire de métacaractères à la fonction de correspondance de chaîne. L'on a toutefois estimé que, comme la correspondance avec une expression régulière est assurée en tant qu'utilitaire dans les systèmes conformes à l'interface POSIX, il était logique de continuer à utiliser cette capacité qui ajoute au langage une correspondance enrichie entre structures plutôt que d'exiger des développeurs qu'ils mettent en œuvre une capacité spéciale offrant beaucoup moins de possibilités fonctionnelles.

Etant donné qu'aussi bien le langage de filtrage de notifications en chaînes et le langage d'expression régulière utilisent le caractère "\" pour l'échappement, il faut utiliser dans une chaîne de filtrage les caractères "\\" (qui seront convertis en "\" par l'analyseur de chaînes de filtrage) chaque fois que le caractère "\" doit être inclus dans l'expression régulière. Dans l'exemple suivant, le caractère "." d'une expression régulière peut remplacer n'importe quel caractère isolé. L'opération suivante peut être utilisée dans un filtre afin de s'assurer que le caractère mis en correspondance est bien un point ("."): 'itu\\.int' # $.domain_name.

7.4.2.3 Prise en charge des tests d'attributs à valeur d'ensemble ou à valeur de séquence

Les applications de gestion de réseau ont tendance à dépendre étroitement des attributs des objets gérés et ces attributs sont souvent, en réalité, des ensembles ou des séquences de valeurs. (Les ensembles sont différents des séquences du fait qu'ils ne devraient pas contenir d'éléments dupliqués

et que l'ordre n'a pas d'importance. Dans les séquences, les éléments dupliqués sont permis et l'ordre est important.) Le langage de contraintes par défaut du service de notification nécessite une extension afin de prendre en charge l'utilisation, dans les expressions de filtrage, d'attributs à valeur d'ensemble ou à valeur de séquence. Deux groupes d'extensions sont requis afin de permettre l'utilisation d'ensembles et de séquences: le premier permet d'inclure des ensembles et séquences de valeurs de littéraux dans des expressions de filtrage; le second définit des opérateurs pour les ensembles et séquences.

7.4.2.3.1 Ensembles et séquences de valeurs littérales

Les ensembles et séquences de valeurs littérales sont inclus dans les expressions de filtrage par insertion entre crochets d'une liste de valeurs littérales séparées par des virgules. Par exemple:

{1, 16, 21}	Ensemble ou séquence d'entiers
{5.2, 6.8, 7.01}	Ensemble ou séquence de nombres en virgule flottante
{'pomme', 'orange'}	Ensemble ou séquence de chaînes
{Critique, Majeure, Mineure}	Ensemble ou séquence de valeurs énumérées
{}	Ensemble ou séquence vide

Les valeurs littérales doivent toujours être du type "simple" défini pour le langage de contraintes par défaut du service de notification (valeur booléenne, courte, courte non signée, longue, longue non signée, en virgule flottante, double, caractère, caractère large, chaîne, chaîne large) ou être des valeurs énumérées. Toutes les valeurs d'un ensemble ou d'une séquence doivent être du même type.

Il est évident que dans ce langage orienté-contraintes, les ensembles et séquences de littéraux sont définis de la même façon, ce qui en fait correspond au cas des types d'attribut dans une interface CORBA. A la différence de certains autres langages syntaxiques d'interface, le langage IDL du groupe OMG n'a qu'une structure séquentielle, sans aucun type d'ensemble. C'est pour en tenir compte que différentes opérations sont définies pour les ensembles et séquences. Lorsqu'un opérateur de séquence est appliqué à une paire de séquences (de valeurs de littéraux ou d'attributs), ces séquences sont traitées comme étant vraies, c'est-à-dire que l'on tient compte de l'ordre des valeurs. Lorsque cependant deux séquences sont impliquées dans une opération sur un ensemble, ces séquences sont en fait traitées comme des ensembles, c'est-à-dire que l'ordre des valeurs dans l'ensemble n'a pas d'importance. De même, bien que les objets gérés ne doivent jamais renvoyer de copies dans la valeur d'un attribut à valeur d'ensemble, il convient de négliger d'éventuelles copies.

7.4.2.3.2 Opérateurs d'ensemble

Afin d'inclure des attributs à valeur d'ensemble dans des expressions de filtrage, il faut des opérateurs agissant sur des ensembles. Le présent paragraphe développe le langage de contraintes du service de notification en définissant la façon dont les opérateurs déjà définis pour ce service doivent être appliqués à des ensembles. Un seul opérateur nouveau, utilisant le symbole de l'accent circonflexe ('^') est défini pour tester l'intersection de deux ensembles. L'on définit également deux fonctions internes qui prennent des ensembles comme arguments.

Noter que le langage de contraintes par défaut du service de notification définit déjà un opérateur agissant sur des ensembles: l'opérateur "dans". L'expression "A dans B" ne peut être appliquée que si A est un type simple comme défini ci-dessus et B une séquence du même type simple. L'expression est évaluée comme étant vraie si la valeur représentée par A est égale à une valeur contenue dans B. De même, le langage de contraintes par défaut du service de notification prend en charge l'utilisation de l'opérateur "existe" dans des paramètres à valeur d'ensemble. Ce comportement sera également pris en charge pour des opérations sur objets multiples.

En général, afin d'utiliser l'un quelconque des opérateurs d'ensemble dans une expression telle que "A <opérateur> B", il faut qu'un des deux opérandes soit une séquence de l'un des types énumérés

ci-dessus dans le paragraphe relatif aux ensembles de valeurs littérales. Si un des opérandes est une séquence de type X, l'autre doit être soit une séquence de type X ou une valeur de type X. Etant donné qu'un des opérandes, ou les deux, sont en fait des séquences et non des ensembles, les opérations doivent négliger d'éventuelles valeurs dupliquées dans une séquence et ne doivent pas dépendre d'un ordre quelconque des valeurs dans une séquence. Les opérateurs étendus aux actions sur des attributs à valeurs d'ensemble sont définis ci-dessous:

A == B	Vrai si toutes les valeurs de chaque opérande sont présentes dans l'autre opérande.
A != B	Faux si toutes les valeurs de chaque opérande sont présentes dans l'autre opérande.
A < B	Vrai si toutes les valeurs dans A sont dans B et que B contienne au moins uneautre valeur non contenue dans A.
A <= B	Vrai si toutes les valeurs dans A sont dans B. (Si A est un attribut à valeur unique, cette expression est la même que "A dans B".)
A > B	Vrai si toutes les valeurs dans B sont dans A et que A contienne au moins uneautre valeur non contenue dans B.
A >= B	Vrai si toutes les valeurs dans B sont dans A.
A ^ B	Vrai si une valeur quelconque dans A est présente dans B (intersection nonvide).

En plus de ces opérations, deux fonctions internes, prenant un ensemble comme argument et renvoyant une seule valeur à partir de cet ensemble, sont définies:

MAX(<ensemble ou séquence>)	Renvoie la valeur la plus élevée de l'ensemble.
MIN(<ensemble ou séquence>)	Renvoie la valeur la moins élevée de l'ensemble.

Si aucun maximum ni aucun minimum ne peut être déduit de l'ensemble (parce que les valeurs ne sont pas numériques), la valeur renvoyée doit être indéterminée et toute comparaison avec cette valeur indéterminée doit être évaluée comme fausse.

7.4.2.3.3 Opérateurs de séquence

Afin d'inclure des attributs à valeur de séquence dans des expressions de filtrage, il faut des opérateurs agissant sur des séquences. Le présent paragraphe développe le langage de contraintes du service de notification en définissant la façon dont les opérateurs déjà définis pour ce service doivent être appliqués à des séquences.

Un seul opérateur est défini pour les séquences car la seule exigence est d'obtenir une correspondance d'égalité dans des séquences. Les opérateurs définis pour s'appliquer à des attributs à valeur de séquence sont les suivants:

A % B	Vrai si A et B ont le même nombre de valeurs et que toutes les valeurs dans A correspondent à celles qui sont dans B, dans l'ordre.

Les fonctions intégrées MAX et MIN, définies dans le paragraphe précédent, peuvent également être appliquées à des séquences.

7.4.3 Exigences du service d'opérations MOO

Ce paragraphe résume les exigences relatives au service d'opérations MOO.

(R) MOO-1 Un système géré doit instancier au moins un objet serveur d'opérations MOO. De même, chaque contexte de nommage à la racine locale d'un système doit avoir au moins une corrélation de noms pour un objet du service d'opérations MOO. La valeur de la chaîne *ID* dans cette corrélation doit simplement désigner le serveur, éventuellement avec une valeur comme "MOO1".

La chaîne *kind (sorte)* contenue dans la corrélation doit désigner la classe de l'objet ("itut_q816::BasicMooService" ou une sous-classe).

(R) MOO-2 L'interface prise en charge par l'objet (les objets) de serveur MOO doit être l'interface "de base" du service MOO décrite ci-dessus et définie dans l'Annexe A en langage IDL de l'architecture CORBA.

(O) MOO-3 En option, l'interface prise en charge par l'objet (les objets) de serveur MOO peut être l'interface "évoluée" du service MOO qui est décrite ci-dessus et définie dans l'Annexe A en langage IDL de l'architecture CORBA.

(R) MOO-4 L'objet (les objets) de serveur MOO doivent au moins prendre en charge le langage de contraintes par défaut qui est défini ci-dessus pour la spécification de filtres et peuvent prendre en charge d'autres grammaires. Le langage de contraintes par défaut, désigné par "MOO 1.0", est celui qui est défini par le service de notification CORBA mais étendu comme décrit ci-dessus pour prendre en charge:

- le filtrage de valeurs d'attribut d'objet plutôt que de valeurs de membre d'une structure de notification;
- la correspondance d'expressions régulières;
- le filtrage d'attributs contenant des ensembles ou des séquences de valeurs.

