UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

UIT-T

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

X.722

Amendement 3
(08/97)

SÉRIE X: RÉSEAUX POUR DONNÉES ET COMMUNICATION ENTRE SYSTÈMES OUVERTS

Gestion OSI – Structure de l'information de gestion

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

Recommandation UIT-T X.722 – Amendement 3

(Antérieurement Recommandation du CCITT)

RECOMMANDATIONS UIT-T DE LA SÉRIE X

RÉSEAUX POUR DONNÉES ET COMMUNICATION ENTRE SYSTÈMES OUVERTS

RÉSEAUX PUBLICS POUR DONNÉES	
Services et fonctionnalités	X.1–X.19
Interfaces	X.20–X.49
Transmission, signalisation et commutation	X.50–X.89
Aspects réseau	X.90–X.149
Maintenance	X.150–X.179
Dispositions administratives	X.180–X.199
INTERCONNEXION DES SYSTÈMES OUVERTS	
Modèle et notation	X.200–X.209
Définitions des services	X.210–X.219
Spécifications des protocoles en mode connexion	X.220–X.229
Spécifications des protocoles en mode sans connexion	X.230–X.239
Formulaires PICS	X.240–X.259
Identification des protocoles	X.260–X.269
Protocoles de sécurité	X.270–X.279
Objets gérés de couche	X.280–X.289
Tests de conformité	X.290–X.299
INTERFONCTIONNEMENT DES RÉSEAUX	
Généralités	X.300–X.349
Systèmes de transmission de données par satellite	X.350–X.399
SYSTÈMES DE MESSAGERIE	X.400–X.499
ANNUAIRE	X.500–X.599
RÉSEAUTAGE OSI ET ASPECTS SYSTÈMES	
Réseautage	X.600–X.629
Efficacité	X.630–X.639
Qualité de service	X.640–X.649
Dénomination, adressage et enregistrement	X.650–X.679
Notation de syntaxe abstraite numéro un (ASN.1)	X.680–X.699
GESTION OSI	
Cadre général et architecture de la gestion-systèmes	X.700–X.709
Service et protocole de communication de gestion	X.710–X.719
Structure de l'information de gestion	**X.720–X.729**
Fonctions de gestion et fonctions ODMA	X.730–X.799
SÉCURITÉ	X.800–X.849
APPLICATIONS OSI	
Engagement, concomitance et rétablissement	X.850–X.859
Traitement transactionnel	X.860–X.879
Opérations distantes	X.880–X.899
TRAITEMENT RÉPARTI OUVERT	X.900–X.999

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

NORME INTERNATIONALE 10165-4

RECOMMANDATION UIT-T X.722

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

Résumé

Le présent amendement à la Rec. X.722 du CCITT | ISO/CEI 10165-4 contient un exemple qui illustre le meilleur usage actuel du langage de description formelle Z pour spécifier le comportement des objets gérés. Il vise à établir une base et une compréhension communes de cette méthode formelle particulière, qui favoriseront la cohérence des réalisations analogues. Il devrait fournir un utile point de départ aux utilisateurs de GDMO qui souhaitent utiliser le langage Z pour améliorer leurs spécifications de comportement.

La valeur des spécifications de comportements d'objets gérés tient à ce qu'elles soient claires et sans ambiguïté. Elaborer une spécification formelle force à analyser de près les détails du comportement. Cela peut donc servir aussi d'outil pour identifier et corriger les imprécisions d'une spécification qui demeurera en langage naturel.

Le présent amendement contient un guide technique d'application du langage Z dans la définition du comportement des objets gérés utilisés pour l'interfonctionnement de la gestion OSI. De nature descriptive, il n'est pas normatif. Il ne requiert pas de techniques de définition formelle (FDT) pour spécifier le comportement des objets gérés. Si les techniques FDT doivent être employées, il n'est pas nécessaire d'utiliser le langage Z; d'autres langages, tels que le SDL, conviennent également.

Source

La Recommandation X.722, Amendement 3, de l'UIT-T a été approuvée le 9 août 1997. Un texte identique est publié comme Norme internationale ISO/CEI 10165-4.

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.

La Conférence mondiale de normalisation des télécommunications (CMNT), qui se réunit tous les quatre ans, détermine les thèmes d'études 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 n° 1 de la CMNT.

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

NORME INTERNATIONALE

RECOMMANDATION UIT-T

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

1) Table des matières

Ajouter à la table des matières la référence suivante:

Annexe B – Directives pour l'utilisation du langage Z dans la formalisation du comportement des objets gérés

2) Paragraphe 2.1

Ajouter la référence suivante:

– Recommandation X.731 du CCITT (1992) | ISO/CEI 10164-2:1992, *Technologies de l'information – Interconnexion des systèmes ouverts – Gestion-systèmes: fonction de gestion d'états*.

3) Nouveau paragraphe 2.3

Ajouter un nouveau paragraphe comme suit:

2.3 Autres Références

– ISO/CEI 13568:[1)], *Technologies de l'information – Langage de spécification Z*.

[1)] Actuellement à l'état de projet.

4) Nouvelle Annexe B

Ajouter une nouvelle Annexe B comme suit:

Annexe B

Directives pour l'utilisation du langage Z dans la formalisation du comportement des objets gérés

(Cette annexe ne fait pas partie intégrante de la présente Recommandation | Norme internationale)

B.1 Introduction

La présente annexe est un guide technique d'application du langage Z dans la définition du comportement des objets gérés utilisés pour l'interfonctionnement de la gestion OSI. De nature descriptive, elle n'est pas normative. Elle ne requiert pas de techniques de définition formelles (FDT, *formal definition techniques*) pour spécifier le comportement des objets gérés. Si les techniques FDT doivent être employées, il n'est pas nécessaire d'utiliser le langage Z; d'autres langages, tels que le SDL, conviennent également. Même s'il est indiqué d'utiliser le langage Z, on peut spécifier le comportement des objets gérés par d'autres moyens.

La spécification formelle du comportement des objets gérés a une utilité évidente car elle est claire et univoque. Comme elle oblige à examiner de près le détail des comportements, elle peut aussi servir d'outil pour déceler des imprécisions qui auraient pu passer inaperçues dans une spécification faisant uniquement usage du langage naturel, et pour y remédier. C'est pourquoi elle permet d'améliorer la spécification des comportements.

La présente annexe contient, à titre indicatif, un exemple montrant la meilleure manière de procéder actuelle. Elle vise à établir une base et une compréhension communes de cette approche formelle particulière qui contribuera à l'uniformité dans les réalisations similaires. Elle sera un point de départ utile pour les utilisateurs des directives GDMO qui souhaitent utiliser le langage Z pour améliorer la spécification des comportements.

Elle est destinée aux utilisateurs qui sont familiarisés avec les concepts de base de la spécification des objets gérés au moyen des modèles GDMO et du langage Z.

Dans la suite de la présente annexe, les expressions «objet géré» et «MO» seront utilisées pour désigner une définition de classe d'objets gérés faite au moyen des modèles GDMO.

