UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

UIT-T X.880

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

(07/94)

RÉSEAUX DE COMMUNICATION DE DONNÉES ET COMMUNICATION ENTRE SYSTÈMES OUVERTS

APPLICATIONS OSI – OPÉRATIONS DISTANTES

TECHNOLOGIE DE L'INFORMATION – OPÉRATIONS DISTANTES: CONCEPTS, MODÈLE ET NOTATION

Recommandation UIT-T X.880

(Antérieurement «Recommandation du CCITT»)

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. Au sein de l'UIT-T, qui est l'entité qui établit les normes mondiales (Recommandations) sur les télécommunications, participent quelque 179 pays membres, 84 exploitations de télécommunications reconnues, 145 organisations scientifiques et industrielles et 38 organisations internationales.

L'approbation des Recommandations par les Membres de l'UIT-T s'effectue selon la procédure définie dans la Résolution n° 1 de la Conférence mondiale de normalisation des télécommunications (CMNT), (Helsinki, 1993). De plus, la CMNT, qui se réunit tous les quatre ans, approuve les Recommandations qui lui sont soumises et établit le programme d'études pour la période suivante.

Dans certains secteurs de la technologie 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. Le texte de la Recommandation X.880 de l'UIT-T a été approuvé le 1er juillet 1994. Son texte est publié, sous forme identique, comme Norme internationale ISO/CEI 13712-1.

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.

RECOMMANDATIONS UIT-T DE LA SÉRIE X

RÉSEAUX POUR DONNÉES ET INTERCONNEXION DE SYSTÈMES OUVERTS

(Février 1994)

ORGANISATION DES RECOMMANDATIONS DE LA SÉRIE X

Domaine	Recommandations
RÉSEAUX PUBLICS POUR DONNÉES	
Services et services complémentaires	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
Définition 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
Test de conformité	X.290-X.299
INTERFONCTIONNEMENT DES RÉSEAUX	
Considérations générales	X.300-X.349
Système mobiles de transmission de données	X.350-X.369
Gestion	X.370-X.399
SYSTÈMES DE MESSAGERIE	X.400-X.499
ANNUAIRE	X.500-X.599
RÉSEAUTAGE OSI ET ASPECTS DES SYSTÈMES	
Réseautage	X.600-X.649
Dénomination, adressage et enregistrement	X.650-X.679
Notation de syntaxe abstraite numéro un (ASN.1)	X.680-X.699
GESTION OSI	X.700-X.799
SÉCURITÉ	X.800-X.849
APPLICATIONS OSI	
Engagement, concomitance et rétablissement	X.850-X.859
Traitement des transactions	X.860-X.879
Opérations distantes	X.880-X.899
TRAITEMENT OUVERT RÉPARTI	X.900-X.999

TABLE DES MATIÈRES

Résumé

La présente Recommandation | Norme internationale utilise la notation de syntaxe abstraite numéro un (ASN.1) pour définir les classes d'objets informationnels correspondant aux concepts fondamentaux du service d'opérations distantes (ROS). Celui-ci fournit à son tour la notation qui permettra aux concepteurs d'applications de spécifier des instances de ces classes. La présente Recommandation | Norme internationale fournit également une collection de définitions qui permettent de spécifier le protocole générique d'échange entre objets communiquant par opérations distantes. Elle renferme aussi un certain nombre de définitions d'intérêt général pour les concepteurs des applications à base d'opérations distantes.

Introduction

Le concept d'opérations distantes (ROS) est un paradigme de la communication interactive entre objets. En tant que tel, il peut être utilisé pour la conception et la spécification des applications réparties. L'interaction de base mise en jeu est l'invocation d'une opération par un objet (l'invocateur), son exécution par un autre (l'exécutant), éventuellement suivie par un rapport sur le résultat de l'opération retourné à l'invocateur.

Les concepts d'opérations distantes (ROS) sont abstraits, et peuvent être réalisés de multiples manières. Ainsi, les objets dont les interactions mettent en jeu les concepts d'opérations distantes peuvent être séparés par une interface logicielle ou par un réseau OSI.

La présente Recommandation | Norme internationale décrit les concepts et le modèle de service ROS. Elle utilise l'ASN.1 pour spécifier les classes d'objets informationnels correspondant aux concepts fondamentaux du service ROS, tels que les classes opération et erreur. Ces éléments fournissent à leur tour la notation qui permet aux concepteurs de spécifier des instances particulières de ces classes, comme des opérations ou des erreurs spécifiques.

La présente Recommandation | Norme internationale fournit un ensemble générique d'unités de données de protocole (PDU) qui peuvent être utilisées pour réaliser les concepts de service ROS entre objets distants les uns des autres. Ces PDU sont utilisées dans les applications OSI des concepts de service ROS, qui sont spécifiées dans les Recommandations | Normes internationales jumelles de celle-ci.

La présente Recommandation | Norme internationale fournit également un certain nombre de définitions d'ordre général pour les concepteurs d'applications à base d'opérations distantes ROS.

L'Annexe A fait partie intégrante de la présente Recommandation | Norme internationale.

Les Annexes B, C et D ne font pas partie intégrante de la présente Recommandation | Norme internationale.

NORME INTERNATIONALE

RECOMMANDATION UIT-T

TECHNOLOGIE DE L'INFORMATION – OPÉRATIONS DISTANTES: CONCEPTS, MODÈLE ET NOTATION

1 Domaine d'application

La présente Recommandation | Norme internationale spécifie le service d'opérations distantes (ROS) et utilise la notation de syntaxe abstraite numéro un (ASN.1) pour définir les classes d'objets informationnels correspondant aux concepts fondamentaux du service ROS. Ces éléments fournissent à leur tour la notation qui permettra aux concepteurs d'applications de spécifier des instances particulières de ces classes.

La présente Recommandation | Norme internationale renferme également une collection de définitions qui spécifient le protocole générique d'échange entre objets communiquant selon les concepts du service ROS. Les Recommandations | Normes internationales associées à celle-ci utilisent ces définitions pour définir les unités de données de protocole, les primitives de service et les définitions de contexte applicatif qui interviennent dans la réalisation OSI du service ROS.

Un certain nombre de définitions d'intérêt général pour les concepteurs des applications à base d'opérations distantes sont également données.

Aucune spécification n'est imposée quant à la conformité à la présente Recommandation | Norme internationale.

2 Références normatives

Les Recommandations de l'UIT-T et les Normes internationales suivantes contiennent des dispositions qui, par suite de la référence qui y est faite, constituent des dispositions valables pour la présente Spécification. Au moment de la publication, les éditions indiquées étaient en vigueur. Toutes Recommandations et Normes internationale sont sujettes à révision et les parties prenantes aux accords fondés sur la présente Spécification sont invitées à rechercher la possibilité d'appliquer les éditions les plus récentes des Recommandations et Normes internationales indiquées ci-après. Les membres de la CEI et de l'ISO possèdent le registre des Normes internationales en vigueur. Le Bureau de la normalisation des télécommunications de l'UIT tient à jour une liste des Recommandations UIT-T en vigueur.