7.5 Service de pulsation

Le service de pulsation sert à vérifier le fonctionnement des canaux de notification dans un système géré, ainsi que le réseau de communications entre le système géré et le système gérant. Il envoie périodiquement une petite notification à un système gérant intéressé à la recevoir, qui identifie le système qui a émis la pulsation ainsi que le canal de notification par lequel cette émission a transité. Après la configuration de ce service, un système gérant peut créer un filtre de notifications de pulsation dans tout canal auquel il s'intéresse à garantir le fonctionnement. Comme ces notifications transitent par les mêmes canaux, les mêmes logiciels et les mêmes réseaux que les notifications issues d'autres ressources, le fonctionnement de celles-ci est périodiquement vérifié.

Le service de pulsation est détecté par recherche dans le service de nommage.

Le module IDL suivant (sans commentaires) décrit l'interface du service de pulsation:

```
interface Heartbeat {

        attribute SystemLabelType systemLabel;

        HeartbeatPeriodType periodGet()
        raises (itut_x780::ApplicationError);

        void periodSet(in HeartbeatPeriodType period)
        raises (itut_x780::ApplicationError);;

}; // fin de l'interface de pulsation

interface Notifications {
…
        void heartbeat (
                in SystemLabelType      systemLabel,
                in ChannelIDType        channelID,
                in HeartbeatPeriodType  period,
                in GeneralizedTimeType  timeStamp
        );
…
}; // fin de l'interface de notification
```

Comme on peut le voir, le service de pulsation comporte un attribut nommé *systemLabel* et des opérations de création et d'obtention de l'intervalle entre pulsations. L'attribut *systemLabel* est un

identificateur fourni par l'utilisateur, dont l'usage prévu est de permettre à un système gérant d'insérer une étiquette désignant le système qui fournit la pulsation.

Le service de pulsation émet périodiquement une notification dans chaque canal d'événements qu'il peut trouver dans le service détecteur de canaux. Celui-ci fournit une liste de tous les canaux du système, assortis chacun d'un identificateur, ainsi que des informations complémentaires sur l'utilisation de chaque canal. (Le détecteur de canaux ne doit pas énumérer les notifications de pulsation comme faisant partie des notifications qu'il manipule: les notifications de pulsation ne sont pas envoyées par des objets gérés mais sont envoyées à tous les canaux.) A la fin de chaque période, le service de pulsation envoie une notification dans chacun des canaux énumérés. La notification envoyée à chaque canal comporte l'identificateur du canal correspondant.

L'intervalle entre pulsations est commandé par l'opération *periodSet*. La valeur soumise pour cette opération est la période, en secondes, pendant laquelle le service de pulsation attend entre chaque émission de notification. La mise à jour de cette période provoque une émission immédiate, par le service, d'une notification comportant la nouvelle valeur d'intervalle et le début d'une nouvelle période. La mise à zéro de l'intervalle fait que le service émet une seule notification finale comportant une valeur d'intervalle égale à zéro, puis l'arrêt de ces émissions (jusqu'à réinitialisation de l'intervalle).

Chaque notification comporte la valeur de l'attribut *systemLabel*, l'identificateur du canal par lequel la notification a été envoyée, la valeur actuelle de la période et un marqueur temporel.

(R) HEARTBEAT-1 Un système géré peut instancier au moins un objet de service de pulsation. Si ce service est pris en charge, chaque contexte de nommage en racine locale d'un système doit avoir au moins une corrélation de noms pour un objet de service de pulsation. La valeur de la chaîne *ID* dans cette corrélation doit simplement identifier le serveur, avec une valeur comme "Heartbeat1". La chaîne *kind* dans la corrélation doit désigner la classe de l'objet ("itut_q816::Heartbeat").

(R) HEARTBEAT-2 Chaque objet serveur de pulsations doit prendre en charge l'interface de pulsation décrite ci-dessus et définie en langage IDL CORBA dans l'Annexe A. La capacité décrite ci-dessus doit être prise en charge.

(R) HEARTBEAT-3 La mise à jour de la période doit faire que le service achemine vers tous les canaux une notification contenant la nouvelle valeur de période puisqu'une nouvelle période commence. La mise à zéro de l'intervalle fait que le service émet une seule notification finale comportant une valeur d'intervalle égale à zéro, puis l'arrêt de ces émissions (jusqu'à réinitialisation de l'intervalle).

(R) HEARTBEAT-4 Les notifications de pulsation doivent être envoyées à tous les canaux une seule fois par période jusqu'à un changement de période. L'intervalle d'envoi des notifications de pulsation à un canal ne doit jamais être supérieur à deux fois la période.

7.6 Autres services supports

Le présent cadre général anticipe le besoin d'autres services supports de gestion de réseau mais reconnaît qu'il n'est pas pratique de les intégrer tous dans un même document cadre. La délimitation exacte est un peu arbitraire cependant. Etant centré sur le RGT et sur la nécessité de prendre en charge les modèles d'information existants, le présent cadre inclut des services qui sont équivalents à ceux qui sont fournis par le protocole CMIP et par les capacités d'information de gestion RGT les plus fondamentales. Comme dans le cas du protocole CMIP, l'on s'attend que de nouveaux services supports seront définis, très probablement dans des Recommandations distinctes.

8 Observance et conformité

Le présent paragraphe définit les critères qui doivent être satisfaits par les autres documents de normalisation revendiquant l'observance du présent cadre général. Il définit également les fonctions

qui doivent être mises en œuvre par les systèmes revendiquant la conformité à la présente Recommandation.

8.1 Conformité du système

8.1.1 Points de conformité

Le présent paragraphe résume les fonctions individuelles qui ont été décrites plus haut dans la présente Recommandation. Ces points de conformité seront ensuite combinés en profils qui devront être pris en charge par les systèmes revendiquant la conformité à la présente Recommandation.

1) Une réalisation revendiquant la conformité aux exigences du service de nommage doit:
   - prendre en charge la version du service de nommage CORBA spécifiée au § 5.2;
   - prendre en charge toutes les exigences du service de nommage spécifiées au § 6.1.
2) Une réalisation revendiquant la conformité aux exigences du service de notification doit:
   - prendre en charge:
     – soit la version du service de notification CORBA spécifiée au § 5.2;
     – soit l'interface NotificationIRPOperations du groupe 3GPP spécifiée dans la référence [13].
   - Prendre en charge toutes les exigences du service de notification spécifiées au § 6.2.

NOTE – Un complément d'étude est nécessaire pour identifier un sous-ensemble minimal de capacités de service de notification devant être pris en charge pour l'observance du cadre général.

3) Une réalisation revendiquant la conformité aux exigences du service de journalisation des télécommunications doit:
   - prendre en charge la version du service de journalisation des télécommunications CORBA spécifiée au § 5.2;
   - prendre en charge toutes les exigences du service de journalisation des télécommunications spécifiées au § 6.3;
4) Une réalisation revendiquant la conformité aux exigences du service de sécurité doit:
   - prendre en charge la version du service de sécurité CORBA spécifiée au § 5.2;
   - prendre en charge toutes les exigences du service de sécurité spécifiées au § 6.5.
5) Une réalisation revendiquant la conformité aux exigences du service de transaction doit:
   - prendre en charge la version du service de transaction CORBA spécifiée au § 5.2;
   - prendre en charge toutes les exigences du service de transaction spécifiées au § 6.6.
6) Une réalisation revendiquant la conformité aux exigences du service détecteur d'atelier doit:
   - prendre en charge l'interface du service détecteur d'atelier décrite au § 7.1 et définie dans l'Annexe A en langage IDL de l'architecture CORBA.
7) Une réalisation revendiquant la conformité aux exigences du service détecteur de canaux doit:
   - prendre en charge l'interface du service détecteur de canaux décrite au § 7.2 et définie dans l'Annexe A en langage IDL de l'architecture CORBA.
8) Une réalisation revendiquant la conformité aux exigences du service terminateur doit:
   - prendre en charge l'interface du service terminateur décrite au § 7.3 et définie dans l'Annexe A en langage IDL de l'architecture CORBA.

9) Une réalisation revendiquant la conformité aux exigences du service d'opérations MOO de base doit:

- prendre en charge les exigences obligatoires du service d'opérations MOO décrites au § 7.4.3.

10) Une réalisation revendiquant la conformité aux exigences du service d'opérations MOO évolué doit:

- prendre en charge les exigences obligatoires et facultatives du service d'opérations MOO décrites au § 7.4.3.

11) Une réalisation revendiquant la conformité au service de pulsation doit:

- prendre en charge l'interface du service de pulsation décrite au § 7.5 et définie dans l'Annexe A en langage IDL CORBA.

8.1.2 Profil de conformité de base

Un système revendiquant la conformité au profil de base UIT.Q.816 doit prendre en charge:

1) la version de l'architecture CORBA qui est spécifiée au § 5.2;
2) les exigences du service de nommage (voir le point de conformité 1 ci-dessus);
3) les exigences du service de notification (voir le point de conformité 2 ci-dessus);
4) le service détecteur d'atelier (voir point de conformité 6);
5) le service détecteur de canaux (voir point de conformité 7);
6) le service terminateur (voir point de conformité 8);
7) le service d'opérations MOO de base (voir point de conformité 9).

8.2 Directives de déclaration de conformité

Les utilisateurs du présent cadre général doivent, lors de la rédaction des déclarations de conformité, veiller à ce qui suit. Etant donné que les modules IDL vont être utilisés comme des espaces nominatifs, ils pourront, comme permis par les règles IDL du groupe OMG, être subdivisés entre plusieurs fichiers. Lorsqu'un module sera étendu, son nom ne change pas mais un nouveau fichier IDL sera simplement ajouté. Le simple fait de déclarer le nom d'un module dans une déclaration de conformité ne suffira donc pas pour désigner un ensemble d'interfaces en langage IDL. La déclaration de conformité doit désigner un document et son année de publication afin de garantir que la version correcte de la spécification IDL est bien désignée.