B.2 Questions de langage

Le langage Z est une notation de spécification formelle fondée sur la théorie des ensembles et des calculs fonctionnels. Sa capacité d'expression est suffisante pour décrire des classes uniques d'objets gérés.

Toutefois, elle ne contient aucune notion d'encapsulage. Une spécification en langage Z est généralement constituée d'un modèle d'un certain état et d'une suite d'opérations pour modifier cet état. Le langage Z n'incorpore aucune méthode permettant de réunir l'état et ses opérations dans un module unique pouvant être réutilisé dans une autre spécification. La conséquence de cela apparaît quand il faut décrire des objets gérés qui ont des variables et un comportement hérités d'autres définitions de classe d'objets gérés.

Cet effet d'héritage peut être obtenu par la technique de l'inclusion de schémas, au prix d'un peu de clarté. A tout autre égard, le langage Z convient pour exprimer des classes particulières d'objets gérés.

B.3 Ce qu'il y a lieu de traduire

Les définitions d'un comportement ou de parties de celui-ci doivent être traduites en langage Z à partir de leur description informelle. La mesure dans laquelle les parties restantes des modèles GDMO doivent être formalisées dépend dans une large mesure des besoins du spécificateur.

Les modèles GDMO contiennent une définition semi-formelle des types de données en ASN.1. Il est possible d'écrire une spécification en langage Z en utilisant ces définitions en ASN.1 comme base pour les types utilisés dans la spécification en langage Z, ce qui permet d'économiser beaucoup de travail.

Toutefois, si une spécification est écrite de cette manière, le spécificateur aura davantage de travail pour s'assurer qu'elle est syntaxiquement correcte. Sans spécification en langage Z des définitions en ASN.1, on ne peut pas utiliser les outils en Z existants, qui permettent de contrôler la syntaxe et la sémantique statique d'une spécification en Z.

En résumé, on peut améliorer les définitions de comportement en utilisant le langage Z sans récrire les types de données ASN.1, mais on peut obtenir un avantage significatif en traduisant entièrement en langage Z les types de données en ASN.1. Le paragraphe B.7.1 donne des exemples de la manière de convertir en langage Z les types de base écrits en ASN.1.

B.3.1 Des modèles GDMO au langage Z

Le présent paragraphe contient les directives générales sur la manière de traduire un objet géré en langage Z à partir de sa description informelle, telle qu'elle est donnée dans la présente Recommandation | Norme internationale. Il convient de souligner d'emblée qu'une telle traduction ne peut être faite qu'informellement étant donné qu'une traduction formelle nécessiterait, pour le moins, que les langages source et cible soient tous deux formels.

De plus, comme c'est invariablement le cas lorsque l'on met en correspondance deux langages distincts, il y a inévitablement une certaine discordance entre leurs structures. Le problème s'aggrave lorsque l'un des langages est informel ou qu'il comporte des composantes informelles.

Le présent paragraphe contient la liste de quelques-unes des principales caractéristiques des modèles définis dans la présente Recommandation | Norme internationale, avec les points sur lesquels ils diffèrent des structures en langage Z ou y correspondent. Elles sont accompagnées de méthodes générales pour résoudre les discordances ou de conseils sur la manière de les traiter individuellement suivant les circonstances.

La présente annexe est orientée sur ce qui est nécessaire pour décrire le comportement d'un objet géré. Des informations supplémentaires sur la manière de convertir des types ASN.1 sont données au paragraphe B.6.

B.3.2 Types de données

La première étape consiste à récrire sous la forme de types en langage Z les types de données de la présente Recommandation | Norme internationale. L'ASN.1 fournit les moyens habituels de typage des données mais ses constructeurs sont orientés vers la description des flux de données communiqués entre les systèmes.

En ASN.1, les constructeurs de types sont définis comme des formes de liste. En Z, les types sont des ensembles. Bien qu'il soit possible de modéliser les constructeurs de types ASN.1 comme des séquences Z, il est parfois plus simple d'envisager les opérations disponibles sur les types ASN.1 et de les mettre en correspondance avec les types Z qui décrivent plus clairement leur structure. Les types séquence ASN.1 et les types ensemble peuvent être représentés par des nuplets Z. Le type séquence-de ASN.1 peut être représenté par une séquence Z. Le type ensemble-de ASN.1 peut être représenté par un ensemble Z.

L'ASN.1 inclut une prise en charge particulière du codage, comme les étiquettes de types et les valeurs par défaut. Elles n'ont pas à être représentées en Z, n'ayant pas d'incidence sur la définition du comportement.

Le paragraphe B.6.2 offre des informations supplémentaires sur la conversion des types ASN.1.

B.3.3 Attributs d'objets gérés

Par définition, les objets gérés ont certains attributs de gestion; ceux-ci ont un type de données défini en ASN.1 et des identificateurs d'objet leur sont attribués. Ils peuvent en outre avoir une propriété de correspondance. Deux manières de modéliser de tels attributs ont été proposées:

- les types d'attribut simples;
- les types d'attribut ayant la forme de schémas.

La solution la plus simple consiste à représenter l'attribut d'objet géré dans l'objet géré sous forme de variable Z avec le type de données approprié. Parallèlement, il faut définir une constante qui représente l'identificateur d'objet de cet attribut. Cette constante sera liée à l'attribut en question par convention seulement. On peut recourir à la propriété de correspondance en question si des opérations de mise en correspondance pour cet attribut ont été définies. Un exemple en est donné au B.6.3.

On peut aussi encapsuler toutes les propriétés de l'attribut dans un type de schéma unique qui sera alors le type de la variable Z modélisant l'attribut d'objet géré. Le schéma inclura alors la valeur de l'attribut aussi bien que l'identificateur d'objet et la propriété de correspondance éventuelle. Un exemple en est donné au B.6.4. Là où d'autres règles de correspondance que l'égalité sont requises, on peut définir le paramètre de correspondance (*matches-for*) comme une relation Z sur le type de la valeur de l'attribut. Cela permet la représentation formelle de règles de correspondance particulières arbitraires, qui peut être importante dans la validité, le filtrage et la sélection d'objets.

Il est difficile de modéliser en langage Z le type ANY de l'ASN.1. Un cas où le problème est fréquent est celui dans lequel il faut donner une liste de valeurs d'attribut. En conséquence, un modèle entièrement formel nécessitera sans doute un type exempt de langage Z réunissant les types d'attribut qui sont déjà définis. On en trouvera un exemple au B.6.1 et au B.6.5.

Les identificateurs d'objet sont formellement modélisés par un ensemble donné.

[OBJECTID]

B.3.4 Autres identificateurs d'objet

Beaucoup d'éléments autres que les attributs ont également un identificateur d'objet. Il est utile de les introduire tous sous la forme de constantes dans des définitions axiomatiques. On leur attribuera par convention le suffixe «Oid». Généralement, de telles constantes seront nécessaires pour les classes, pour les lots de propriétés (paquetages) et pour les notifications.