2.1 Recommandations et Normes internationales identiques

- Recommandation UIT-T X.680 (1994) | ISO/CEI 8824-1:1994, *Technologie de l'information – Notation de syntaxe abstraite numéro un: Spécification de la notation de base*.
- Recommandation UIT-T X.681 (1994) | ISO/CEI 8824-2:1994, *Technologie de l'information – Notation de syntaxe abstraite numéro un: Spécification des objets informationnels.*
- Recommandation UIT-T X.682 (1994) | ISO/CEI 8824-3:1994, *Technologie de l'information – Notation de syntaxe abstraite numéro un: Spécification des contraintes*.
- Recommandation UIT-T X.683 (1994) | ISO/CEI 8824-4:1994, *Technologie de l'information – Notation de syntaxe abstraite numéro un: Paramétrage des spécifications ASN.1*.
- Recommandation UIT-T X.200 (1994) | ISO/CEI 7498-1:1994, *Technologie de l'information – Interconnexion de systèmes ouverts – Modèle de référence de base. Le modèle de base*.
- Recommandation UIT-T X.881 (1994) | ISO/CEI 13712-2:1994, *Technologie de l'information – Opérations distantes: Applications OSI – Définition du service de l'élément de service d'opérations distantes*.
- Recommandation UIT-T X.882 (1994) | ISO/CEI 13712-3:1994, *Technologie de l'information – Opérations distantes: Applications OSI – Spécification du protocole de l'élément de service d'opérations distantes*.

2.2 Paires de Recommandations | Normes internationales équivalentes par leur contenu technique

– Recommandation X.219 du CCITT (1988), *Opérations distantes: modèle, notation et définition du service.*

ISO/CEI 9072-1:1989, *Systèmes de traitement de l'information – Communication de texte – Opérations à distance – Partie 1: Modèle, notation et définition du service*.

– Recommandation X.229 du CCITT (1988), *Opérations distantes: Spécification du protocole*.

ISO/CEI 9072-2:1989, *Systèmes de traitement de l'information – Communication de texte – Opérations à distance – Partie 2: Spécification du protocole*.

2.3 Autres références

– Recommandation X.407 du CCITT (1988), *Systèmes de messagerie: Conventions pour la définition des services abstraits.*

3 Définitions

3.1 Définitions relatives au modèle de référence OSI

La présente Recommandation | Norme internationale utilise les termes suivants, définis dans la Rec. UIT-T X.200 | ISO/CEI 7498-1:

a) syntaxe abstraite;

b) unité de données de protocole;

c) qualité de service.

3.2 Définitions relatives à la notation ASN.1

La présente Recommandation | Norme internationale utilise les termes suivants, définis dans la Rec. UIT-T X.680 | ISO/CEI 8824-1:

a) type (de données);

b) valeur (de données).

La présente Recommandation | Norme internationale utilise les termes suivants, définis dans la Rec. UIT-T X.681 | ISO/CEI 8824-2:

a) champ;

b) objet (informationnel);

c) classe d'objets (informationnels);

d) ensemble d'objets (informationnels).

La présente Recommandation | Norme internationale utilise le terme suivant, défini

dans la Rec. UIT-T X.682 | ISO/CEI 8824-3:

a) contrainte;

b) valeur d'exception.

La présente Recommandation | Norme internationale utilise les termes suivants, définis dans la Rec. UIT-T X.683 | ISO/CEI 8824-4:

– paramétré.

3.3 Définitions relatives au service ROS

La présente Recommandation | Norme internationale définit les termes suivants:

3.3.1 argument: Valeur de données accompagnant l'invocation d'une opération.

3.3.2 association: Relation entre un couple d'objets, servant de contexte à l'invocation et à l'exécution d'une opération.

3.3.3 contrat d'association: Spécification des rôles d'un couple d'objets communicants pouvant être associés l'un à l'autre.

3.3.4 asymétrique: Qualificatif d'un lot d'opérations (ou d'un contrat d'association), pour lequel les ensembles d'opérations pouvant être exécutées par chacune des deux parties diffèrent l'un de l'autre.

3.3.5 lot de connexion: Spécification des rôles d'un couple d'objets communicants dans l'établissement ou la libération dynamique de leur association.

3.3.6 contrat: Ensemble de spécifications imposées à un ou plusieurs objets prescrivant un comportement collectif.

3.3.7 erreur: Rapport notifiant l'échec d'exécution d'une opération.

3.3.8 opération liée: Opération invoquée, pendant l'exécution d'une autre opération, par le (précédent) exécutant et qui doit être exécutée par le (précédent) invocateur.

3.3.9 objet: Modèle de système (ou éventuellement de sous-système autonome), caractérisé par son état initial et son comportement découlant d'interactions externes à travers des interfaces bien définies.

3.3.10 opération: Fonction qu'un objet (l'invocateur) peut demander à un autre (l'exécutant) d'exécuter.

3.3.11 lot d'opérations: Collection d'opérations liées utilisée pour spécifier les rôles pour un couple d'objets communicants, chaque opération pouvant être invoquée par un des deux objets ou par les deux pour être exécutée par l'autre.

3.3.12 paramètre (d'une erreur): Valeur de données pouvant accompagner le rapport d'erreur.

3.3.13 résultat: Valeur de données pouvant accompagner le rapport d'exécution avec succès d'une opération.

3.3.14 objet ROS: Objet dont les interactions avec d'autres objets sont décrites à l'aide des concepts d'opérations distantes ROS.

3.3.15 symétrique: Qualificatif d'un lot d'opérations (ou d'un contrat d'association) dans lequel les deux parties sont capables d'exécuter le même ensemble d'opérations.

3.3.16 synchrone: Qualificatif d'une opération qui, une fois invoquée, interdit à son invocateur d'invoquer une autre opération synchrone (avec le même exécutant désigné) tant que son résultat n'a pas été notifié.

4 Abréviations

Pour les besoins de la présente Recommandation | Norme internationale, les abréviations suivantes sont utilisées:

ASN.1	Notation de syntaxe abstraite numéro un (*abstract syntax notation one*)
PDU	Unité de données de protocole (*protocol data unit*)
QOS	Qualité de service (*quality of service*)
RO (ou ROS)	Opérations distantes (*remote operations*)

5 Conventions

La présente Recommandation | Norme internationale utilise l'ASN.1 pour définir:

a) les classes d'objets informationnels correspondant aux concepts ROS; elle indique également la notation avec laquelle les concepteurs d'applications ROS peuvent spécifier des instances particulières de ces classes;

b) les objets informationnels particuliers de ces classes;

c) les PDU du protocole générique d'opérations distantes (protocole ROS);

d) les types de données nécessaires à ces définitions.

Beaucoup de ces définitions sont paramétrées; pour les compléter, les utilisateurs doivent en préciser les paramètres effectifs.