ANNEXE A

Spécification IDL des services supports du cadre

```
/* Ce code IDL est destiné à être mémorisé dans un fichier nommé "itut_q816.idl"
et situé dans le chemin de recherche utilisé par les compilateurs IDL de votre
système. */

#ifndef ITUT_Q816_IDL
#define ITUT_Q816_IDL

#include <CosTime.idl>
#include <itut_x780.idl>

#pragma prefix "itu.int"

module itut_q816 {
```

```
// Types imported from CosTime
typedef TimeBase::UtcT UtcT;

// Types imported from itut_x780
typedef itut_x780::AttributeSetType AttributeSetType;
typedef itut_x780::GeneralizedTimeType GeneralizedTimeType;
typedef itut_x780::NameType NameType;
typedef itut_x780::NameSetType NameSetType;
typedef itut_x780::ObjectClassType ObjectClassType;
typedef itut_x780::ObjectClassSetType ObjectClassSetType;
typedef itut_x780::ScopedNameType ScopedNameType;
typedef itut_x780::ScopedNameSetType ScopedNameSetType;
typedef itut_x780::StringSetType StringSetType;
typedef itut_x780::SystemLabelType SystemLabelType;

// Interfaces imported from itut_x780 (interfaces are not typedeffed
// for efficiency and clarity reasons)
// itut_x780::ManagedObject
// itut_x780::ManagedObjectFactory

// Exceptions imported from itut_x780 (exceptions can’t be typedeffed)
// itut_x780::ApplicationError
// itut_x780::DeleteError
```

// Types de données et structures

```
/** La structure ScopeChoice énumère quatre options pour un domaine de
visibilité. Celui-ci peut ne contenir que l'objet de base nommé, l'ensemble de la
sous-arborescence d'objets située au-dessous de l'objet de base y compris
celui-ci, les objets situés à un certain niveau au-dessous de l'objet de base (les
objets de niveau 1 sont directement contenus dans l'objet de base), ou tous les
objets en descendant jusqu'à un certain niveau, y compris l'objet de base et le
niveau.
*/
enum ScopeChoiceType {baseObjectOnly, wholeSubtree, individualLevel,
                      baseToLevel};

/** La structure Scope sert à indiquer les éventuels objets contenus au-dessous de
l'objet de base qui doivent être inclus dans le domaine d'une opération détectée
et filtrée. Un niveau n'est pas applicable aux options baseObjectOnly et
wholeSubtree mais il l'est pour les deux autres options. Pour illustrer la
différence avec les deux options qui comportent un niveau, une option de domaine
de type individualLevel de niveau = 1 comprendra tous les objets directement
contenus par l'objet de base. Une option de domaine de type baseToLevel de niveau
= 1 comprendra tous les objets directement contenus par l'objet de base, ainsi que
celui-ci.
*/

union ScopeType switch (ScopeChoiceType)
{
        /* Les cas baseObjectOnly et wholeSubtree n'acheminent aucune valeur. */
        case individualLevel:   /* transfert implicite */
        case baseToLevel:       short level;
};


/** La structure BaseAndScopeType combine le nom d'un objet géré de base avec un
"domaine" d'objets contenus au-dessous de cet objet de base. */

struct BaseAndScopeType {
        NameType        base;
        ScopeType       scope;
};

/** La structure BaseAndScopeSetType est un ensemble de types BaseAndScopeTypes.
*/

typedef sequence <BaseAndScopeType> BaseAndScopeSetType;
```

```
/** La structure ChannelIDType est une chaîne utilisée pour contenir un
identificateur de canal. */

typedef string ChannelIDType;


/** La structure EventSetType est une liste de types d'événement. C'est en fait
seulement une liste de chaînes. Les valeurs de ces chaînes sont les noms des types
d'événement (les chaînes qui entrent dans le champ "type_name" de l'événement
structuré). Ces noms sont les mêmes que les noms détectés de l'opération définie
aux interfaces de notification pour envoyer les événements. Par exemple:
itut_x780::Notifications::objectCreation  */

typedef sequence <ScopedNameType> EventSetType;

/** Une structure d'informations de canal contient des informations sur un canal
d'événements.

@member channelID                Identificateur de chaîne pour le canal.
@member channelClass             Nom de classe détecté du canal.
@member baseAndScopes            Objets et domaines d'objets gérés inférieurs
                                 qui envoient des événements à ce canal. Une
                                 liste vide indique que tous les objets gérés
                                 de base dans le système sont couverts par ce
                                 canal.
@member eventTypes               Liste des types d'événement manipulés par ce
                                 canal. Une liste vide indique que tous les
                                 types d'événement sont manipulés par ce
                                 canal.
@member excludedEventTypes       Si le paramètre eventTypes est vide, cette
                                 structure peut être utilisée afin d'exclure
                                 des types d'événement. Si le paramètre
                                 eventTypes n'est pas vide, cette structure
                                 doit être vide et être ignorée.
@member sourceClasses            Liste des types d'objets qui envoient des
                                 événements à ce canal. Une liste vide indique
                                 que tous les types d'objets gérés envoient
                                 des événements à ce canal.
@member excludedSourceClasses    Si le paramètre sourceClasses est vide, cette
                                 structure peut être utilisée afin d'exclure
                                 des classes de source. Si le paramètre
                                 sourceClasses n'est pas vide, cette structure
                                 doit être vide et être ignorée.
@member channel                  Référence du canal.
*/

struct ChannelInfoType {
        ChannelIDType           channelID;
        ObjectClassType         channelClass;
        BaseAndScopeSetType     baseAndScopes;
        EventSetType            eventTypes;
        EventSetType            excludedEventTypes;
        ObjectClassSetType      sourceClasses;
        ObjectClassSetType      excludedSourceClasses;
        Object                  channel;
};

/** Un ensemble informationnel de canal contient une liste de références de canal
et les données qui y sont associées. */

typedef sequence <ChannelInfoType> ChannelInfoSetType;

/** La structure ChannelModification indique le type de modification de canal
d'événements.  */

enum ChannelModificationType {ChannelCreate, ChannelDelete,
                              ChannelUpdate};
```

```
/** La structure DeleteResultsType contient, pour un objet donné, les résultats
d'une opération de suppression détectée. Si les deux fanions booléens du résultat
sont faux, l'objet a été supprimé.
@member name                    Nom de l'objet auquel les résultats
                                s'appliquent.
@member notFilterable           Ce fanion sera vrai si le service n'a pas pu
                                interagir avec l'objet pour voir s'il a
                                vraiment satisfait au filtre.
@member notDeletable            Ce fanion sera vrai si l'objet n'a pas pu être
                                supprimé en raison de sa politique de
                                suppression ou parce qu'il a levé une
                                exception.
*/
struct DeleteResultsType {
        NameType                name;
        boolean                 notFilterable;
        boolean                 notDeletable;
};

/** La structure DeleteResultsSetType est un ensemble de résultats renvoyés par
l'opération de suppression détectée.  */

typedef sequence <DeleteResultsType> DeleteResultsSetType;

/** Une structure informationnelle d'atelier contient des informations sur un
atelier d'objet géré.
@member factoryClass    Nom de la classe d'atelier détectée dans le domaine
@member factoryRef      Référence à l'atelier
*/

struct FactoryInfoType {
        ObjectClassType                 factoryClass;
        itut_x780::ManagedObjectFactory factoryRef;
};

/** Un ensemble informationnel d'atelier contient une liste de références
d'atelier avec leurs noms de classe. */

typedef sequence <FactoryInfoType> FactoryInfoSetType;

/** Un paramètre FilterType achemine l'expression de filtrage utilisée dans une
opération détectée et filtrée.
*/
typedef string FilterType;

/** Les structures GetResults contiennent une liste de valeurs d'attribut pour
chaque objet.
@member name                    Nom CORBA de l'objet
@member notFilterable           Ce fanion sera vrai si le service n'a pas pu
                                interagir avec l'objet pour voir s'il a
                                vraiment satisfait au filtre. S'il est vrai,
                                les attributs et les membres de type
                                failedAttributes seront vides.
@member attributes              Liste d'attributs extraits de l'objet
@member failedAttributes        Liste d'attributs dont la valeur n'a pas pu
                                être extraite de l'objet.
*/

struct GetResultsType {
        NameType                name;
        boolean                 notFilterable;
        AttributeSetType        attributes;
        StringSetType           failedAttributes;
};


/** La structure GetResultsSet est un ensemble de résultats renvoyés par une
opération d'obtention détectée. */

typedef sequence <GetResultsType> GetResultsSetType;
```

```
/** La structure HeartbeatPeriodType contient la longueur de l'intervalle entre
les pulsations émises par le service de pulsation. Le fait d'utiliser un nombre
court non signé pour désigner cet intervalle limite le plus long intervalle
possible à un peu plus de 18 h. */

typedef unsigned short HeartbeatPeriodType;

/** Le paramètre de type LanguageType achemine le code d'expression de filtrage
utilisé dans une opération détectée et filtrée.
*/

typedef string LanguageType;

/** Le paramètre LanguageSetType contient une séquence de langages. */

typedef sequence <LanguageType> LanguageSetType;

/**  La structure ModificationOp sert à indiquer le type de mise à jour à apporter
à un attribut. */

enum ModificationOpType {set, add, remove};

/** Les structures de modification désignent un attribut et une modification à lui
apporter. Plusieurs mises à jour peuvent être apportées à un même attribut par
inclusion de multiples structures avec le même nom d'attribut dans l'ensemble de
modification.
@member attrib          Nom de l'attribut à mettre à jour.
@member op              Opération à effectuer sur l'attribut.
@member val             Valeur à utiliser pour l'opération de mise à jour.
                Son type dépendra de l'attribut.
*/

struct ModificationType {
        string                  attrib; // nom de l'attribut
        ModificationOpType      op;     // opération à effectuer
        any                     value;  // valeur utilisée pour mettre à jour
                                                        l'attribut
};

/** La séquence de modification contient une séquence de modifications à apporter
(dans l'ordre) à chaque objet lors d'une opération de mise à jour détectée. */

typedef sequence <ModificationType> ModificationSeqType;

/** Les structures UpdateResults contiennent le nom d'un objet, un fanion
indiquant si toutes les modifications à cet objet ont été correctement effectuées,
et une liste des attributs qui n'ont pas pu être mis à jour pour cet objet. La
liste sera vide si le fanion de succès est vrai.
@member name                    nom CORBA de l'objet
@member notFilterable           Ce fanion sera vrai si le service n'a pas pu
                                interagir avec l'objet pour voir s'il a
                                vraiment satisfait au filtre. S'il est vrai, le
                                client ne connaîtra aucun attribut pouvant être
                                fixé, de sorte que le membre de type
                                failedAttributes sera vide.
@member failedAttributes        Liste d'attributs qui n'ont pas été
                                correctement mis à jour.

*/

struct UpdateResultsType {
        NameType        name;
        boolean         notFilterable;
        StringSetType   failedAttributes;
};

/** Un ensemble UpdateResults est renvoyé en réponse à une opération de mise à
jour détectée (opération qui pose, ajoute ou retranche par rapport à la valeur
d'un attribut). */

typedef sequence <UpdateResultsType> UpdateResultsSetType;
```