Exemple:

packagesPackageOid : OBJECTID
allomorphsPackageOid : OBJECTID
topClassOid : OBJECTID

B.3.5 Héritage et compatibilité

On peut utiliser le langage Z pour construire des hiérarchies d'héritage d'objets gérés en introduisant des schémas pour modéliser l'héritage et la spécialisation. Cela modélise correctement le comportement d'une classe d'objets gérés et de ses sous-types mais ne rend pas explicite la forte relation de sous-typage qui est effectivement présente. A cet effet, il faut un langage qui modélise l'héritage de manière explicite.

La langage Z peut donc être utilisé pour définir de manière satisfaisante les objets gérés individuels; mais, pour être en mesure de traiter de l'héritage et de la compatibilité, il faut disposer de la puissance additionnelle d'un langage pouvant modéliser l'héritage de manière explicite.

L'héritage n'est pas pris en compte dans le langage Z. Il peut être modélisé simplement par l'introduction de schémas d'état.

La définition de l'héritage d'objets gérés nécessite que les sous-classes soient compatibles; mais, en langage Z, il n'est pas indispensable que les sous-classes soient des sous-types, ce qui est regrettable. Donc, généralement, un objet géré peut annoncer sa classe réelle. Etant donné que l'attribut de classe réelle signale toujours la classe réelle de l'objet, une sous-classe ne peut signaler la classe d'une superclasse. Pour cette raison, une sous-classe ne peut pas afficher le même comportement que sa superclasse par le renvoi de la valeur de son attribut de classe réelle (c'est-à-dire qu'elle n'est pas substituable), même si son comportement est allomorphique. Pour cette raison, la sous-répartition des classes d'objets gérés n'est pas équivalente aux sous-types du langage Z, dans lequel un sous-type afficherait le même comportement que son supertype. Néanmoins, une sous-classe affiche très peu de comportements «non substituables».

On constate ainsi que l'héritage d'objets gérés, tel qu'il est défini dans la présente Recommandation | Norme internationale, permet à un parent d'avoir un comportement spécifique qui ne concorde pas avec le comportement de ses descendants. Etant donné qu'un tel comportement non substituable est très rare, on peut représenter une classe d'objets gérés par deux spécifications de classe. L'une est le comportement que toute instance ou objet géré étendu doit afficher; l'autre est une spécialisation qui capture le comportement affiché uniquement par des instances de la classe compatible et non par une quelconque extension. C'est cette dernière spécification qui est instanciée pour donner le comportement complet d'une instance d'objet géré réelle.

B.3.6 Lots (de propriétés)

De nombreuses parties de la fonctionnalité d'une classe peuvent être présentes dans certains objets gérés individuels et non dans d'autres. La présente Recommandation | Norme internationale décrit ce processus en groupant les fonctionnalités dans des lots conditionnels. Ensuite, chaque objet géré instancie les lots appropriés. En langage Z, on ne peut pas donner la fonctionnalité de cette manière conditionnelle, mais on peut faire en sorte que le comportement de l'objet géré dépende de la nature des lots instanciés. Cela est logique vu qu'un objet géré doit contenir un lot appelé par un attribut de gestion qui énumère les identificateurs d'objet des lots effectivement instanciés. Donc, en modélisant le comportement dans un lot conditionnel, ce comportement lui-même devient conditionnel par la présence de l'identificateur de lot dans l'attribut de lots.

B.3.7 Classe

Pour définir une classe d'objets gérés, il faut représenter ses attributs et ses opérations. Les attributions deviennent des parties du schéma d'état Z et les opérations deviennent des schémas d'opération Z.

B.3.7.1 Attributs

Les attributs d'un objet géré sont déclarés dans un schéma d'état. On donne à chaque attribut un type, qui peut être un type déclaré dans la partie ASN.1 du modèle GDMO ou un type déclaré en langage Z dans un modèle entièrement formel.

B.3.7.2 L'opération Get (obtenir)

Le gestionnaire peut demander qu'une opération Get soit effectuée sur un objet géré. La définition des éléments CMISE d'une opération M-Get a de nombreux paramètres dont la plupart concernent toutefois la commande d'accès, la sélection d'objet et ainsi de suite. Dans ce cas, l'opération Get pourra être modélisée à la limite de l'objet géré, sans tenir compte de ces questions et en remplaçant l'opération Get unique par une série d'opérations Get<name>, où <name> est un attribut unique.

B.3.7.3 L'opération GetAll

L'opération GetAll, qui est dépourvue d'entrée, est également modélisée. Elle renvoie un ensemble non vide de valeurs d'attribut.

B.3.7.4 Les opérations Replace (remplacer)

L'opération CMISE M-Set demande une opération Set sur un objet géré individuel. Cette spécification modélise les opérations Replace telles qu'elles sont vues à la limite de l'objet géré. Dans cette spécification, les opérations Replace se réfèrent aux attributs «operations set», «set to default», «add» et «remove».

La conséquence de ce qui précède est la spécification d'un schéma Z pour représenter chaque modification.

B.3.7.5 Notifications

Les notifications sont des messages non sollicités qui sont envoyés par un objet géré pour rendre compte d'événements internes. Elles ne sont toutefois pas modélisées comme des opérations mais comme des résultats d'opérations sur l'objet géré. Donc, toute opération (qu'elle soit invoquée par le gestionnaire ou, de manière interne, par la ressource) peut produire un résultat. Si celui-ci entraîne une notification, celle-ci doit faire partie du résultat de l'opération.

Cela signifie que le résultat d'un schéma d'opération Z pouvant entraîner des notifications doit être un *ensemble* de notifications. Dès lors, on peut représenter les cas dans lesquels il n'émet pas de notification en donnant pour résultat un ensemble vide.

Les données contenues dans une notification sont constituées d'un type EventType qui est l'identificateur d'objet de sa définition standard. Celui-ci est suivi de diverses informations s'appliquant à cette notification particulière. L'identificateur d'objet peut généralement être défini comme une constante et les données particulières comme un type de schéma. Le comportement de la notification est inclus dans tout objet qui a la capacité de produire la notification.

B.3.7.6 Actions

Les actions sont des opérations effectuées par le gestionnaire sur l'objet géré. En langage Z, elles sont représentées tout naturellement par des opérations.

B.3.8 Spécification du système d'objets

Le reste de la présente annexe décrit la manière de représenter le comportement d'un objet unique. Lorsqu'on aborde la création ou la suppression d'objets, les corrélations de leur nom, leur inclusion et leur dénomination, il faut décrire l'état du système où réside l'objet. On peut représenter la création ou la suppression d'un objet par un changement d'état de ce système. On peut représenter la corrélation des noms et l'inclusion par une relation avec l'ensemble d'objets. La dénomination peut alors être définie à partir de cette relation.

B.4 Exemple

Dans le présent paragraphe, on trouvera des exemples de définitions des attributs «top» et «State Management» d'une classe d'objets gérés. Etant donné que le principal objectif du présent guide est de modéliser le comportement, la création de types Z à partir de types ASN.1 n'est pas présentée dans le présent paragraphe. Une définition formelle complète est donnée au B.7.