6 Modèle ROS

Le concept d'opérations distantes (ROS) est un paradigme de la communication interactive entre objets. Les objets dont les interactions sont décrites et spécifiées à l'aide de concepts ROS sont des **objets ROS**. L'interaction de base mise en jeu est l'invocation d'une opération par un objet ROS (l'invocateur) et son exécution par un autre (l'exécutant).

L'achèvement de l'opération (sur un succès ou un échec) peut entraîner le renvoi par l'exécutant à l'invocateur d'un rapport sur le résultat de l'opération. Ceci est illustré à la Figure 1.

Un rapport notifiant l'achèvement avec succès d'une opération est un **résultat**; un rapport notifiant l'achèvement d'une opération sur un échec est une **erreur**.

Pendant l'exécution d'une opération, l'exécutant peut invoquer des **opérations liées**, à exécuter par l'invocateur de l'opération d'origine.

Pour un interfonctionnement correct, certaines des propriétés de l'opération doivent être connues à la fois de l'invocateur et de l'exécutant, notamment:

- si des rapports doivent être envoyés en retour, et dans l'affirmative, lesquels;
- les types des valeurs accompagnant le cas échéant les invocations d'opérations et les notifications envoyées en retour;
- les opérations pouvant le cas échéant être liées à l'opération d'origine;
- la valeur de code à utiliser pour distinguer l'opération en question des autres opérations pouvant être invoquées.

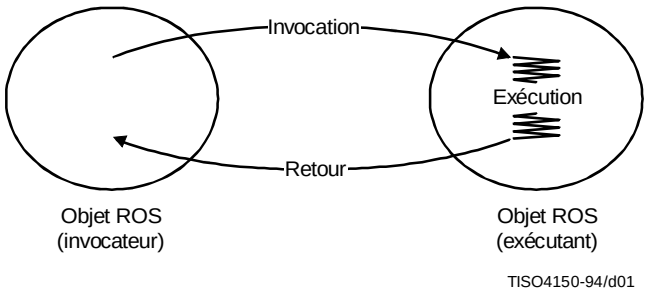

Figure 1 – Invocation, exécution et retour d'une opération

Les capacités d'interfonctionnement des (couples d')objets ROS d'une quelconque **classe d'objets ROS** sont définies en termes d'ensembles d'opérations liées appelés **lots d'opérations**. Un lot peut être **symétrique**, auquel cas il est défini par un seul ensemble d'opérations que chaque objet ROS du couple peut invoquer (pour être exécutées par l'autre). Ou alors, le lot peut être **asymétrique**, auquel cas il est défini par deux ensembles d'opérations, chacun pouvant être invoqué par un seul des deux objets du couple. Pour les besoins de la définition d'un lot asymétrique, les objets ROS sont arbitrairement étiquetés l'un comme **client** et l'autre comme **serveur**.

NOTE 1 – Alors que ces étiquettes sont en général arbitraires, il arrivera souvent que leur affectation soit intuitive, l'un des objets offrant manifestement un service que l'autre consomme.

Un couple d'objets ROS doivent être liés par une association servant de contexte à l'invocation et à l'exécution d'opérations. Chaque association de cette sorte est gouvernée par un **contrat d'association**. Un contrat est spécifié en termes de lots qui déterminent (collectivement) les opérations qui peuvent être invoquées dans le cadre de l'association. Si les spécifications de contrat comprennent un ou plusieurs lots asymétriques, le contrat est lui-même asymétrique. Pour les besoins de la spécification d'un contrat d'association asymétrique, les deux objets ROS qui établissent l'association entre eux sont étiquetés l'un comme **initiateur** et lautre comme **répondeur**.

Une association peut être créée ou dissoute par des moyens «hors ligne». Mais elle peut être aussi établie ou libérée dynamiquement. Une des options décrites dans la présente Recommandation | Norme internationale et permettant d'établir et de libérer dynamiquement une association est réalisée par l'invocation et l'exécution des opérations spéciales respectivement de **rattachement** et de **détachement**. Le contrat de cette dernière catégorie d'associations inclut un **lot de connexion** comprenant les opérations particulières de rattachement et de détachement à utiliser.

NOTE 2 – Le mécanisme d'établissement et de libération d'associations peut également être assuré par d'autres moyens décrits dans d'autres Recommandations | Normes internationales.

Une association nécessite qu'existe entre les deux objets une relation qui corresponde à l'acceptation par ces objets des termes d'un quelconque contrat d'association.

NOTE 3 – Cette spécification ne traite pas des moyens par lesquels de telles relations sont établies ou terminées.

Dans ce qui suit, les seuls objets qu'on voit impliqués dans une opération sont l'invocateur et l'exécutant. Toutefois, l'invocateur et l'exécutant d'une opération ne sont généralement pas directement rattachés l'un à l'autre, mais connectés par un intermédiaire quelconque à travers lequel sont transmis les invocations et les rapports en retour. La Figure 2 illustre ce schéma élargi.

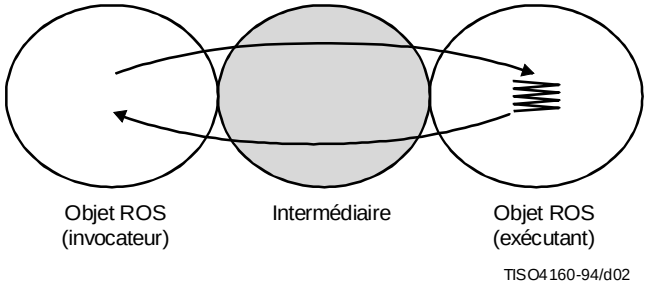

Figure 2 – Schéma élargi

L'intermédiaire peut introduire un retard et une possibilité d'échec ou d'inexactitude dans la transmission tant des invocations que des rapports, ainsi que dans l'établissement, la libération et la maintenance des associations. Il peut également introduire une possibilité de menace pour la sécurité de l'association et de ses opérations. L'importance de ces éléments (ainsi que d'autres facteurs) sont décrits dans le cadre de la qualité de service (QOS).

Les contrats d'association peuvent dans ce cas être vus comme tripartites, la partie tierce étant l'intermédiaire. Les obligations de l'intermédiaire au titre du contrat sont de satisfaire aux spécifications de qualité de service.

NOTE 4 – Ultérieurement, les spécifications objectifs et les spécifications minimales en matière de qualité de service pourront faire partie de la spécification des opérations, des lots d'opérations et du contrat d'association lui-même. Aux spécifications de chacun de ces niveaux correspondent différents aspects de qualité de service.

7 Réalisation des services ROS

Une **réalisation** de service ROS implique la définition d'un intermédiaire approprié permettant de véhiculer les invocations et les rapports entre objets ROS. Un tel intermédiaire peut par exemple comprendre:

a) une capacité de passation de message ou d'appel de procédure permettant de programmer séparément l'invocateur et l'exécutant d'une opération dans des modules logiciels distincts dans un même ordinateur;

b) une capacité de communication, permettant de programmer l'invocateur et l'exécutant d'une opération dans des ordinateurs distincts.