//Constantes

```
/** Le filtre par défaut laisse tout passer. */
const FilterType defaultFilter = "TRUE";
/** Le langage par défaut est la grammaire décrite dans la présente
Recommandation. */
const LanguageType defaultLanguage = "MOO 1.0";
```

// Exceptions

```
/**  Une exception de canal déjà inscrit est renvoyée si l'on tente d'inscrire un
canal avec de multiples identificateurs. */

exception ChannelAlreadyRegistered {};

/**  Une exception de canal non trouvé est renvoyée si un canal d'événements ne
peut pas être trouvé. */

exception ChannelNotFound {};

/** Une exception FactoryNotFound est levée lorsqu'un atelier demandé ne
peut pas être trouvé.  */

exception FactoryNotFound {};

/** Une exception FilterComplexityLimit est levée lorsqu'une expression de
filtrage est valide dans une opération détectée mais trop complexe pour être
traitée.  */

exception FilterComplexityLimit {};

/** Une exception InvalidFilter est levée lorsqu'un client inclut une expression
de filtrage qui ne peut pas être analysée. Le texte entourant l'erreur de syntaxe
doit être renvoyé aux fins de dépannage. */

exception InvalidFilter {string badText;};

/** Une exception InvalidParameter est levée lorsque la valeur d'un paramètre
n'est pas valide pour l'opération.
@param parameter        nom du paramètre erroné
*/

exception InvalidParameter {string parameter;};
```

// Interfaces

// Interface de détecteur d'atelier

```
/**
Cette interface définit un service simple pour localiser un atelier d'objet géré.
*/

interface FactoryFinder {

        /** Cette méthode sert à trouver un atelier d'objet géré.
        @param factoryClass     Nom de la classe détectée de l'atelier,
                                ce n'est PAS l'objet géré à créer.
        */

        itut_x780::ManagedObjectFactory find (in ObjectClassType factoryClass)
                raises (FactoryNotFound),
                itut_x780::ApplicationError);
```

```
        /** Cette méthode renvoie la liste des ateliers inscrits dans le détecteur
        d'atelier. */

        FactoryInfoSetType list()
        raises (itut_x780::ApplicationError);

};  // fin de l'interface FactoryFinder


/**
Cette interface développe l'interface FactoryFinder afin d'ajouter des méthodes de
prise en charge de l'inscription et de la désinscription des ateliers.
*/

interface FactoryFinderComponent : FactoryFinder {

        /** Cette méthode sert aux ateliers à s'inscrire. Elle ne devrait pas être
        utilisée par les systèmes gérants.
        @param factoryClass     Nom de classe détecté de l'atelier,
                                ce n'est PAS l'objet géré à créer.
        @param factoryRef       Référence à l'atelier.
        */

        void register (in ObjectClassType factoryClass,
                in itut_x780::ManagedObjectFactory factoryRef)
                raises (itut_x780::ApplicationError);

        /** Cette méthode sert aux ateliers à se désinscrire, si nécessaire. Elle
        ne devrait pas être utilisée par les systèmes gérants.
        @param factoryClass     Nom de classe détecté de l'atelier,
                                ce n'est PAS l'objet géré à créer.
        @param factoryRef       Référence à l'atelier.
        */

        void unregister (in ObjectClassType factoryClass,
                in itut_x780::ManagedObjectFactory factoryRef)
                raises (FactoryNotFound),
                itut_x780::ApplicationError;


};  // fin de l'interface FactoryFinderComponent
```

// Interface de détecteur de canaux

```
        /**
        Cette interface définit un service simple pour localiser les canaux d'événements.
        */

        interface ChannelFinder {

                /** Cette méthode renvoie la liste des canaux inscrits auprès du détecteur
                de canaux.  */

                ChannelInfoSetType list()
                raises (itut_x780::ApplicationError);

        };  // fin de l'interface ChannelFinder

        /**
        Cette interface développe l'interface ChannelFinder afin d'ajouter des méthodes de
        prise en charge de l'inscription et de la désinscription des canaux.
        */

        interface ChannelFinderComponent : ChannelFinder {

                /** Cette méthode sert aux canaux pour s'inscrire. Elle ne devrait pas
                être utilisée par les systèmes gérants. La réinscription d'un canal (par
                réutilisation d'un identificateur de canal existant) entraîne la mise à
                jour de cette entrée. Sinon, un canal peut être inscrit plusieurs fois
```

```
        (pour plusieurs objets de base). Les autres informations déjà associées à
        cette entrée sont écrasées. Une exception de type ChannelAlreadyRegistered
        peut être levée lorsque l'on essaie d'inscrire un canal avec de multiples
        identificateurs, mais ce n'est pas la bonne solution. (Le service ne peut
        pas garantir que deux objets, parce que leurs références diffèrent, ne
        font pas référence au même objet. Il est donc nécessaire que le système
        géré fasse en sorte que le même canal ne soit pas inscrit deux fois.) Une
        notification de changement de canal est envoyée chaque fois que l'appel de
        cette méthode se traduit par un changement.
@param channelID                        Identificateur de chaîne pour le
                                        canal.
@param channelClass                     Nom de classe détecté du canal
                                        d'événements.
@param baseAndScopes                    Objets et domaines d'objets gérés
                                        inférieurs qui envoient des
                                        événements à ce canal. Une liste vide
                                        indique que tous les objets gérés de
                                        base dans le système sont couverts
                                        par ce canal.
@param eventTypes                       Liste des types d'événement manipulés
                                        par ce canal. Une liste vide a une
                                        des implications suivantes:
                                        – pour les interfaces de canal
                                        d'événement qui ne fournissent pas
                                        une interrogation explicite pour les
                                        types d'événement manipulés par le
                                        canal (p. ex. le canal de
                                        notification OMG), cela implique que
                                        tous les types d'événement sont
                                        manipulés par ce canal;
                                        - pour les interfaces de canal
                                        d'événement qui fournissent bien une
                                        interrogation explicite pour les
                                        types d'événement manipulés par le
                                        canal (p. ex. interface
                                        NotificationIRPOperations du groupe
                                        3GPP), cela implique que
                                        l'interrogation explicite doit être
                                        utilisée pour déterminer les types
                                        d'événement manipulés.
@param excludedEventTypes               Si le paramètre eventTypes est vide,
                                        cette structure peut être utilisée
                                        afin d'exclure des types d'événement.
                                        Si le paramètre eventTypes n'est pas
                                        vide, cette structure doit être vide
                                        et être ignorée.

@param excludedSourceClasses            Si le paramètre sourceClasses est
                                        vide, cette structure peut être
                                        utilisée afin d'exclure des classes
                                        de source. Si le paramètre
                                        sourceClasses n'est pas vide, cette
                                        structure doit être vide et être
                                        ignorée.
@param channel                          Référence du canal.
        */

        void register (in ChannelIDType channelID,
                in ObjectClassType channelClass,
                in BaseAndScopeSetType baseAndScopes,
                in EventSetType eventTypes,
                in EventSetType excludedEventTypes,
                in ScopedNameSetType sourceClasses,
                in ScopedNameSetType excludedSourceClasses,
                in Object channel)
                raises (ChannelAlreadyRegistered,
                        itut_x780::ApplicationError);

        /** Cette méthode sert aux systèmes gérés à désinscrire des canaux, si
        nécessaire. Elle ne devrait pas être utilisée par les systèmes gérants.
        @param channelID        Identificateur de chaîne du canal.
```

```
        */

        void unregister (in ChannelIDType channelID)
                raises (ChannelNotFound, itut_x780::ApplicationError);

};  // fin de l'interface ChannelFinderComponent
```

// Interface du service de pulsation

```
/**
Cette interface définit un service utilisé pour contrôler périodiquement le
fonctionnement des canaux de notifications dans un système. Le service qui prend
en charge cette interface émet périodiquement une brève notification "de
pulsation" dans chaque canal du système.
*/
interface Heartbeat {

**/     L'attribut systemLabel est envoyé dans les notifications de pulsation. Il
sert à désigner l'instance du service de pulsation dont la notification provient.
Sa réinitialisation ne provoque pas une émission immédiate de notification par le
service mais la nouvelle valeur sera envoyée dans la prochaine notification. */

attribute SystemLabelType systemLabel;

**/     La période est l'intervalle, en secondes, auquel le service de pulsation
émet les notifications de pulsation. Si cet intervalle est égal à zéro, le service
n'émet pas de notifications. */

HeartbeatPeriodType periodGet()
raises (itut_x780::ApplicationError);

**/     La mise à jour de la période provoquera l'envoi par le service à tous les
canaux d'une notification avec la nouvelle valeur puis le début d'une nouvelle
période. La mise à zéro de la période se traduit par l'émission d'une seule
notification finale par le service avec une valeur de période égale à zéro, puis
l'arrêt des émissions (jusqu'à la rénitialisation de la période).
Les réalisations peuvent limiter l'étendue des valeurs prises en charge par cette
opération, en particulier vers le bas, car une trop grande fréquence de pulsations
présenterait une contrainte pour le système géré. Un essai de fixation de la
période à une valeur extérieure à l'étendue prise en charge se traduira par une
erreur d'application avec le code d'erreur mis à la valeur invalidParameter. */

void periodSet(in HeartbeatPeriodType period)
        raises(itut_x780::ApplicationError);

}; // fin de l'interface de pulsation
```