B.4.1 top (sommet)

La première classe qu'il y a lieu de définir est *top*, qui est l'ascendant ultime (le sommet de la hiérarchie d'héritage) de tous les objets gérés.

La classe *top* a quatre attributs de gestion, à savoir *objectClass, packages, allomorphs* et *nameBinding*. L'attribut *objectClass* contient l'identificateur d'objet de la classe, l'attribut *packages* contient les identificateurs d'objet des blocs qu'il instancie, l'attribut *nameBinding* contient l'identificateur d'objet de la correspondance de noms utilisée pour instancier l'objet et l'attribut *allomorphs* contient les identificateurs d'objet des classes pour lesquelles l'objet peut être allomorphique. Etant donné que les attributs de gestion peuvent former des blocs, les attributs présents dans les objets gérés d'une classe donnée peuvent varier. Cela est modélisé par l'inclusion d'un attribut de modélisation additionnel appelé *attributes*, qui contient des identificateurs d'objet des attributs qui sont réellement instanciés dans l'objet géré individuel. On notera que tous les attributs présents dans la classe *top* sont fixes pendant la durée de vie de tout objet géré individuel.

Le langage Z ne modélise pas explicitement les interfaces et pour cette raison il n'est pas possible de définir formellement celles des opérations qui sont invoquées de manière interne et celles qui le sont de manière externe, par le gestionnaire.

TopState

allomorphs: $\mathbb{P}$ *OBJECTID*
objectClass: OBJECTID
nameBinding: OBJECTID
packages: $\mathbb{P}$ *OBJECTID*
attributes: $\mathbb{P}$ *OBJECTID*

{objectClassOid, nameBindingOid} $\mathbb{F}$ *attributes*
allomorphsPackageOid $\mathfrak{P}$ *packages* $\cup$ *allomorphsOid* $\mathfrak{P}$ *attributes*
packagesPackageOid $\mathfrak{Q}$ *packages*
packages $\mathbb{P}$ $\rightarrow$ $\cup$ *packagesOid* $\mathfrak{P}$ *attributes*

L'élément *attributes* n'est pas un attribut d'objet géré mais une nouvelle composante d'état, définie pour des raisons de commodité, qui énumère les attributs d'objet géré contenus dans cet objet géré. Donc l'invariant l'oblige à contenir les identificateurs d'objet des attributs appropriés décrits au B.3.3 (et définis au B.7.4). Les attributs *objectClass* et *nameBinding* sont obligatoires; l'attribut *packages* est présent si un lot enregistré donné est instancié séparément de *packagesPackage*. Dans ce cas, cela signifie *allomorphsPackage*.

L'opération *TopGetNameBinding* interroge l'objet géré et retourne la valeur de l'attribut *nameBinding* sans changer l'état *TopState*. L'opération *TopGetNameBinding* est invoquée par le gestionnaire.

TopGetNameBinding

Ξ *TopState*
result!: OBJECTID

---

result! = nameBinding

Les opérations *TopGetAllomorphs, TopGetObjectClass* et *TopGetPackages* n'ont pas été définies ici. On notera qu'il n'y a pas d'opération pour obtenir *attributes* étant donné que ce n'est pas un attribut réel d'objet géré tel que spécifié dans le modèle GDMO.

L'opération *TopGetAll* obtient toutes les valeurs d'attribut d'un objet. Elle retourne toujours des valeurs pour *objectClass* et *nameBinding*. En cas de présence de lots conditionnels ou d'allomorphismes, il les obtient aussi. L'opération *TopGetAll* est invoquée par le gestionnaire.

TopGetAll

✠ *TopState* *result!:* ℗ *AttributeValues*
 # attributes = # result! *ObjectClassValue objectClass ⩤ result!* *NameBindingValue nameBinding ⩤ result!* *PackagesOid ⩤ attributes ∪ packagesValue packages ⩤ result!* *AllomorphsOid ⩤ attributes ∪ allomorphsValue allomorphs ⩤ result!*

L'opération *TopEventReport* est une façon de modéliser les notifications qui survient spontanément et représente la manière dont les rapports d'événement ne sont pas sous le contrôle du gestionnaire.

TopEventReport

✠ *TopState* *notification!: EventInfo*

B.4.2 StateManagement Class

Cette classe ne reflète aucune classe d'objets gérés spécifique mais le comportement de tout objet qui inclut un certain attribut standard tel que *administrativeState*, *operationalState* ou *usageState*. Il est plus pratique, dans ce contexte, de considérer cette inclusion comme un héritage; elle n'est pas un exemple utile.

Le schéma d'état inclut les définitions *TopState* et les prédicats. Il définit quelques variables et conjonctions de prédicats additionnelles.

StateManagementState

TopState *administrativeState:* *AdministrativeState* *operationalState: OperationalState* *usageState: UsageState*
operationalState = disabled ∪ usageState = idle *administrativeState = locked ∪ usageState = idle* *usageState = idle ∪ administrativeState ≠ shuttingDown*

La gestion d'état hérite des opérations depuis le sommet *Top*. Bien que le langage Z n'ait pas de mécanisme pour les opérations d'héritage, il est très simple de redéfinir les opérations en termes de nouvel état. La partie prédicat de l'état *StateManagementState* découle de la définition de la fonction de gestion d'état donnée dans les Rec. X.721 du CCITT (1992) | ISO/CEI 10165-2:1992 et X.731 du CCITT (1992) | ISO/CEI 10164-2:1992.

On peut aisément définir l'opération *SMGetNameBinding* étant donné qu'elle n'a pas d'effet sur les nouvelles variables d'état déclarées dans l'état *StateManagementState*. On peut réutiliser la définition de l'opération *TopGetNameBinding*:

SMGetNameBinding

TopGetNameBinding ✠ *StateManagementState*

Les définitions des opérations visant à obtenir d'autres attributs de l'état *StateManagementState* ont également été omises dans cet exemple. Les opérations *SMGetAllomorphs*, *SMGetObjectClass* et *SMGetPackages* peuvent réutiliser les définitions du sommet *Top*, comme pour l'opération *SMGetNameBinding*. Il sera nécessaire de définir de nouvelles opérations relatives aux états *GetSMAdministrativeState*, *GetSMOperationalState* et *SMGetUsageState*. On peut également réutiliser l'opération *SMEventReport*.

Le schéma *SMGetAll* utilise également une opération définie sur la base de *TopState*. Elle inclut la définition de *TopGetAll* et renforce la condition postérieure.

SMGetAll

StateManagementState
TopGetAll

administrativeStateOid ⨀ *attributes*
  ∪ *administrativeStateValue administrativeState* ⨀ *result!*
OperationalStateOid ⨀ *attributes*
  ∪ *operationalStateValue operationalState* ⨀ *result!*
UsageStateOid ⨀ *attributes*
  ∪ *usageStateValue usageState* ⨀ *result!*

L'opération *SMReplaceAdministrativeState* décrit le comportement spécifique à la classe de gestion d'état par laquelle l'état administratif est remplacé par une autre valeur fournie comme entrée. Selon l'état de l'objet au moment de l'exécution de l'opération, l'état d'utilisation peut également être modifié. L'état opérationnel n'est pas modifié par cette opération.