Une application peut être polyvalente, et peut alors être utilisée pour prendre en charge un contrat d'association quelconque. D'autres applications peuvent être spécifiques et n'accepter que des contrats d'un type particulier.

La Figure 3 illustre une façon de réaliser un service ROS avec un système de communication en intermédiaire, schéma qui sera vraisemblablement largement utilisé.

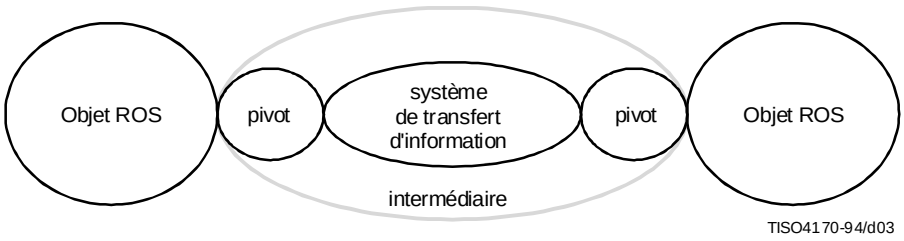

Figure 3 – Service ROS avec un système de communication en intermédiaire

Dans cette approche, l'intermédiaire est constitué d'objets **pivots** (*stub*), un pour chaque objet ROS, plus un objet de transfert d'information. L'objet pivot associé à chaque objet ROS apparaît comme jouant le rôle de l'objet ROS partenaire. En fait, il n'invoque ni n'exécute aucune opération, et se contente de transformer les invocations et les rapports en PDU et vice versa. Ces PDU sont échangées entre les pivots au moyen de l'objet de transfert d'information.

L'invocateur adresse donc son invocation au pivot qui lui est associé, lequel forme une PDU décrivant l'invocation. Le pivot utilise ensuite la capacité de transfert d'information pour transférer la PDU à l'autre pivot. Ce dernier interprète la PDU puis invoque l'opération appropriée de l'objet ROS qui lui est associé, c'est-à-dire l'exécutant. Une fois l'opération exécutée, l'exécutant remet s'il y a lieu un rapport au pivot qui lui est associé. Ce dernier forme la PDU décrivant le rapport, puis utilise la capacité de transfert d'information pour transférer la PDU à l'autre pivot, qui l'interprète et notifie le résultat à l'invocateur.

L'article 9 définit une collection de PDU appropriées.

Différentes capacités de transfert d'information peuvent être utilisées pour réaliser un système ROS de ce type. Parmi celles-ci, les capacités de transfert d'information de l'architecture OSI revêtent une importance particulière. Les deux Recommandations | Normes internationales jumelles UIT-T X.881 | ISO/CEI 13712-2 et UIT-T X.882 | ISO/CEI 13712-3 décrivent de telles réalisations.

8 Concepts ROS

8.1 Introduction

8.1.1 Cet article définit les classes d'objets informationnels suivantes, qui correspondent aux concepts de base d'opérations distantes, et spécifie les caractéristiques des objets de ces classes:

- **OPERATION** (décrit les opérations)
- **ERROR** (décrit les erreurs)
- **OPERATION-PACKAGE** (décrit les lots d'opérations)
- **CONNECTION-PACKAGE** (décrit les lots de connexion)
- **CONTRACT** (décrit les contrats d'association)
- **ROS-OBJECT-CLASS** (décrit les classes d'objets ROS)

8.1.2 Les classes d'objets informationnels sont définies en ASN.1. Ces définitions servent ensuite aux concepteurs d'applications de service ROS pour spécifier des instances particulières de ces classes. Les concepteurs sont encouragés à adopter cette approche pour leurs spécifications, mais pas obligés de le faire. Si une autre approche est suivie, la spécification résultante devra comprendre une description sur la manière d'en dériver une notation valide, ou faire référence à une telle explication.

NOTE – Un certain nombre de spécifications existantes utilisent la notation en macro-instructions ASN.1 (définie dans des versions antérieures de la présente Recommandation | Norme internationale: voir la Rec. X.219 du CCITT | ISO/CEI 9072-1) pour spécifier les opérations, les erreurs et les autres classes d'objets informationnels se rapportant au service ROS. L'Annexe C décrit la manière de transformer ces macro-instructions dans la notation indiquée. Ces macro-instructions ne devront plus être utilisées pour les nouvelles applications.

8.2 Opération

8.2.1 Une opération est une fonction qu'un objet (l'invocateur) peut demander à un autre objet (l'exécutant) d'exécuter. La classe d'objets informationnels **OPERATION**, à laquelle toutes les opérations appartiennent, est spécifiée comme suit, les différents champs étant décrits dans 8.2.2 à 8.2.13:

```
OPERATION ::= CLASS
{
        &ArgumentType           OPTIONAL,
        &argumentTypeOptional   BOOLEAN OPTIONAL,
        &returnResult           BOOLEAN DEFAULT TRUE,
        &ResultType             OPTIONAL,
        &resultTypeOptional     BOOLEAN OPTIONAL,
        &Errors                 ERROR OPTIONAL,
        &Linked                 OPERATION OPTIONAL,
        &synchronous            BOOLEAN DEFAULT FALSE,
        &alwaysReturns          BOOLEAN DEFAULT TRUE,
        &InvokePriority         Priority OPTIONAL,
        &ResultPriority         Priority OPTIONAL,
        &operationCode          Code UNIQUE OPTIONAL
}
WITH SYNTAX
{
        [ARGUMENT               &ArgumentType   [OPTIONAL   &argumentTypeOptional]]
        [RESULT                 &ResultType     [OPTIONAL   &resultTypeOptional]]
        [RETURN RESULT          &returnResult]
        [ERRORS                 &Errors]
        [LINKED                 &Linked]
        [SYNCHRONOUS            &synchronous]
        [ALWAYS RESPONDS        &alwaysReturns]
        [INVOKE PRIORITY        &InvokePriority]
        [RESULT-PRIORITY        &ResultPriority]
        [CODE                   &operationCode]
}
```

8.2.2 Le champ **&ArgumentType** (*type d'argument*) spécifie le type de données de l'argument de l'opération. Si, pour une opération donnée, ce champ est omis, l'opération ne prend pas de valeur d'argument.

8.2.3 Le champ **&argumentTypeOptional** (*type d'argument optionnel*), qui ne peut exister que lorsque le champ **&ArgumentType** est lui-même présent, spécifie si le type de données de l'argument de l'opération peut optionnellement être omis. Si ce champ est absent ou s'il prend la valeur **FALSE** (*faux*), la valeur de **&ArgumentType** ne peut être omise de la PDU **Invoke{}** (voir 9.3).

8.2.4 Le champ **&returnResult** (*retourner le résultat*) spécifie si un résultat doit être retourné en cas d'achèvement avec succès de l'opération, et prend la valeur **TRUE** (*vrai*) si c'est le cas, **FALSE** (*faux*) sinon.

8.2.5 Le champ **&ResultType** (*type de résultat*) spécifie le type de données de la valeur retournée avec le résultat de l'opération. Si cette indication est omise, l'opération ne retourne pas de valeur de résultat. Ce champ est omis si le champ **&returnResult** a la valeur **FALSE** (*faux*).