// Interface du service terminateur

```
/**
Cette interface définit un service qui prend en charge la suppression d'objets
gérés par des clients. Un des objectifs du cadre général est de permettre des
réalisations dans lesquelles les objets gérés n'ont pas besoin de conserver les
informations de l'arbre de nommage. Les ateliers sont un des lieux de mise en
œuvre des fonctions nécessaires pour créer des corrélations de noms et ce service
peut servir à nettoyer l'arbre de nommage après la suppression d'objets. <p>
De même, ce service peut appliquer les règles de suppression d'objets sur la base
de la politique de suppression des objets gérés.
*/

interface TerminatorService {

        /** Cette méthode sert à supprimer un objet géré en spécifiant son nom.*/

        void deleteByName (in NameType name)
                raises (itut_x780::ApplicationError,
                        itut_x780::DeleteError);
```

```
        /** Cette méthode sert à supprimer un objet géré par référence. */

        void deleteByRef (in itut_x780::ManagedObject mo)
                raises (itut_x780::ApplicationError,
                        itut_x780::DeleteError);

};  // fin de l'interface TerminatorService
```

// Interface d'itérateur de résultats de suppression

```
/** L'interface d'itérateur de résultats de suppression sert à extraire les
résultats issus d'une opération de suppression détectée au moyen de la structure
de conception de l'itérateur. */

interface DeleteResultsIterator {

        /** Cette méthode sert à extraire les prochains résultats "howMany"
        contenus dans l'ensemble de résultats. */

        @param howMany          Nombre maximal d'objets à renvoyer dans les
                                résultats.
        @param results          Lot suivant de résultats.
        @return                 Vrai s'il reste des résultats après ceux qui
                                ont été renvoyés. La valeur de retour ne
                                devrait pas être vraie si l'ensemble de
                                résultats est vide, car cela force le client à
                                rechercher les résultats. Au contraire, l'appel
                                de fonction devrait être bloquant.
        */

        boolean getNext(in unsigned short howMany,
                out DeleteResultsSetType results)
                raises (itut_x780::ApplicationError);


        /** Cette méthode sert à détruire l'itérateur et à libérer ses ressources.
        */

        void destroy();

};  // fin de l'interface d'itérateur de résultats de suppression
```

// Interface d'itérateur de résultats d'obtention

```
/** L'interface d'itérateur de résultats d'obtention sert à extraire les résultats
d'une opération d'obtention détectée au moyen de la structure de conception de
l'itérateur. */

interface GetResultsIterator {

        /** Cette méthode sert à extraire les prochains résultats "howMany"
        contenus dans l'ensemble de résultats.

        @param howMany          Nombre maximal d'objets à renvoyer dans les
                                résultats.
        @param results          Lot suivant de résultats.
        @return                 Vrai s'il reste des résultats après ceux qui
                                ont été renvoyés. La valeur de retour ne
                                devrait pas être vraie si l'ensemble de
                                résultats est vide, car cela force le client à
                                rechercher les résultats. Au contraire, l'appel
                                de fonction devrait être bloquant.
        */

        boolean getNext(in unsigned short howMany,
                out GetResultsSetType results)
                raises (itut_x780::ApplicationError);
```

```
        /** Cette méthode sert à détruire l'itérateur et à libérer ses ressources.
        */

        void destroy();

};  // fin de l'interface d'itérateur de résultats d'obtention
```

// Interface d'itérateur de résultats de mise à jour

```
/** L'interface d'itérateur de résultats de mise à jour sert à extraire les
résultats d'une opération de mise à jour détectée (poser, ajouter, retrancher) au
moyen de la structure de conception d'itérateur.
*/

interface UpdateResultsIterator {

        /** Cette méthode sert à extraire les prochains résultats "howMany"
        contenus dans l'ensemble de résultats.  */

        @param howMany          Nombre maximal d'objets à renvoyer dans les
                                résultats.
        @param results          Lot suivant de résultats.
        @return                 Vrai s'il reste des résultats après ceux qui
                                ont été renvoyés. La valeur de retour ne
                                devrait pas être vraie si l'ensemble de
                                résultats est vide, car cela force le client à
                                rechercher les résultats. Au contraire, l'appel
                                de fonction devrait être bloquant.
        */

        boolean getNext(in unsigned short howMany,
                out UpdateResultsSetType results)
                raises (itut_x780::ApplicationError);

        /** Cette méthode sert à détruire l'itérateur et à libérer ses ressources.
        */

        void destroy();

};  // fin de l'interface d'itérateur de résultats de mise à jour
```

// Interface du service d'opérations MOO de base

```
/** L'interface de détection et de filtrage de base offre un service commun pour
effectuer des opérations d'extraction d'attribut sur des objets multiples.
*/

interface BasicMooService {

        /** Cette opération sert à extraire la liste des langages de filtrage pris
        en charge par le service. Cette liste doit au moins comprendre la valeur
        de la constante defaultLanguage définie ci-dessus. */

        LanguageSetType getFilterLanguages()
                                raises (itut_x780::ApplicationError);

        /** Cette opération sert à extraire des attributs d'objets multiples au
        moyen d'un petit nombre d'invocations de méthode. La méthode renvoie le
        premier lot de résultats, un par objet. Chaque résultat contient le nom de
        l'objet et une liste de paires nom-valeur indiquant les attributs qui ont
        pu être extraits, avec leurs valeurs.

        @param baseName             Nom de l'objet situé à la base de l'arbre
                                    de visibilité.
        @param scope                Valeur indiquant les objets contenus dans
                                    le domaine de visibilité des opérations.
                                    Voir ScopeType pour les détails.
```

```
        @param filter               Chaîne contenant une expression à évaluer
                                    avec les valeurs d'attribut provenant de
                                    chaque objet du domaine. Les valeurs
                                    d'attribut ne sont renvoyées que pour les
                                    objets dont l'expression est vérifiée.
        @param language             Chaîne désignant le langage dans lequel
                                    l'expression de filtrage est écrite.
        @param attributes           Noms des attributs pour lesquels des
                                    valeurs doivent être renvoyées. Si cette
                                    liste est vide, tous les attributs
                                    doivent être renvoyés.
        @param howMany              Nombre maximal d'objets pour lesquels des
                                    résultats doivent être renvoyés dans le
                                    premier lot.
        @param resultsIterator      Référence à un itérateur qui peut servir
                                    à extraire le reste des résultats. Cette
                                    référence sera vide si tous les résultats
                                    ont été renvoyés dans le premier lot.
        */

        GetResultsSetType scopedGet (
                in NameType baseName,
                in ScopeType scope,
                in FilterType filter,
                in LanguageType language,
                in StringSetType attributes,
                in unsigned short howMany,
                out GetResultsIterator resultsIterator)
                raises (InvalidParameter,
                        InvalidFilter,
                        FilterComplexityLimit,
                        itut_x780::ApplicationError);

};  // fin de l'interface BasicMooService
```

// Interface du service évolué d'opérations MOO

```
/** L'interface évoluée de détection et de filtrage offre un service commun pour
effectuer des mises à jour sur attributs multiples d'objets multiples ou pour
supprimer des objets multiples.
*/

interface AdvancedMooService : BasicMooService {

        /** Cette opération sert à modifier des attributs multiples d'objets
        multiples au moyen d'un petit nombre d'invocations de méthode. La méthode
        renvoie le premier lot de résultats avec une liste d'objets pour lesquels
        une ou plusieurs modifications ont échoué. Chaque résultat indique
        l'attribut (les attributs) d'objet qui n'a (n'ont) pas pu être mis à jour.


        @param baseName             Nom de l'objet situé à la base de l'arbre
                                    de visibilité.
        @param scope                Valeur indiquant les objets contenus dans
                                    le domaine de visibilité des opérations.
                                    Voir ScopeType pour les détails.
        @param filter               Chaîne contenant une expression à évaluer
                                    avec les valeurs d'attribut provenant de
                                    chaque objet du domaine. Les mises à jour
                                    ne sont effectuées que sur les objets dont
                                    l'expression est vérifiée.
        @param language             Chaîne désignant le langage dans lequel
                                    l'expression de filtrage est écrite.
        @param modifications        Liste des modifications à apporter à
                                    chaque objet.
        @param failuresOnly         Si cette structure est vraie, seuls les
                                    résultats pour les objets défectueux
                                    seront renvoyés.
```

```
        @param howMany              Nombre maximal d'objets pour lesquels des
                                    résultats doivent être renvoyés dans le
                                    premier lot.
        @param resultsIterator      Référence à un itérateur qui peut servir à
                                    extraire le reste des résultats. Cette
                                    référence sera vide si tous les résultats
                                    ont été renvoyés dans le premier lot.
        */

        UpdateResultsSetType scopedUpdate (
                in NameType baseName,
                in ScopeType scope,
                in FilterType filter,
                in LanguageType language,
                in ModificationSeqType modifications,
                in boolean failuresOnly,
                in unsigned short howMany,
                out UpdateResultsIterator resultsIterator)
                raises (InvalidParameter,
                        InvalidFilter,
                        FilterComplexityLimit,
                        itut_x780::ApplicationError);

        /** Cette opération sert à supprimer des objets multiples au moyen d'un
        petit nombre d'invocations de méthode. La méthode renvoie le premier lot
        de résultats avec une liste des objets qui n'ont pas pu être supprimés.

        @param baseName             Nom de l'objet situé à la base de l'arbre
                                    de visibilité.
        @param scope                Valeur indiquant les objets contenus dans
                                    le domaine de visibilité des opérations.
                                    Voir ScopeType pour les détails.
        @param filter               Chaîne contenant une expression à évaluer
                                    avec les valeurs d'attribut provenant de
                                    chaque objet du domaine. Les mises à jour
                                    ne sont effectuées que pour les objets dont
                                    l'expression est vérifiée.
        @param language             Chaîne désignant le langage dans lequel
                                    l'expression de filtrage est écrite.
        @param failuresOnly         Si cette valeur est vraie, seuls seront
                                    renvoyés les résultats concernant des
                                    objets défaillants.
        @param howMany              Nombre maximal d'objets pour lesquels des
                                    résultats doivent être renvoyés dans le
                                    premier lot.
        @param resultsIterator      Référence à un itérateur qui peut servir à
                                    extraire le reste des résultats. Cette
                                    référence sera vide si tous les résultats
                                    ont été renvoyés dans le premier lot.
        */

        DeleteResultsSetType scopedDelete (
                in NameType baseName,
                in ScopeType scope,
                in FilterType filter,
                in LanguageType language,
                in boolean failuresOnly,
                in unsigned short howMany,
                out DeleteResultsIterator resultsIterator)
                raises (InvalidParameter,
                        InvalidFilter,
                        FilterComplexityLimit,
                        itut_x780::ApplicationError);

};  // fin de l'interface AdvancedMooService
```