SMReplaceAdministrativeState

StateManagementState
TopState
input?: AdministrativeState

administrativeState' ⨀
  IF *usageState* ∈ *idle*
  THEN *{ unlocked* ⇸ *unlocked,locked* ⇸ *locked,*
    shuttingDown ⇸ *locked,locked* ⇸ *shuttingDown,*
    shuttingDown ⇸ *shuttingDown}* ↾ *{ input?}*
  ELSE *{unlocked* ⇸ *unlocked, locked* ⇸ *locked,*
    shuttingDown ⇸ *locked}* ↾ *{ input?}*
administrativeState' = locked ∪ *usageState' = idle*
administrativeState' ∈ *locked* ∪ *usageState' = usageState*
operationalState' = operationalState

Le comportement spécifié dans la partie prédicative du schéma est une formalisation de la description informelle donnée dans la Rec. X.731 du CCITT | ISO/CEI 10164-2. Pour des raisons d'exhaustivité, il faut également définir les opérations de remplacement de l'état opérationnel et de l'état d'utilisation.

Enfin, il y a un certain nombre d'autres opérations qui décrivent le comportement spécifique de la classe de gestion d'état. Ces opérations ne sont pas énumérées ici mais figurent au B.7. Il s'agit notamment de *SMCapacityDecrease*, *SMCapacityIncrease*, *SMDisable*, *SMEnable*, *SMNewUser* et *SMUserQuit*.

B.4.3 Classes instanciables

Aucune des classes décrites ci-dessus ne peut être instanciée. La procédure utilisée peut être poursuivie. *StateManagement* peut être réutilisé pour définir une classe appelée *CIRCUIT*, qui à son tour peut être utilisée pour définir *ECIRCUIT* et donc la classe instanciable *ActualECircuit*.

Cela n'est pas indiqué dans le guide, étant donné que les procédures sont exactement les mêmes que celles qui ont été expliquées et qu'il est inutile de les répéter.

B.5 Questions en suspens

Le présent paragraphe énumère les principales questions rencontrées au cours de la traduction en langage Z des spécifications des objets gérés fondées sur les directives GDMO. Quand un aspect se rapporte au langage Z sans avoir de structure correspondante pour une caractéristique particulière de la spécification des objets gérés, le traitement informel proposé dans cette annexe est également inclus.

B.5.1 Définition du comportement dans les objets gérés

Dans les modèles, on utilise l'expression «définition de comportement» pour pratiquement toutes les entités, qu'elles soient des données ou des processus. Dans ce dernier cas, elle peut inclure des renseignements sur le comportement réel (au sens strict) ou des informations statiques sur l'entité, telles que son utilisation prévue, voire les deux. Au cours de la traduction, il convient d'analyser le texte figurant sous ce titre et d'en tirer les informations comportementales qui s'appliquent à l'entité en question. Ces informations comportementales seront utilisées dans la traduction formelle, alors que le texte proprement dit pourra être inclus comme un commentaire dans la spécification en langage Z.

B.5.2 Opérations internes en langage Z

Une opération interne dans un objet géré représente le cas où une notification est émise spontanément (sans invocation par la gestion). Les opérations internes sont également une caractéristique souhaitée de nombreux autres systèmes. Actuellement, en langage Z, cette caractéristique est représentée de manière informelle par un commentaire sous forme de texte en langage naturel, ce qui est une caractéristique importante de toute spécification en langage Z correctement rédigée.

B.5.3 Classes abstraites en langage Z

Il est parfois utile d'identifier des classes abstraites, à savoir des classes qui n'ont pas d'instanciation propre. Certaines classes d'objets gérés (telles que *top*) ne peuvent pas être instanciées. Il serait utile de pouvoir montrer quelles sont les parties des spécifications Z correspondantes qui représentent les classes pouvant être instanciées. Pour l'heure, cela se fait par annotation informelle.

B.5.4 Sémantique de PARAMETER

L'incorporation de la sémantique de PARAMETER dans les objets n'est pas abordée dans la présente Recommandation | Norme internationale.

B.6 Conversion de type de données ASN.1 en langage Z

Les questions de traduction seront décrites individuellement pour chaque constructeur ASN.1.

B.6.1 Types simples

L'ASN.1 comporte quelques types simples incorporés. Ceux-ci ont une structure normalisée, mais ils ne sont généralement pas intéressants dans le contexte des spécifications et pour cette raison on peut, dans la majorité des cas, les représenter comme des ensembles donnés. Il existe un large choix de types de chaînes de caractères:

[NUMERICSTRING, PRINTABLESTRING, TELETEXSTRING,
VIDEOTEXSTRING,VISIBLESTRING, IA5STRING,
GRAPHICSTRING, GENERALSTRING]

Deux d'entre eux ont des synonymes:

T61STRING == TELETEXSTRING
ISO64STRING == VISIBLESTRING

Parmi les autres types simples, on peut représenter les entiers par ℤ, les opérateurs booléens et le type nul par des types libres:

Boolean ::= btrue | bfalse
Null ::= null

On notera que ces types libres définissent aussi la notation de valeur pour ces types.

Real, BitString et OctetString peuvent généralement être considérés comme des ensembles donnés (bien qu'il soit parfois nécessaire de structurer les types BitString et OctetString).

[REAL,BITSTRING,OCTETSTRING]

La présente Recommandation | Norme internationale décrit également un autre type spécial qui sera fourni comme un ensemble donné.

[OBJECTID]

Dans ce cas, *OBJECTID* représente un identificateur d'objet ASN.1.

Les identificateurs d'objet sont en fait des séquences non vides de fractions ⑦, et il peut être utile de les modéliser en tant que telles plutôt que comme un ensemble donné. Dans ce cas, il convient de choisir judicieusement la notation de valeur appropriée.

Il existe également certains types «utiles» qui sont définis en rotation ASN.1 dans la norme ASN.1. Aussi, bien qu'ils puissent être définis en termes d'autres structures ASN.1, il est une fois de plus utile de les fournir comme des ensembles donnés.

[GENERALIZEDTIME, UTCTIME, OBJECTDESCRIPTOR, EXTERNAL]

Any

La notation ASN.1 permet un type spécial ANY qui peut contenir n'importe quel autre type ASN.1. Un tel type n'est pas permis dans le langage Z, et il serait difficile d'étendre celui-ci pour en contenir un. Toutefois, à partir d'un ensemble de types donné, il est possible de définir un type Z libre pouvant inclure n'importe lequel de ces autres types. Une autre tactique consiste à définir ANY comme un ensemble donné pour les besoins du contrôle de type. Cela peut se faire mais uniquement à cette fin. Généralement, le type *AttributeValues* remplace *ANY*, ce qui est défini ci-après.

B.6.2 Types structurés

Les constructeurs réalisent d'autres types en notation ASN.1.