8.2.6 Le champ **&resultTypeOptional** (*type de résultat optionnel*), qui ne peut exister que lorsque le champ **&ResultType** est lui-même présent, spécifie si le type de données de la valeur retournée comme résultat de la bonne exécution de l'opération peut optionnellement être omis. Si ce champ est absent ou s'il prend la valeur **FALSE** (*faux*), la valeur de **&ResultType** ne peut être omise de la PDU **ReturnResult{}** (voir 9.4).

8.2.7 Le champ **&Errors** (*erreurs*) spécifie un ensemble d'erreurs, l'une quelconque de ces valeurs pouvant être retournée pour notifier l'échec d'exécution de l'opération. Si ce champ est omis, alors soit que l'échec d'exécution de l'opération est impossible, soit qu'il n'est pas notifié.

8.2.8 Le champ **&alwaysReturns** (*toujours retourner*) spécifie s'il faut toujours retourner le résultat de l'opération, et prend la valeur **TRUE** (*vrai*) si c'est le cas, **FALSE** (*faux*) sinon. Si ce champ a la valeur **TRUE** (*vrai*), au moins l'un des deux champs **&returnResult** ou **&Errors** doit être présent.

8.2.9 Le champ **&Linked** (*lié*) spécifie, lorsqu'il est présent, un ensemble d'opérations, l'une quelconque de ces opérations pouvant être invoquée en tant qu'opération liée pendant l'exécution de l'opération d'origine. Si ce champ est omis, aucune opération ne peut être liée à l'invocation de l'opération en cours.

8.2.10 Le champ **&synchronous** (*synchrone*) spécifie si l'opération est synchrone ou non, et prend la valeur TRUE (*vrai*) si c'est le cas, **FALSE** (*faux*) sinon. Si le champ **&returnResult** a la valeur **FALSE** (*faux*), ce champ aura toujours la valeur **FALSE** (*faux*). Si le champ **&synchronous** a la valeur **TRUE** (*vrai*), ceci implique qu'aucune autre opération *synchrone* ne pourra être invoquée par cette source jusqu'à la réception du rapport d'exécution en retour.

NOTE – La combinaison des champs **&alwaysReturns** et **&synchronous** remplace le concept antérieur de «classe d'opération» défini dans la Rec. X.219 du CCITT | ISO/CEI 9072-1.

8.2.11 Le champ **&InvokePriority** (*priorité de l'invocation*) spécifie les niveaux de priorité **Priority** (voir 8.9) avec lesquels il est permis d'invoquer l'opération.

8.2.12 Le champ **&ResultPriority** (*priorité de résultat*) spécifie les niveaux de priorité **Priority** (voir 8.9) avec lesquels il est permis de retourner le résultat de l'opération. Si le champ **&returnResult** a la valeur **FALSE** (*faux*), ce champ sera omis.

8.2.13 Le champ **&operationCode** (*code d'opération*) spécifie lorsqu'il est présent la valeur **Code** (voir 8.8) utilisée pour identifier l'opération, par exemple pour l'invoquer.

NOTE – Une opération qui n'a pas un **&operationCode** ne peut être invoquée par une PDU **&Invoke{}** (voir 9.3). De fait, toutes les opérations doivent se voir affecter un code d'opération sauf lorsqu'il est prévu de les utiliser dans des circonstances particulières (les opérations de rattachement par exemple).

8.3 Erreur

8.3.1 Une erreur est un rapport notifiant l'échec d'exécution d'une opération. La classe d'objets informationnels **ERROR**, à laquelle toutes les erreurs appartiennent, est spécifiée comme suit, les différents champs étant décrits dans 8.3.2 à 8.3.5:

```
ERROR ::= CLASS
{
        &ParameterType          OPTIONAL,
        &parameterTypeOptional  BOOLEAN OPTIONAL,
        &ErrorPriority          Priority OPTIONAL,
        &errorCode              Code UNIQUE OPTIONAL
}
WITH SYNTAX
{
        [PARAMETER              &ParameterType  [OPTIONAL  &parameterTypeOptional]]
        [PRIORITY               &ErrorPriority]
        [CODE                   &errorCode]
}
```

8.3.2 Le champ **&ParameterType** (*type de paramètre*) spécifie le type de données du paramètre de l'erreur. Si, pour une erreur donnée, ce champ est omis, l'erreur ne prend alors pas de valeur de paramètre.

8.3.3 Le champ **&parameterTypeOptional** (*type de paramètre optionnel*), qui ne peut exister que lorsque le champ **&ParameterType** est lui-même présent, spécifie si le type de données de la valeur retournée en tant que paramètre qualifiant l'erreur peut optionnellement être présent. Si ce champ est absent ou s'il prend la valeur **FALSE** (*faux*), la valeur de **&ParameterType** ne peut être omise de la PDU **ReturnError{}** (voir 9.5).

8.3.4 Le champ **&ErrorPriority** (*priorité d'erreur*) spécifie les niveaux de priorité **Priority** (voir 8.9) avec lesquels il est permis de retourner le résultat d'échec de l'opération.

8.3.5 Le champ **&errorCode** (*code d'erreur*) spécifie la valeur Code (voir 8.8) utilisée pour identifier l'erreur, par exemple pour la retourner.

NOTE – Une erreur qui n'a pas un **&errorCode** ne peut être retournée par la PDU **&ReturnError{}** définie ci-dessous. De fait, toutes les erreurs doivent se voir affecter un code d'erreur sauf lorsque leur utilisation se fait dans des circonstances particulières, comme pour les erreurs de rattachement.

8.4 Lot d'opérations

8.4.1 Un lot d'opérations est la spécification des rôles tenus par un couple d'objets communicants, en termes d'opérations que chacun des deux objets (l'invocateur) peut demander à l'autre (l'exécutant) d'exécuter. Lorsque le lot d'opérations est asymétrique, les termes «client» et «serveur» sont utilisés pour désigner les deux objets impliqués. La classe d'objets informationnels **OPERATION-PACKAGE**, à laquelle tous les lots d'opérations appartiennent, est spécifiée comme suit, les différents champs étant décrits dans 8.4.2 à 8.4.7:

```
OPERATION-PACKAGE ::= CLASS
{
        &Both                   OPERATION OPTIONAL,
        &Consumer               OPERATION OPTIONAL,
        &Supplier               OPERATION OPTIONAL,
        &id                     OBJECT IDENTIFIER UNIQUE OPTIONAL
}
WITH SYNTAX
{
        [OPERATIONS             &Both]
        [CONSUMER INVOKES       &Supplier]
        [SUPPLIER INVOKES       &Consumer]
        [ID                     &id]
}
```

8.4.2 Le champ **&Both** (*couple*) spécifie un ensemble d'opérations **OPERATION** que les deux objets peuvent exécuter. Ce champ peut être omis.