// Interface de notification

```
        /**  L'interface de notification définit les notifications émises par les services
        cadres et non par les objets gérés proprement dits.
        */

        interface Notifications {

                /** La notification de changement de canal est spécial en ce sens qu'elle
                est émise par le cadre (le détecteur de canaux) et non par un objet géré.
                Elle signale l'adjonction, la suppression ou la modification d'un canal
                d'événements inscrit.

                @param channelModification          Indique si un canal a été ajouté,
                                                    supprimé ou mis à jour.
                @param channelInfo                  Fournit les informations sur le
                                                    canal affecté.
                */

                void channelChange (
                        in ChannelModificationType      channelModification,
                        in ChannelInfoType              channelInfo
                );

                /** Cette signature d'opération définit la notification émise par le
                service de pulsation.
                @param systemLabel              Valeur actuelle de l'attribut
                                                systemLabel du service de pulsation.
                @param channelID                Identificateur de chaîne pour le
                                                canal.
                @param period                   Valeur actuelle de l'attribut de
                                                période du service de pulsation.
                @param timeStamp                Heure de l'émission de la
                                                notification.
                */

                void heartbeat (
                        in SystemLabelType      systemLabel,
                        in ChannelIDType        channelID,
                        in HeartbeatPeriodType  period,
                        in GeneralizedTimeType  timeStamp
                );

                /** Ces constantes définissent le nom des notifications déclarées
                ci-dessus et visent à réduire les erreurs. */

                const string channelChangeTypeName =
                        "itut_q816::Notifications::channelChange";

                const string heartbeatTypeName =
                        "itut_q816::Notifications::heartbeat";

                /** Ces constantes définissent le nom des notifications déclarées
                ci-dessus et visent à réduire les erreurs.  */

                const string channelIDName = "channelID";
                const string channelModificationName = "channelModification";
                const string channelInfoName = "channelInfo";
                const string periodName = "period";
                const string systemLabelName = "systemLabel";
                const string timeStampName = "timeStamp";

        }; // fin de l'interface de Notifications


}; // fin du module itut_q816

#endif  // fin de la spécification #ifndef ITUT_Q816_IDL
```

ANNEXE B

Le langage de contrainte BNF

Le formalisme BNF présenté dans cette annexe est une extension du formalisme BNF utilisé par le langage de filtrage du service de notification du groupe OMG. Les extensions sont indiquées par les grands caractères gras ci-dessous.

B.1 Le langage de contrainte proprement dit en termes de jetons lexicaux

```
<constraint>:=/* empty */
        | <bool>
<preference>:=/* <empty> */
        | min <bool>
        | max <bool>
        | with <bool>
        | random
        | first
<bool>:= <bool_or>
<bool_or>:=<bool_or> or <bool_and>
        | <bool_and>
<bool_and>:=<bool_and> and <bool_compare>
        | <bool_compare>
<bool_compare>:=<expr_in> == <expr_in>
        | <expr_in> != <expr_in>
        | <expr_in> < <expr_in>
        | <expr_in> <= <expr_in>
        | <expr_in> > <expr_in>
        | <expr_in> >= <expr_in>
        | <expr_in>
        | <expr> ^ <expr> /* non-null set intersection test*/
        | <seq_factor> % <seq_factor>  /* true if sequence
                                  operands have identical values */
<expr_in>:=<expr_twiddle> in <Ident>
        | <expr_twiddle>
        | <expr_twiddle> in $ < Component>
<expr_twiddle>:=<expr> ~ <expr>
        | <expr>
        | <expr> # <expr>
<expr>:= <expr> + <term>
        | <expr> – <term>
        | <term>
<term>:= <term> * <factor_not>
        | <term> / <factor_not>
        | <factor_not>
<factor_not>:=not <factor>
        | <factor>
<factor>:= ( <bool_or> )
        | exist <Ident>
        | <Ident>
        | <Number>
        | – <Number>
        | <String>
        | TRUE
        | FALSE
        | + < Number>
        | exist $ < Component>
        | $ < Component>
        | default $ < Component>
        | MAX ( <seq_factor> )
        | MIN ( <seq_factor> )
        | <seq_literal>
<seq_factor> := <Ident>
        | <seq_literal>
<seq_literal> := { <factor_list> }
<factor_list> := /*empty*/
        |  <factor_list> , <factor>
        | <factor>
```

B.2 Formalisme BNF pour jetons lexicaux jusqu'aux problèmes de jeu de caractères

```
<Ident>:= <Leader> <FollowSeq>
     |\ < Leader> < FollowSeq>< Component> := /* empty */
     |. < CompDot>
     |<CompArray>
     |<CompAssoc>
     |<Ident> < CompExt> /* run-time variable */
< CompExt> := /* empty */
     |. < CompDot>
     |<CompArray>
     |<CompAssoc>
< CompDot>:=<Ident> < CompExt>
     |<CompPos>
     |<UnionPos>
     | _length
     |_d
     | _type_id
     | _repos_id
< CompArray>:=[ < Digits> ] < CompExt>
< CompAssoc>:=( < Ident> ) < CompExt>
< CompPos>:=<Digits> < CompExt>
< UnionPos>:=( < UnionVal> ) < CompExt>
< UnionVal> := /* empty */
     |<Digits>
     |- < Digits>
     |+ < Digits>
     |<String>

<FollowSeq>:=/* <empty> */
     | <FollowSeq> <Follow>
<Number>:=<Mantissa>
     | <Mantissa> <Exponent>
<Mantissa>:=<Digits>
     | <Digits> .
     | . <Digits>
     | <Digits> . <Digits>
<Exponent>:=<Exp> <Sign> <Digits>
<Sign>:= +
     |-<
Exp>:= E
     |e
<Digits>:=<Digits> <Digit>
     | <Digit>
<String>:=’ <TextChars> ’
<TextChars>:=/* <empty> */
     | <TextChars> <TextChar>
<TextChar>:=<Alpha>
     | <Digit>
     |<Other>
     |<Special>
<Special>:=\\
     |\’
```

B.3 Problèmes de jeu de caractères

Le précédent formalisme BNF était complet jusqu'aux non-terminaux <Leader>, <Follow>,<Alpha>, <Digit> et <Other>. Pour un jeu de caractères particulier, il faut définir les caractères qui constituent ces classes de caractères.

Chaque jeu de caractères que le service de courtage doit prendre en charge doit définir ces classes de caractères. La présente annexe définit ces classes de caractères pour le jeu des caractères ASCII.

```
<Leader>:=<Alpha>
<Follow>:=<Alpha>
     | <Digit>
     |_
<Alpha> is the set of alphabetic characters [A-Za-z]
<Digit> is the set of digits [0-9]
<Other> is the set of ASCII characters that are not <Alpha>, <Digit>, or <Special>
```

APPENDICE I

Scénarios d'interfonctionnement entre modèles utilisant le cadre UIT et modèles conformes aux spécifications du Forum ATM ou du Forum ADSL

I.1 Introduction

Le présent appendice décrit la façon dont des systèmes conçus pour utiliser les méthodes de ce cadre peuvent interfonctionner avec des systèmes conçus pour utiliser les méthodes spécifiées par le Forum ATM et par le Forum ADSL.

Les méthodes suivantes ont été utilisées pour définir des interfaces pour les objets gérés en architecture CORBA:

– un modèle à granularité fine possède une relation univoque entre instances d'interface CORBA [c'est-à-dire qu'elles ont leur propre référence d'objet interopérable (IOR)] et instances d'objet géré;

– un modèle à granularité de classe possède une seule interface CORBA pour chaque classe d'objets gérés. Un autre mécanisme (comme un nom d'objet géré placé dans cette interface sous la forme d'un paramètre d'entrée pour chaque opération) doit être pris en charge par l'interface à granularité de classe CORBA afin de permettre la gestion de chaque instance d'objet géré.

Une méthode dite *à granularité neutre* utilise une structure contenant à la fois le nom d'objet géré et la référence IOR utilisés pour donner accès à chaque objet géré. Noter que la méthode à granularité neutre, bien qu'imposant au client de transmettre l'instance d'objet géré sous la forme d'un paramètre pour chaque opération (c'est-à-dire en visibilité de granularité de classe pour le client), permet d'avoir dans le serveur une mise en œuvre à granularité soit de classe soit fine.

Le cadre général décrit dans la présente Recommandation fait appel, pour définir des objets gérés en architecture CORBA, à une méthode de granularité fine, assortie à une opération *attributesGet* fondée sur un type valeur associé à une sous-classe d'objet géré CORBA qui utilise l'héritage de type valeur pour étendre les éléments du type valeur associé à son hyperclasse.

Les spécifications du Forum ATM et du Forum ADSL contiennent la référence IOR pour la classe et le nom de l'objet géré sous la forme d'une liste appariée de paramètres pour les opérations (c'est-à-dire qu'elles sont à granularité neutre). Le client utilise le nom pour désigner l'entité spécifique et n'a pas besoin qu'il y ait une référence IOR distincte pour chaque entité. Ces spécifications utilisent également des structures (c'est-à-dire qu'elles n'emploient pas l'héritage) pour leurs opérations *attributesGet*.