Set (Ensemble)

Les ensembles «Set» en ASN.1 peuvent être représentés en langage Z comme des nuplets ou des schémas. Comme les nuplets Z ne permettent pas de donner un nom aux composantes, il est peut être plus judicieux d'utiliser des schémas. Toutefois, la notation de valeur Z pour les schémas est moins pratique. L'étiquetage n'est pas nécessaire – et n'est pas possible – en langage Z étant donné que les composantes de l'ensemble «set» peuvent toujours être discriminées soit par leur position dans un nuplet, soit par leur nom de composante dans un schéma.

Les composantes, dans ce type structuré et dans d'autres, peuvent être facultatives (*OPTIONAL*). Cela peut être représenté en langage Z par l'élargissement du type de la composante en option au moyen d'une valeur spéciale «absent». Les valeurs *DEFAULT* ne peuvent pas être aisément représentées comme des caractéristiques d'un type de données. On peut représenter le comportement sous-entendu par la valeur par défaut dans n'importe quelle opération sur cette donnée.

Sequence (Séquence)

Les séquences en ASN.1 peuvent être modélisées exactement comme des ensembles ASN.1 étant donné que la seule différence réside dans la présence d'un ordre explicite. Comme tel est le cas, on pourrait estimer qu'un nuplet est plus approprié, mais on peut également utiliser des schémas.

Set-of (Ensemble de)

Les «ensembles de» en ASN.1 sont en fait des sacs qui peuvent être définis comme tels en langage Z. Il convient de noter que le modèle MIM requiert explicitement que tous ces sacs soient traités comme des ensembles. Il est donc en fait plus utile de modéliser ce type sous la forme d'un ensemble Z.

Lorsqu'il est nécessaire de sous-typer un type ASN.1, il faut généralement ajouter une contrainte «prédicat» au type. Dans certains cas, le sous-type d'entier ou de schéma par exemple, on peut le faire dans la définition même. Dans d'autres (le sous-type de sac), la contrainte doit être appliquée à des variables définies comme étant de ce type.

Sequence-of (Séquence de)

Les types Sequence-of en ASN.1 peuvent être aisément modélisés comme des séquences Z.

Choice (Choix)

Les types *Choice* en ASN.1 sont des énumérations directes pouvant être modélisées par des types libres en langage Z.

Ce type présente un problème considérable au niveau de la visibilité. Au sein de l'ASN.1, les constructeurs dans un type *Choice* s'appliquent uniquement à ce type. Un nom de constructeur peut donc être utilisé dans plusieurs énumérations. En langage Z, ces noms sont globaux et ne peuvent donc pas être réutilisés. On résout généralement le problème en modifiant le nom des constructeurs, généralement par l'adjonction d'un préfixe à leur nom de type.

Un problème analogue survient quand on produit des types ASN.1 qui sont des synonymes d'un entier, par exemple, mais avec des valeurs nommées. Celles-ci sont locales pour le type du synonyme en ASN.1 mais sont des synonymes globaux pour les entiers en langage Z. Là aussi le problème doit être résolu par la modification du nom des constructeurs.

B.6.3 Types d'attribut simples

La solution la plus simple consiste à représenter l'attribut d'objet géré contenu dans cet objet géré comme une variable Z ayant le type de donnée approprié. Il faudra définir une constante qui représente l'identificateur *OBJECTID* de cet attribut. Lorsque l'on définit des opérations de correspondance pour cet attribut, on peut utiliser la valeur normalisée effective du paramètre Propriété de correspondance.

Soit donc l'état administratif de l'attribut d'objet géré. Nous avons défini un type:

AdministrativeState ::= unlocked | locked | shuttingDown

et on peut définir une constante pour représenter l'identificateur d'objet de l'attribut:

administrativeStateOid : OBJECTID

La valeur de cet identificateur peut être présentée comme une contrainte imposée à cette définition axiomatique.

Ensuite sera définie une variable d'état dans l'objet géré:

MOState

administrativeState: AdministrativeState

Cette solution est simple et pratique mais nécessite des listes de définitions axiomatiques *OBJECTID*. Elle rend également le lien entre le nom de l'attribut et son identificateur *OBJECTID* purement syntaxique. L'habitude d'ajouter le suffixe *Oid* a été prise.

B.6.4 Types d'attribut en tant que schémas

On peut également encapsuler les caractéristiques d'un attribut dans un seul type de schéma qui sera le type de la variable Z modélisant l'attribut d'objet géré:

AdministrativeStateType

value:AdministrativeState
Oid:OBJECTID

Oid = ↑4, 3, 19, 27, 1, 3 ↑

Il est important de fournir, à ce stade, une valeur pour *OBJECTID* car il faut sous-entendre qu'il ne peut être modifié, même si la valeur le peut.

Aucune structure *OBJECTID* n'a été définie, mais en écrivant:

*OBJECTID == **seq**$_1$* ⦶

on peut donner un sens au schéma précédent. Celui-ci pourrait également contenir le paramètre Propriété de correspondance si on estime qu'il est important de représenter cela dans la spécification.

L'objet géré contiendrait alors un attribut de ce type:

MOState

administrativeState : AdministrativeStateType

La référence à sa valeur ou à son identificateur d'objet peut être faite par la sélection d'une composante, comme dans *administrativeState.value*.

Cette technique permet de conférer aisément une sémantique à la connexion entre un attribut et son *OBJECTID*. Toutefois, il peut paraître étrange au lecteur de la spécification que l'identificateur d'objet soit présent dans l'état d'objet géré, même s'il ne peut être modifié (alors qu'en fait il est une constante globale, connue au moment de la spécification).

B.6.5 Type AttributeValues

Comme indiqué ci-dessus, il est difficile de modéliser en langage Z le type ANY de la notation ASN.1. Un cas fréquent est celui où l'on donne des listes de valeurs d'attribut. Il faudra en langage Z une définition de type libre combinant des types d'attribut déjà définis. Cette méthode fonctionne tant que l'ensemble d'attributs utilisé est fixe au moment de la spécification. En général, on obtiendra à peu près ce qui suit:

AttributeValues ::= administrativeStateValue ⓐAdministrativeState ❍ |
objectClassValue ⓐOBJECTID ❍ |
nameBindingValue ⓐOBJECTID ❍ |
packagesValue ⓐ ⦶ OBJECTID ❍ |
allomorphsValue ⓐ ⦶ OBJECTID ❍ |
operationalStateValue ⓐOperationalState ❍ |
usageStateValue ⓐUsageState ❍

B.7 Un exemple complet

Le présent paragraphe représente le modèle formel complet sur lequel est fondé l'exemple du B.4. Il est présenté dans le style Z habituel de la déclaration avant l'utilisation; autrement dit, les définitions des types convertis depuis l'ASN.1 figurent au début et les définitions de comportement à la fin.