8.4.3 Le champ **&Consumer** (*client*) spécifie un ensemble d'opérations **OPERATION** que l'un des deux objets, le client, peut exécuter. Ce champ peut être omis, auquel cas le client ne pourra exécuter que les opérations spécifiées par le champ **&Both** (*couple*).

8.4.4 Le champ **&Supplier** (*serveur*) spécifie un ensemble d'opérations **OPERATION** que l'un des deux objets, le serveur, peut exécuter. Ce champ peut être omis, auquel cas le serveur ne pourra exécuter que les opérations spécifiées par le champ **&Both** (*couple*).

NOTE – Un opérateur **switch{}** a été défini (voir 10.12) pour permettre de dériver un lot d'opérations d'un autre par simple commutation des rôles.

8.4.5 Le champ **&id** (*identificateur*) spécifie, lorsqu'il est présent, la valeur **OBJECT IDENTIFIER** (*identificateur d'objet*) utilisée pour identifier ce lot, s'il doit par exemple être annoncé ou négocié.

NOTE – Un lot d'opérations qui n'a pas d'identificateur **&id** ne peut être ni annoncé ni négocié.

8.4.6 Toutes les opérations incluses (directement ou indirectement) dans les champs **&Both**, **&Consumer** et **&Supplier** auront des codes d'opération **&operationCode** distincts.

8.4.7 Toutes les erreurs incluses dans les champs **&Errors** de toutes les opérations concernées (voir 8.4.6) auront des codes d'erreur **&errorCode** distincts.

8.5 Lot de connexion

8.5.1 Un lot de connexion est la spécification des opérations et de la qualité de service intervenant dans l'établissement et la libération dynamiques d'une association. Un lot de connexion ne sera spécifié que si des opérations de rattachement et de détachement sont utilisées pour respectivement établir et libérer dynamiquement une association.

La classe d'objets informationnels **CONNECTION-PACKAGE**, à laquelle tous les lots de connexion appartiennent, est spécifiée comme suit, les différents champs étant décrits dans 8.5.2 à 8.5.6:

```
CONNECTION-PACKAGE ::= CLASS
{
        &bind                   OPERATION DEFAULT emptyBind,
        &unbind                 OPERATION DEFAULT emptyUnbind,
        &responderCanUnbind     BOOLEAN DEFAULT FALSE,
        &unbindCanFail          BOOLEAN DEFAULT FALSE,
        &id                     OBJECT IDENTIFIER UNIQUE OPTIONAL
}
WITH SYNTAX
{
        [BIND                   &bind]
        [UNBIND                 &unbind]
        [RESPONDER UNBIND       &responderCanUnbind]
        [FAILURE TO UNBIND      &unbindCanFail]
        [ID                     &id]
}
```

8.5.2 Le champ **&bind** (*rattachement*) spécifie une opération **OPERATION** à exécuter dans le cadre de l'établissement d'une association. Une telle opération doit avoir ses champs **&returnResult** (*retourner le résultat*) et **&alwaysReturns** (*toujours retourner*) à la valeur **TRUE** (*vrai*), et doit avoir une seule erreur présente dans le champ **&Errors.** Si l'association est établie avec succès, l'opération de rattachement renverra un résultat. Si l'établissement de l'association se solde par un échec, l'opération de rattachement renverra une erreur. Si le champ **&bind** n'est pas explicitement mentionné, le lot de connexion y inclura une opération de rattachement vide **emptyBind** (voir 10.2).

8.5.3 Le champ **&unbind** (*détachement*) spécifie une opération **OPERATION** à exécuter dans le cadre de la libération d'une association. Une telle opération doit avoir ses champs **&returnResult** (*retourner le résultat*) et **&alwaysReturns** (*toujours retourner*) à la valeur **TRUE** (*vrai*), et doit avoir une seule erreur définie par le champ **&Errors.** Si l'association est libérée avec succès, l'opération de détachement renverra un résultat. Si la libération de l'association se solde par un échec, l'opération de détachement ne renverra pas de résultat, mais une erreur. Si le champ **&unbind** n'est pas explicitement mentionné, le lot de connexion y inclura une opération de détachement vide **emptyUnbind** (voir 10.3).

8.5.4 Le champ **&responderCanUnbind** (*le répondeur peut détacher*) indique si oui ou non le répondeur de l'association peut invoquer (comme l'initiateur) l'opération de détachement **&unbind**.

8.5.5 Le champ **&unbindCanFail** (*le détachement peut échouer*) indique s'il est ou non possible à l'association de continuer à exister après que l'opération de détachement **&unbind** a signalé une erreur.

8.5.6 Le champ **&id** (*identificateur*) spécifie, lorsqu'il est présent, la valeur **OBJECT IDENTIFIER** (*identificateur d'objet*) utilisée pour identifier le lot de connexion, s'il doit par exemple être annoncé ou négocié.

NOTE – Un lot de connexion qui n'a pas d'identificateur **&id** ne peut être ni annoncé ni négocié.

8.6 Contrat d'association

8.6.1 Un contrat d'association est la spécification des rôles joués par un couple d'objets communicants pouvant établir une association entre eux. La classe d'objets informationnels **CONTRACT**, à laquelle tous les contrats appartiennent, est spécifiée comme suit, les différents champs étant décrits dans 8.6.2 à 8.6.6:

```
CONTRACT ::= CLASS
{
        &connection             CONNECTION-PACKAGE OPTIONAL,
        &OperationsOf           OPERATION-PACKAGE OPTIONAL,
        &InitiatorConsumerOf    OPERATION-PACKAGE OPTIONAL,
        &InitiatorSupplierOf    OPERATION-PACKAGE OPTIONAL,
        &id                     OBJECT IDENTIFIER UNIQUE OPTIONAL
}
WITH SYNTAX
{
        [CONNECTION             &connection]
        [OPERATIONS OF          &OperationsOf]
        [INITIATOR CONSUMER OF  &InitiatorConsumerOf]
        [RESPONDER CONSUMER OF  &InitiatorSupplierOf]
        [ID                     &id]
}
```

8.6.2 La présence d'un champ **&connection** (*connexion*) indique que les associations gouvernées par ce contrat sont dynamiquement établies et libérées respectivement par les opérations de rattachement et de détachement spécifiées dans le cadre d'un lot de connexion, ce lot étant indiqué par le contenu du champ.

8.6.3 Le champ **&OperationsOf** (*opérations de*) spécifie, lorsqu'il est présent, un ou plusieurs lots d'opérations applicables tant que l'association existe, et qui sont symétriques ou alors dans lesquels l'initiateur de l'association peut aussi bien tenir le rôle de client que de serveur.

8.6.4 Le champ **&InitiatorConsumerOf** (*initiateur client de*) spécifie, lorsqu'il est présent, un ou plusieurs lots d'opérations applicables tant que l'association existe, et dans lesquels l'initiateur de l'association tient le rôle de client.

NOTE – L'opérateur **switch{}** a été défini (voir 10.12) pour permettre de dériver un lot d'opérations d'un autre par simple commutation des rôles.