I.2 Terminologie

Les termes suivants sont introduits aux fins de l'analyse:

– serveur à granularité neutre – Système géré qui met en œuvre des objets définis au moyen d'un modèle à granularité neutre avec l'architecture CORBA 2.1;

– serveur du cadre UIT – Système géré qui met en œuvre des objets définis au moyen du cadre général contenu dans la présente Recommandation, prenant donc en charge l'architecture CORBA 2.3;

– système gérant à granularité neutre – Client capable de gérer des objets CORBA définis au moyen d'un modèle à granularité neutre avec l'architecture CORBA 2.1;

– système gérant du cadre UIT – Client capable de gérer des objets CORBA définis conformément au cadre général contenu dans la présente Recommandation, prenant donc en charge l'architecture CORBA 2.3.

I.3 Scénarios d'interfonctionnement

I.3.1 Serveur à granularité neutre en migration vers serveur du cadre UIT

Lors d'une migration à partir d'un serveur à granularité neutre, un serveur du cadre UIT peut ajouter de nouvelles capacités; il doit cependant préserver les anciennes capacités se trouvant dans la version à granularité neutre.

Il convient que le nouveau serveur mette en œuvre une fonction d'adaptation qui devrait présenter les interfaces à granularité neutre aux systèmes gérants à granularité neutre existants. Une méthode de mise en œuvre peut faire appel à la délégation d'opérations invoquées au sujet d'objets à granularité neutre vers des objets construits conformément au cadre général. Une telle approche par délégation émettra les mêmes opérations au sujet des objets individuels que celles qui seraient invoquées par un système gérant conforme au cadre UIT.

Il convient de convertir les paramètres d'opération d'*obtention* dans chaque classe *de tous les attributs* à granularité neutre en types valeurs structurés conformément au cadre décrit dans la présente Recommandation.

Certaines fonctions doivent être remplies par le logiciel d'interfonctionnement au-delà des différences résultant de l'emploi d'un adaptateur POA et de l'objet valeur. Le logiciel de délégation nécessite une spécialisation afin de tenir compte des différences de structure de nommage, y compris l'utilisation du champ *de sorte* dans le présent cadre général. Par exemple pour expliquer la façon dont la référence d'objet contextuel, utilisée lors de la construction du nom, doit être remplacée par le champ de sorte de la structure de nommage des services COS.

I.3.2 Client à granularité neutre en migration vers client du cadre UIT

Un tel client du cadre UIT a besoin de gérer aussi bien des serveurs à granularité neutre que des serveurs du cadre UIT.

Un client du cadre UIT en architecture CORBA 2.3 a besoin d'utiliser l'arbre de nommage non seulement pour les serveurs du cadre UIT mais aussi pour les serveurs à granularité neutre. La mise en œuvre de l'architecture CORBA 2.3 peut nécessiter une fonction d'adaptation dans laquelle l'application émettant la requête d'objet doit être adaptée au serveur à granularité neutre. Si l'application utilise le type valeur, elle doit être décomposée en opération d'obtention propre à l'objet pour l'ancien serveur.

Pour qu'un client conforme au cadre UIT gère à la fois les serveurs du cadre UIT et les serveurs antérieurs à ce cadre, ce client aura besoin de prendre en charge des interfaces d'objets distants aussi bien pour les serveurs UIT que pour les serveurs préUIT. Par ailleurs, une certaine fonction d'interfonctionnement pourra être nécessaire, comme analysé ci-dessus et indiqué dans la Figure I.1.

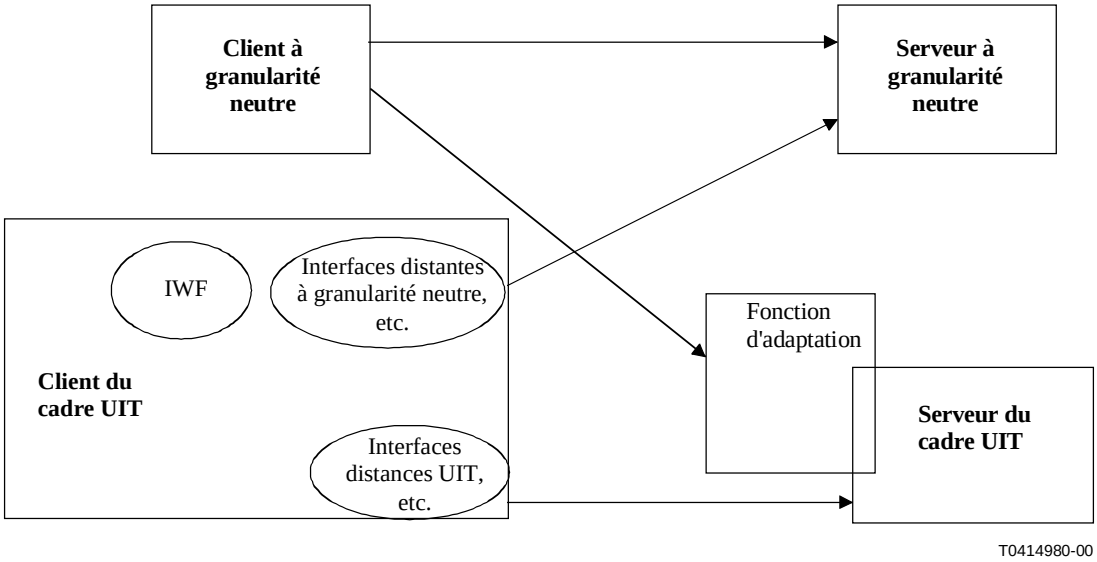

Figure I.1/Q.816 – Scénarios d'interfonctionnement

APPENDICE II

Evénements structurés de filtrage, natifs ou convertis

La présente Recommandation définit des notifications utilisant des signatures d'opération IDL, avec le type d'événement comme nom d'opération et avec un paramètre d'opération *in* distinct pour chaque élément de données contenu dans la notification. Ces signatures d'opération peuvent servir à expédier directement des notifications dans un canal au moyen de consommateurs mandataires typés, comme défini dans le service de notification de l'OMG et comme décrit dans la Figure II.1 ci-dessous [flèche numérotée (1)].

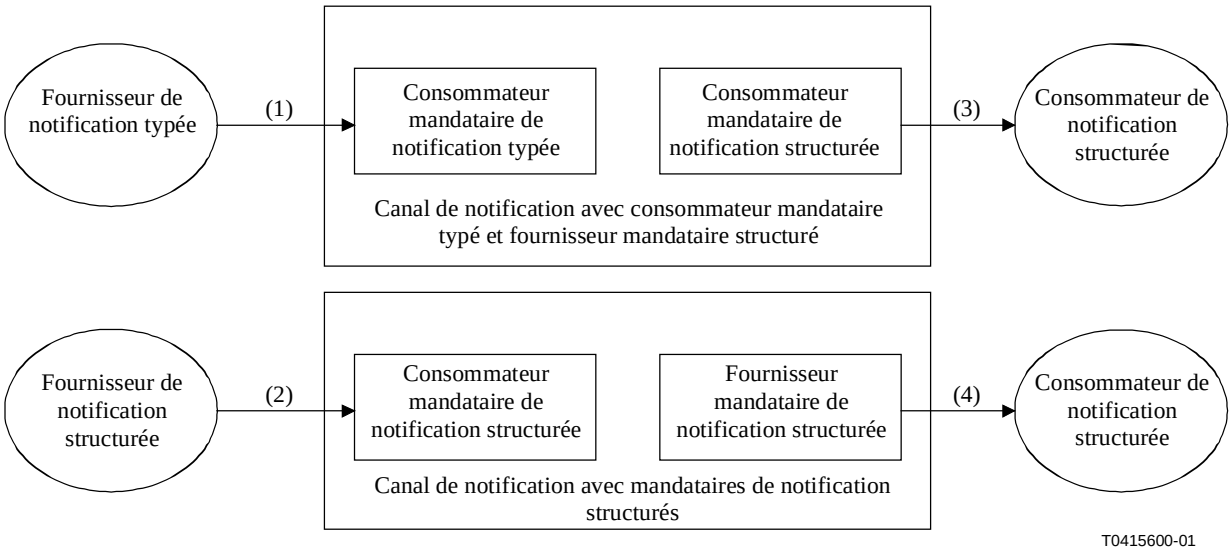

Figure II.1/Q.816 – Exemple d'expédition de notifications typées et structurées avec remise structurée

Etant donné que les événements structurés sont largement pris en charge, la présente Recommandation fournit également un algorithme afin de définir un compte rendu d'événement

structuré pour chacune des notifications définies. Ce format ainsi défini est utilisé pour les notifications structurées nativement, qui sont expédiées par des fournisseurs à des consommateurs mandataires structurés au moyen de canaux, comme représenté par la flèche numérotée (2) sur la Figure II.1. Ce format sert également pour la remise non convertie, comme représenté par la flèche numérotée (4) sur la Figure II.1.

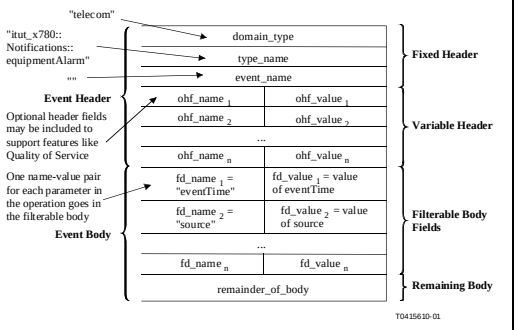

Figure X-2 – Notification nativement structurée

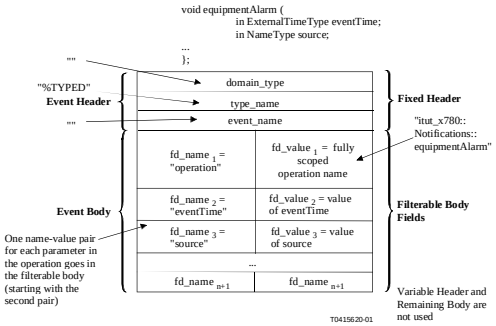

Figure X-3 – Notification convertie de typée en structurée

Ces événements structurés nativement sont définis comme suit (voir le résumé de la Figure 6) à partir des définitions de notification typée:

1) l'élément de données domain_type est mis à "Télécommunications";
2) l'élément de données type_name est mis au nom complet de l'opération définissant le type d'événement, p. ex. "itut_X780::Notifications::communicationsAlarm";
3) l'élément de données event_name est mis à la valeur de chaîne vide;
4) les éléments nominatifs des paires nom-valeur du champ de corps filtrable sont réglés de façon à indiquer le nom de chaque paramètre (p. ex. "eventTime", "source", "sourceClass", etc.). Le n^{e} élément nominatif contient le nom du n^{e} paramètre;
5) les éléments de valeur des paires nom-valeur du champ de corps filtrable sont réglés de façon à indiquer la valeur qui a été transmise au paramètre correspondant.