Un point du style de notification mérite quelques observations. Les définitions *AttributeValues* et *OBJECTINSTANCE* sont mutuellement récursives. Techniquement, cela est interdit en langage Z, et en conséquence on a procédé de la manière suivante pour permettre la définition: *AttributeValues* a été introduit comme un ensemble donné, *OBJECTINSTANCE* a ensuite été défini au moyen de l'ensemble donné *AttributeValues*. Cette définition de l'instance *OBJECTINSTANCE* a ensuite été utilisée pour introduire les restrictions que l'ensemble *AttributeValues* a l'autorisation de prendre. L'obligation de prouver que de tels ensembles existent réellement a été remplie, mais elle n'est pas présentée ici.

B.7.1 Types de base ASN.1

[NUMERICSTRING, PRINTABLESTRING, TELETEXSTRING, VIDEOTEXSTRING]
[VISIBLESTRING, IA5STRING, GRAPHICSTRING, GENERALSTRING]

T61STRING == TELETEXSTRING

ISO64STRING == VISIBLESTRING

Boolean ::= btrue | bfalse

Null ::= null

[REAL,BITSTRING,OCTETSTRING]

[OBJECTID]

[ANY]

[GENERALIZEDTIME,UTCTIME,OBJECTDESCRIPTOR,EXTERNAL]

B.7.2 Attributs d'objet géré

L'ensemble donné suivant est une marque de réservation pour une définition de type libre plus complexe et plus complète, donnée progressivement à mesure que chaque nouvelle classe est définie.

[AttributeValues]

AttributeValuesOptional ::= present ◎ AttributeValues ❍ | absent

RelativeDistinguishedName == AttributeValues

*RDNSequence == **seq** RelativeDistinguishedName*

DistinguishedName == RDNSequence

OBJECTINSTANCE ::= distinguishedName ◎DistinguishedName ❍ | nonSpecificForm ◎◴❍
| localDistinguishedName ◎RDNSequence ❍

B.7.3 Notifications

ProbableCause == OBJECTID

SpecificIdentifier ::= globalvalue ◎OBJECTID ❍ | localValue ◎◴❍

SpecificProblems == ◴ SpecificIdentifier

SpecificProblemsOptional ::= sPPresent ◎ SpecificProblems ❍ | sPAbsent

PerceivedSeverity ::= *indeterminate | critical | major | minor | warning | cleared*

BackedUpStatus == Boolean

BackedUpStatusOptional ::= bUSPresent ◎ BackedUpStatus ❍ | bUSAbsent

ObjectInstanceOptional ::= oIPresent ◎ OBJECTINSTANCE ❍ | oIAbsent

TrendIndication ::= lessSevere | noChange | moreSevere

TrendIndicationOptional ::= tIPresent ◎ TrendIndication ❍ | tIAbsent

ObservedValue ::= int ◎◴❍ | real ◎ REAL ❍

ObservedValueOptional ::= oVPresent ◎ ObservedValue ❍ | oVAbsent

ThresholdLevelInd ::=
up ◎ ObservedValue ◈ObservedValueOptional ❍
| down ◎ ObservedValue ◈ ObservedValueOptional ❍

ThresholdLevelIndOptional ::= tLIPresent ◎ThresholdLevelInd ❍ | tLIAbsent

ArmTimeOptional ::= aTPresent ◎GENERALIZEDTIME ❍| aTAbsent

ThresholdInfo ==
OBJECTID ◈ ObservedValue ◈ ThresholdLevelIndOptional ◈ ArmTimeOptional

ThresholdInfoOptional ::= thIPresent ◎ThresholdInfo ❍ | thIAbsent

NotificationIdentifier == ◴

NotificationIdentifierOptional ::= nIPresent ◎ NotificationIdentifier ❍ | nIAbsent

CorrelatedNotifications == ◴*((◴ NotificationIdentifier) ◈ ObjectInstanceOptional)*

CorrelatedNotificationsOptional ::= cNPresent ◎ CorrelatedNotifications ❍ | cNAbsent

AttributeValueChangeDefinition == ◴*(OBJECTID ◈ AttributeValuesOptional ◈ AttributeValues)*

AttributeValueChangeDefinitionOptional ::=
aVCDPresent ◎ AttributeValueChangeDefinition ❍ | aVCDAbsent

MonitoredAttributes == ◴ *OBJECTID*

MonitoredAttributesOptional ::= mAPresent ◎ MonitoredAttributes ❍ | mAAbsent

ProposedRepairActions == ◴ *SpecificIdentifier*

ProposedRepairActionsOptional ::= pRAPresent ◎ ProposedRepairActions ❍ | pRAAbsent

AdditionalTextOptional ::= adTPresent ◎ GRAPHICSTRING ❍ | aDTAbsent

ManagementExtension == OBJECTID ◈ Boolean ◈ ANY

AdditionalInformation == ◴ *ManagementExtension*

AdditionalInformationOptional ::= aIPresent ◎AdditionalInformation ❍ | aIAbsent

SourceIndicator ::= resourceOperation | managementOperation | sIUnknown

SourceIndicatorOptional ::= sIPresent ◎ SourceIndicator ❍ | sIAbsent

AttributeIdentifierList == 🕘 OBJECTID

AttributeIdentifierListOptional ::= atIPresent ◎ AttributeIdentifierList ❍ | atIAbsent

Attribute == OBJECTID 🏷 AttributeValues

AttributeList == 🕘 Attribute

AttributeListOptional ::= aLPresent ◎ AttributeList ❍ | aLAbsent

AlarmInfo

probableCause: ProbableCause
specificProblems: SpecificProblemsOptional
perceivedSeverity: PerceivedSeverity
backedUpStatus: BackedUpStatusOptional
backUpObject: ObjectInstanceOptional
trendIndication: TrendIndicationOptional
thresholdInfo: ThresholdInfoOptional
notificationIdentifier: NotificationIdentifierOptional
correlatedNotifications: CorrelatedNotificationsOptional
stateChangeDefinition: AttributeValueChangeDefinitionOptional
monitoredAttributes: MonitoredAttributesOptional
proposedRepairActions: ProposedRepairActionsOptional
additionalText: AdditionalTextOptional
additionalInformation: AdditionalInformationOptional

AttributeValueChangeInfo

sourceIndicator: SourceIndicatorOptional
attributeIdentifierList: AttributeIdentifierListOptional
attributeValueChangeDefinition: AttributeValueChangeDefinitionOptional
notificationIdentifier: NotificationIdentifierOptional
correlatedNotifications: CorrelatedNotificationsOptional
additionalText: AdditionalTextOptional
additionalInformation: AdditionalInformationOptional

ObjectInfo

sourceIndicator: SourceIndicatorOptional
attributeList: AttributeListOptional
notificationIdentifier: NotificationIdentifierOptional
correlatedNotifications: CorrelatedNotificationsOptional
additionalText: AdditionalTextOptional
additionalInformation: AdditionalInformationOptional

RelationshipChangeInfo

sourceIndicator: SourceIndicatorOptional
attributeIdentifierList: AttributeIdentifierListOptional
relationshipChangeDefinition: AttributeValueChangeDefinitionOptional
notificationIdentifier: NotificationIdentifierOptional
correlatedNotifications: CorrelatedNotificationsOptional
additionalText: AdditionalTextOptional
additionalInformation: AdditionalInformationOptional