8.6.5 Le champ **&InitiatorSupplierOf** (*initiateur serveur de*) spécifie, lorsqu'il est présent, un ou plusieurs lots d'opérations applicables tant que l'association existe, et dans lesquels l'initiateur tient le rôle de serveur.

8.6.6 Le champ **&id** (*identificateur*) spécifie, lorsqu'il est présent, la valeur **OBJECT IDENTIFIER** (*identificateur d'objet*) utilisée pour identifier ce contrat d'association, s'il doit par exemple être annoncé ou négocié.

NOTE – Un contrat d'association qui n'a pas d'identificateur **&id** ne peut être ni annoncé ni négocié.

8.7 Classe d'objets ROS

8.7.1 Une classe d'objets ROS définit les capacités d'un ensemble d'objets ROS ayant en commun leur aptitude à interagir avec d'autres objets ROS sur la base d'un ensemble donné de contrats. La classe d'objets informationnels **ROS-OBJECT-CLASS** est spécifiée comme suit, les différents champs étant décrits dans 8.7.2 à 8.7.6:

```
ROS-OBJECT-CLASS ::= CLASS
{
        &Is                     ROS-OBJECT-CLASS OPTIONAL,
        &Initiates              CONTRACT OPTIONAL,
        &Responds               CONTRACT OPTIONAL,
        &InitiatesAndResponds   CONTRACT OPTIONAL,
        &id                     OBJECT IDENTIFIER UNIQUE
}
WITH SYNTAX
{
        [IS                     &Is]
        [BOTH                   &InitiatesAndResponds]
        [INITIATES              &Initiates]
        [RESPONDS               &Responds]]
        ID                      &id
}
```

8.7.2 Le champ **&Is** (*hyperensemble*) spécifie, lorsqu'il est présent, un ensemble de classes d'objets ROS constituant des hyperclasses de la classe qui est définie. Les objets ROS de cette dernière auront la capacité de prendre en charge tous les contrats découlant de leur appartenance à chacune des hyperclasses spécifiées, en même temps que les contrats explicitement désignés dans les champs définis dans 8.7.3 à 8.7.5 ci-dessous.

8.7.3 Le champ **&InitiatesAndResponds** (*initie et répond*) spécifie un ensemble de **CONTRACT**s (*contrats*) pour lesquels les objets ROS de la classe peuvent tenir les rôles d'initiateur et de répondeur. Ce champ peut être omis.

8.7.4 Le champ **&Initiates** (*initie*) spécifie un ensemble de **CONTRACT**s (*contrats*) pour lesquels les objets ROS de la classe peuvent tenir le rôle d'initiateur. Ce champ peut être omis, auquel cas les objets ROS auront juste la possibilité d'être les initiateurs des contrats d'association spécifiés par le champ **&InitiatesAndResponds**.

8.7.5 Le champ **&Responds** (*répond*) spécifie un ensemble de contrats pour lesquels les objets de la classe peuvent tenir le rôle de répondeur. Ce champ peut être omis, auquel cas les objets auront juste la possibilité d'être les répondeurs des contrats d'association spécifiés par le champ **&InitiatesAndResponds**.

8.7.6 Le champ **&id** (*identificateur*) spécifie la valeur **OBJECT IDENTIFIER** (*identificateur d'objet*) utilisée pour identifier cette classe d'objets, lorsque par exemple elle est annoncée ou négociée.

8.8 Code

Le type Code fournit les valeurs utilisées pour renseigner les champs **&operationCode** des opérations et les champs **&errorCode** des erreurs. Il est spécifié comme suit:

```
Code ::= CHOICE
{
        local       INTEGER,
        global      OBJECT IDENTIFIER
}
```

8.9 Priorité

Le type **Priority** est spécifié comme suit:

```
Priority ::= INTEGER (0..MAX)
```

Ce paramètre définit la priorité affectée au transfert de l'invocation correspondante (d'une opération) ou au rapport renvoyé en retour, par rapport aux autres invocations (et rapports) échangés entre deux objets ROS.

Plus cette valeur est petite, plus la priorité est grande.

9 Protocole générique ROS

9.1 Introduction

Le présent article fournit une série de définitions pouvant être utilisées pour spécifier des protocoles mettant en œuvre les concepts d'opérations distantes ROS. Les définitions primaires sont:

a) un ensemble paramétré de PDU pour les invocations et les rapports en retour des opérations (**ROS{}**);

b) une PDU paramétrée pour l'invocation et le rapport en retour des opérations de rattachement (**Bind{}**);

c) une PDU paramétrée pour l'invocation et le rapport en retour des opérations de détachement (**Unbind{}**).

De plus, il existe un certain nombre de définitions secondaires sur lesquelles s'appuient les définitions ci-dessus.

9.2 Type ROS

9.2.1 Le type paramétré **ROS{}** (opération distante) constitue une base pour la définition d'une syntaxe abstraite contenant les PDU relatives à l'invocation des opérations, au renvoi des résultats ou des erreurs, et au rejet des PDU non valides. Il est spécifié comme suit:

```
ROS {InvokeId:InvokeIdSet, OPERATION:Invokable, OPERATION:Returnable} ::= CHOICE
{
        invoke          [1]     Invoke {{InvokeIdSet}, {Invokable}},
        returnResult    [2]     ReturnResult {{Returnable}},
        returnError     [3]     ReturnError {{Errors{{Returnable}}}},
        reject          [4]     Reject
}
(CONSTRAINED BY { -- doit être conforme à la définition ci-dessus -- }
! RejectProblem : general-unrecognizedPDU)
```

NOTE – Dans une réalisation pratique, l'emploi de règles de codage non autodélimitantes et non auto-identifiantes peut empêcher d'établir la distinction entre violations de contraintes de niveau externe et violations de contraintes de niveau interne.

9.2.2 Les paramètres ASN.1 suivants doivent être fournis pour constituer un type complètement défini:

a) **InvokeIdSet** (*ensemble d'identificateurs d'invocation*) – Cet ensemble de valeurs du type **InvokeId** (*identificateur d'invocation*) définit les valeurs pouvant être utilisées pour identifier les invocations, et permettant donc de les corréler avec les rapports envoyés en retour. Si, dans une réalisation quelconque, la fonction de corrélation peut être assurée par d'autres moyens, on peut affecter à ce paramètre la valeur **NoInvokeId** (*pas d'identificateur d'invocation*) (voir 9.9);

NOTES

1 Le concepteur de l'application peut décider de laisser ce paramètre sous la forme d'un entier (**INTEGER**) non contraint, ou d'en indiquer l'étendue exacte, ou de le reprendre en tant qu'un des paramètres de la syntaxe abstraite (voir 10.14, 10.15, 10.16).

2 Une application OSI ne supporte pas la valeur **noInvokeId** (voir 9.9.1) pour les PDU de type **Invoke**, **ReturnResult** et **ReturnError**.

b) le paramètre **Invokable** (*invocable*) spécifie un ensemble d'objets qui décrit les opérations pouvant être invoquées;

c) le paramètre **Returnable** (*retournable*) spécifie un ensemble d'objets qui décrit les opérations pour lesquelles il pourra être nécessaire de générer des réponses.