Un exemple d'événement structuré natif, correspondant au mappage normalisé et appliquant les règles X.780 normales, ressemblerait à ce qui suit:

```
domain_type = "Telecommunications"
type_name = "<fullyScopedOperationName>"
event_name = ""
fd_name1 = "<arg 1>"    fd_value1 = <arg1 value>
fd_name2 = "<arg 2>"    fd_value2 = <arg2 value>
fd_name3 = "<arg 3>"    fd_value3 = <arg3 value>
```

Le service de notification du groupe OMG définit des procédures permettant aux canaux de notification de convertir automatiquement les événements typés reçus des fournisseurs (voir flèche numérotée (1) sur la Figure II.1) en notifications structurées pour remise aux consommateurs (voir flèche numérotée (2) sur la Figure II.1). Ces procédures sont résumées comme suit:

1) l'élément de données domain_type est mis à la valeur de chaîne vide;
2) l'élément de données type_name est mis à la chaîne "%TYPED";
3) l'élément de données event_name est mis à la valeur de chaîne vide;
4) l'élément nominatif de la première paire nom-valeur du champ de corps filtrable est mise à la chaîne "operation";

5) l'élément de valeur de la première paire nom-valeur du champ de corps filtrable est mis à une chaîne contenant le nom d'opération complètement détecté. En l'occurrence, ce sera par exemple: "itu_X780::Notifications::communicationsAlarm";

6) les éléments nominatifs restants des paires nom-valeur du champ de corps filtrable sont réglés de façon à indiquer le nom de chaque paramètre (p. ex. "eventTime", "source", "sourceClass", etc.). Le n^e élément nominatif contient le nom du n^e – 1 paramètre;

les éléments de valeur restants des paires nom-valeur du champ de corps filtrable sont réglés de façon à indiquer la valeur qui a été transmise pour chaque paramètre (par exemple la valeur du paramètre eventTime, celle du paramètre source, celle du paramètre sourceClass, etc.). Le n^e élément de valeur contient la valeur du n^e – 1 paramètre.

L'emploi, pour un événement typé et converti, du même exemple que ci-dessus pour les notifications structurées natives permettra à un canal de distribuer des éléments comme les suivants (où fd_namei et fd_valuei correspondent au nom et à la valeur du i^e élément de la liste de paires nom-valeur filtrables):

```
domain_type = ""
type_name = "%TYPED"
event_name = ""
fd_name1 = "operation"  fd_value1 = "<fullyScopedOperationName>"
fd_name2 = "<arg 1>"    fd_value2 = <arg1 value>
fd_name3 = "<arg 2>"    fd_value3 = <arg2 value>
fd_name4 = "<arg 3>"    fd_value4 = <arg3 value>
```

Noter que le nom d'opération complètement détecté est inséré comme premier élément de la liste de paires nom-valeur filtrables et que chaque paramètre *in* de l'opération d'événement typé est un élément subséquent.

On trouvera ci-après un exemple de construction d'une structure filtrante utilisant des variables d'exécution permettant de contrôler le type d'événement `itut_x780::Notifications::attributeValueChange` et fonctionnant pour les deux formes de l'événement structuré (natif et typé puis converti par le canal) ci-dessus:

```
$typename == "%TYPED" and
$operation == "itut_x780::Notifications::attributeValueChange"
```

or

```
$typename == "itut_x780::Notifications::attributeValueChange"
```

APPENDICE III

Bibliographie

Les Recommandations suivantes contiennent des informations qui ont été utilisées lors de l'élaboration du présent cadre général. Comme indiqué dans l'introduction, un objectif essentiel de conception du présent cadre consiste à permettre la réutilisation de modèles existants d'informations de gestion de réseau, au moins sans modifications sémantiques notables. Ces Recommandations fournissent beaucoup de détails sur le cadre CMIP de l'UIT-T et définissent donc une partie des capacités que le cadre CORBA doit prendre en charge.

– UIT-T M.3010 (2000), *Principes des réseaux de gestion des télécommunications*.

– UIT-T M.3120 (2001), *Modèle générique d'informations de réseau en architecture CORBA*.

– UIT-T Q.821 (2000), *Description des étapes 2 et 3 pour l'interface Q3 – Supervision des alarmes*.

– UIT-T X.703 (1997), *Technologies de l'information – Architecture de gestion répartie ouverte*.

– UIT-T X.710 (1997) | ISO/CEI 9595:1998, *Technologies de l'information – Interconnexion des systèmes ouverts – Service commun d'information de gestion*.

– UIT-T X.711 (1997) | ISO/CEI 9596-1:1998, *Technologies de l'information – Interconnexion des systèmes ouverts – Protocole commun d'information de gestion: spécification*.

– UIT-T X.711/Cor.2 (2000), *Technologies de l'information – Interconnexion des systèmes ouverts – Protocole commun d'information de gestion: spécification – Corrigendum technique 2: passage à la notation ASN.1: 1997.*

– UIT-T X.720 (1992) | ISO/CEI 10165-1:1993, *Technologies de l'information – Interconnexion des systèmes ouverts – Structure des informations de gestion: modèle d'information de gestion.*

– UIT-T X.720/Cor.1 (1994), *Technologies de l'information – Interconnexion des systèmes ouverts – Structure des informations de gestion: modèle d'information de gestion – Corrigendum technique 1.*

– UIT-T X.721 (1992) | ISO/CEI 10165-2:1992, *Technologies de l'information – Interconnexion des systèmes ouverts – Structure des informations de gestion: définition des informations de gestion*.

– UIT-T X.721/Cor.1 (1994), *Technologies de l'information – Interconnexion des systèmes ouverts – Structure des informations de gestion: définition des informations de gestion – Corrigendum technique 1.*

– UIT-T X.721/Cor.2 (1996), *Technologies de l'information – Interconnexion des systèmes ouverts – Structure des informations de gestion: définition des informations de gestion – Corrigendum technique 2.*

– UIT-T X.722 (1992) | ISO/CEI 10165-4:1992, *Technologies de l'information – Interconnexion des systèmes ouverts – Structure des informations de gestion: directives pour la définition des objets gérés.*

– UIT-T X.722/Amd.1 (1995), *Technologies de l'information – Interconnexion des systèmes ouverts – Structure des informations de gestion: directives pour la définition des objets gérés – Amendement 1: propriété "Set by create" et enregistrement des composants.*

– UIT-T X.722/Amd.2 (1997), *Technologies de l'information – Interconnexion des systèmes ouverts – Structure des informations de gestion: directives pour la définition des objets gérés – Amendement 2: ajout de l'élément syntaxique "NO-MODIFY" et directives complémentaires*

– UIT-T X.722/Amd.3 (1997), *Technologies de l'information – Interconnexion des systèmes ouverts – Structure des informations de gestion: directives pour la définition des objets gérés – Amendement 3: directives pour l'utilisation du langage Z dans la formalisation du comportement des objets gérés.*

– UIT-T X.722/Cor.1 (1996), *Technologies de l'information – Interconnexion des systèmes ouverts – Structure des informations de gestion: directives pour la définition des objets gérés – Corrigendum technique 1.*

– UIT-T X.722/Cor.2 (2000), *Technologies de l'information – Interconnexion des systèmes ouverts – Structure des informations de gestion: directives pour la définition des objets gérés – Corrigendum technique 2: révision de la notation GDMO pour l'adapter à l'ASN.1: 1997.*

– UIT-T X.733 (1992) | ISO/CEI 10164-4:1992, *Technologies de l'information – Interconnexion des systèmes ouverts – Gestion-systèmes: fonction de signalisation des alarmes.*

SÉRIES DES RECOMMANDATIONS UIT-T

Série A	Organisation du travail de l'UIT-T
Série B	Moyens d'expression: définitions, symboles, classification
Série C	Statistiques générales des télécommunications
Série D	Principes généraux de tarification
Série E	Exploitation générale du réseau, service téléphonique, exploitation des services et facteurs humains
Série F	Services de télécommunication non téléphoniques
Série G	Systèmes et supports de transmission, systèmes et réseaux numériques
Série H	Systèmes audiovisuels et multimédias
Série I	Réseau numérique à intégration de services
Série J	Réseaux câblés et transmission des signaux radiophoniques, télévisuels et autres signaux multimédias
Série K	Protection contre les perturbations
Série L	Construction, installation et protection des câbles et autres éléments des installations extérieures
Série M	RGT et maintenance des réseaux: systèmes de transmission, circuits téléphoniques, télégraphie, télécopie et circuits loués internationaux
Série N	Maintenance: circuits internationaux de transmission radiophonique et télévisuelle
Série O	Spécifications des appareils de mesure
Série P	Qualité de transmission téléphonique, installations téléphoniques et réseaux locaux
Série Q	**Commutation et signalisation**
Série R	Transmission télégraphique
Série S	Equipements terminaux de télégraphie
Série T	Terminaux des services télématiques
Série U	Commutation télégraphique
Série V	Communications de données sur le réseau téléphonique
Série X	Réseaux de données et communication entre systèmes ouverts
Série Y	Infrastructure mondiale de l'information et protocole Internet
Série Z	Langages et aspects généraux logiciels des systèmes de télécommunication

Genève, 2001