SecurityAlarmInfo

notificationIdentifier: NotificationIdentifierOptional
correlatedNotifications: CorrelatedNotificationsOptional
additionalText: AdditionalTextOptional
additionalInformation: AdditionalInformationOptional

StateChangeInfo

sourceIndicator: SourceIndicatorOptional
attributeIdentifierList: AttributeIdentifierListOptional
stateChangeDefinition: AttributeValueChangeDefinitionOptional
notificationIdentifier: NotificationIdentifierOptional
correlatedNotifications: CorrelatedNotificationsOptional
additionalText: AdditionalTextOptional
additionalInformation: AdditionalInformationOptional

EventInfo ::= attributeValueChange @AttributeValueChangeInfo ❍
| *communicationsAlarm @AlarmInfo ❍*
| *environmentalAlarm @AlarmInfo ❍*
| *equipmentAlarm @AlarmInfo ❍*
| *integrityViolation\ @ SecurityAlarmInfo ❍*
| *objectCreation @ObjectInfo ❍*
| *objectDeletion @ObjectInfo ❍*
| *operationalViolation @SecurityAlarmInfo ❍*
| *physicalViolation @SecurityAlarmInfo ❍*
| *processingError @AlarmInfo ❍*
| *qualityOfServiceAlarm @AlarmInfo ❍*
| *relationshipChange @RelationshipChangeInfo ❍*
| *securityServiceOrMechanismViolation @SecurityAlarmInfo ❍*
| *stateChange @StateChangeInfo ❍*
| *timeDomainViolation @SecurityAlarmInfo ❍*

B.7.4 "Rec. X.721 du CCITT (1992)" | ISO/CEI 10165-2:1992: Sommet

allomorphsOid: OBJECTID
nameBindingOid: OBJECTID
objectClassOid: OBJECTID
packagesOid: OBJECTID

allomorphsValue : (ℙ OBJECTID) ⨾ AttributeValues
nameBindingValue : OBJECTID ⨾ AttributeValues
objectClassValue : OBJECTID ⨾ AttributeValues
packagesValue : (ℙ OBJECTID) ⨾ AttributeValues

---

disjoint ↑ran allomorphsValue, **ran** nameBindingValue,

ran objectClassValue, **ran** packagesValue ⇧

packagesPackageOid: OBJECTID
allomorphsPackageOid: OBJECTID

TopState

allomorphs: ℙ OBJECTID
objectClass: OBJECTID
nameBinding: OBJECTID
packages: ℙ OBJECTID
attributes: ℙ OBJECTID

{ objectClassOid, nameBindingOid} ⇸ attributes
allomorphsPackageOid ⋵ packages ⨾ allomorphsOid ⋵ attributes
packagesPackageOid ∈ packages
packages ⤔ ⨾ packagesOid ⋵ attributes

TopGetNameBinding

ΞTopState
result!: OBJECTID

---

result! = nameBinding

TopGetAll

ΞTopState
result!: ℙ AttributeValues

 # attributes = # result!
ObjectClassValue objectClass ⋵ result!
NameBindingValue nameBinding ⋵ result!
PackagesOid ⋵ attributes ⨾ packagesValue packages ⋵ result!
AllomorphsOid ⋵ attributes ⨾ allomorphsValue allomorphs ⋵ result!

TopEventReport

ΞTopState
notification!: EventInfo

B.7.5 Classe de gestion d'état

administrativeStateOid: OBJECTID
operationalStateOid: OBJECTID
usageStateOid: OBJECTID

AdministrativeState ::= unlocked | locked | shuttingDown

OperationalState ::= enabled | disabled

UsageState ::= idle | active | busy

administrativeStateValue: AdministrativeState ⓒAttributeValues
operationalStateValue: OperationalState ⓒ AttributeValues
usageStateValue: UsageState ⓒ AttributeValues

disjoint *↑**ran** allomorphsValue, **ran** nameBindingValue, **ran** objectClassValue,*
__ran__ packagesValue, __ran__ administrativeStateValue,
__ran__ operationalStateValue, __ran__ usageStateValue ⇧

StateManagementState

TopState
administrativeState:
AdministrativeState
operationalState: OperationalState
usageState: UsageState

operationalState = disabled ∪ usageState = idle
administrativeState = locked ∪ usageState = idle
usageState = idle ∪ administrativeState ◴ shuttingDown

SMGetNameBinding

TopGetNameBinding
⌘StateManagementState

SMEventReport

TopEventReport
⌘StateManagementState

SMGetAll

⌘StateManagementState
TopGetAll

administrativeStateOid ⊽attributes
∪ administrativeStateValue administrativeState ⊽result!
OperationalStateOid ⊽attributes ∪ operationalStateValue operationalState ⊽result!
UsageStateOid ⊽attributes ∪ usageStateValue usageState ⊽result!

SMCapacityDecrease

StateManagementState
TopState

usageState = active ⇒ *usageState' ∈{active, busy}*
usageState ≠ active ⇒ *usageState' = usageState*
administrativeState' = administrativeState
operationalState' = operationalState

SMCapacityIncrease

StateManagementState
TopState

usageState = busy ⇒ *usageState' = active*
usageState ≠ busy ⇒ *usageState' = usageState*
administrativeState' = administrativeState
operationalState' = operationalState

SMDisable

StateManagementState
TopState

administrativeState' =
  IF *administrativeState = shuttingDown*
  THEN *locked*
  ELSE *administrativeState*
operationalState' = disabled
usageState' = idle

SMEnable

StateManagementState
TopState

operationalState = disabled
operationalState' = enabled
administrativeState' = administrativeState
usageState' = usageState

SMNewUser

StateManagementState
TopState

operationalState = enabled
administrativeState = unlocked
usageState ∈{ idle, active}
usageState' ∈{active, busy}
administrativeState' = administrativeState
operationalState' = operationalState

SMUserQuit

StateManagementState
TopState

administrativeState = shuttingDown ⇒ usageState' = idle
∨ administrativeState' = locked
administrativeState ≠ shuttingDown ∧ usageState' ≠ idle
∨ administrativeState' = administrativeState
usageState ∈ { active, busy}
usageState' ∈ {idle, active}
operationalState' = operationalState

SMReplaceAdministrativeState

StateManagementState
TopState

administrativeState' ∈
IF *usageState ≠ idle*
THEN *{ unlocked ∈ unlocked,locked ∈ locked,*
shuttingDown ∈ locked,locked ∈ shuttingDown,
shuttingDown ∈ shuttingDown} ⇒ { input?} ⇒
ELSE *{ unlocked ∈ unlocked, locked ∈ locked,*
shuttingDown ∈ locked} ⇒ { input?} ⇒
administrativeState' = locked ∨ usageState' = idle
administrativeState' ≠ locked ∨ usageState' = usageState
operationalState' = operationalState

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	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, de télégraphie, de télécopie, circuits téléphoniques 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 pour données et communication entre systèmes ouverts**
Série Y	Infrastructure mondiale de l'information
Série Z	Langages de programmation