NOTE – Ce paramètre sert à mettre en évidence l'asymétrie de la syntaxe abstraite (voir 10.15 et 10.16). Il peut être choisi dans l'ensemble d'objets informationnels de type Invokable. Les opérations de cet ensemble, dont le champ &alwaysReturns n'a pas la valeur FALSE, doivent être insérées dans ce paramètre.

9.2.3 Si la PDU reçue n'appartient pas à l'ensemble des unités définies, l'entité réceptrice produira une PDU de rejet **Reject{}**, l'argument **problem** de cette unité prenant la valeur d'exception **general-unrecognizedPDU** (*général-PDU non reconnue*).

9.3 Type Invoke

9.3.1 Le type paramétré **Invoke{}** (*invoquer*) fournit une PDU permettant d'invoquer des opérations. Il est spécifié comme suit:

```
Invoke {InvokeId:InvokeIdSet, OPERATION:Operations} ::= SEQUENCE
{
    invokeId        InvokeId    (InvokeIdSet)
                                (CONSTRAINED BY  {-- doit être non ambiguë --}
                                ! RejectProblem        : invoke-duplicateInvocation),
    linkedId        CHOICE      {
                                    present    [0]     IMPLICIT present < InvokeId,
                                    absent     [1]     IMPLICIT NULL
                                }
                                (CONSTRAINED BY  {-- doit identifier une opération en cours  --}
                                ! RejectProblem        : invoke-unrecognizedLinkedId)
                                (CONSTRAINED BY  {-- ayant une ou plusieurs opérations liées --}
                                ! RejectProblem        : invoke-linkedResponseUnexpected)
                                    OPTIONAL,
    opcode          OPERATION.&operationCode
                                ({Operations}
                                !  RejectProblem       : invoke-unrecognizedOperation)
    argument        OPERATION.&ArgumentType
                                ({Operations}   {@opcode}
                                ! RejectProblem        : invoke-mistypedArgument)
                                    OPTIONAL
}
(CONSTRAINED BY  {-- doit être conforme à la définition ci-dessus --}
! RejectProblem        : general-mistypedPDU)
(
    WITH COMPONENTS
    {...,
        linkedId  ABSENT
    }
|   WITH COMPONENTS
    {...,
        linkedId  PRESENT,
        opcode
        (CONSTRAINED BY {-- doit être dans le champ &Linked de l'opération associée --}
        ! RejectProblem        : invoke-unexpectedLinkedOperation)
    }
)
```

NOTE – Dans une réalisation pratique, l'emploi de règles de codage non auto-délimitantes et non auto-identifiantes peut empêcher d'établir la distinction entre violations de contraintes de niveau externe et violations de contraintes de niveau interne.

9.3.2 Les paramètres ASN.1 suivants doivent être fournis pour constituer un type complètement défini:

a) **InvokeIdSet** (*ensemble d'identificateurs d'invocation*) – Cet ensemble de valeurs du type **InvokeId** (*identificateur d'invocation*) définit les valeurs pouvant être utilisées pour identifier les invocations, et permettant donc de les corréler avec les rapports ultérieurs [voir 9.2.2 a)];

b) **Operations** – Opérations pouvant être invoquées.

9.3.3 La PDU résultante peut comporter jusqu'aux quatre arguments suivants:

a) **invokeId** (*identificateur d'invocation*) – Cette composante identifie l'invocation particulière. Elle appartiendra à l'ensemble autorisé par le paramètre **InvokeIdSet**. **InvokeId** ne devra pas déjà être utilisé, (la règle pour déterminer cette double utilisation est une caractéristique du système particulier de mise en correspondance), faute de quoi il y aura production d'une PDU de rejet **Reject{}** (voir 9.6), l'argument **problem** de cette unité prenant la valeur d'exception **invoke-duplicateInvocation** (*invocation dupliquée*).

b) **linkedId** (*identificateur de lien*) – Cette composante, lorsqu'elle est présente, indique que l'invocation présente est liée à une invocation précédente envoyée dans la direction opposée. Il s'agira de l'identificateur **InvokeId** d'une invocation précédente dont les résultats n'auront pas été rapportés, faute de quoi il y aura génération d'une PDU de rejet **Reject{}**, l'argument **problem** de cette unité prenant la valeur d'exception **invoke-unrecognizedLinkedId** (*identificateur de lien non reconnu*). L'opération précédente concernée devra autoriser les opérations liées, faute de quoi il y aura production d'une PDU de rejet **Reject{}**, l'argument problem de cette unité prenant la valeur d'exception **invoke-linkedResponseUnexpected** (*réponse liée non prévue*).

NOTE – Pour une application OSI, la valeur absent: **NULL** n'est pas utilisée pour la composante **linkedID**.

c) **opcode** (*code d'opération*) – Cette composante identifiera par sa valeur **&operationCode** l'une des opérations **Operations**, faute de quoi il y aura génération d'une PDU de rejet **Reject{}**, l'argument **problem** de cette unité prenant la valeur d'exception **invoke-unrecognizedOperation** (*opération non reconnue*). Si la composante d'identification de lien **linkedId** est présente, l'invocation précédente devra se rapporter à une opération permettant d'y lier l'opération concernée, faute de quoi il y aura production d'une PDU de rejet **Reject{}**, l'argument **problem** de cette unité prenant la valeur d'exception **invoke-unexpectedLinkedOperation** (*opération liée non prévue*).

d) **argument** – Cette composante sera présente et aura le type indiqué par le champ **&ArgumentType** de l'opération identifiée par la composante **opcode**, sauf si ce champ est omis ou si le champ **&argumentTypeOptional** indique que le champ **&ArgumentType** peut être facultativement omis, auquel cas cette composante sera elle-même omise. Faute de se conformer à cette condition, il y aura production d'une PDU de rejet **Reject{}**, l'argument problem de cette unité prenant la valeur d'exception **invoke-mistypedArgument** (*argument en violation de contrainte*).

9.3.4 Si la PDU reçue va violer une contrainte, l'entité réceptrice produira une PDU de rejet **Reject{}**, l'argument problem de cette unité prenant la valeur d'exception **general-mistypedPDU** (*général-PDU de type incorrect*).

9.4 Type ReturnResult

9.4.1 Le type paramétré **ReturnResult{}** (*renvoyer le résultat*) fournit une PDU permettant de notifier l'exécution avec succès d'opérations données. Il est spécifié comme suit:

```
ReturnResult {OPERATION:Operations}::= SEQUENCE
{
        invokeId        InvokeId
                                (CONSTRAINED BY   {-- doit être celui d'une opération en cours  --}
                                ! RejectProblem         : returnResult-unrecognizedInvocation)
                                (CONSTRAINED BY   {-- qui retourne en résultat --}
                                ! RejectProblem         : returnResult-resultResponseUnexpected),
        result          SEQUENCE
        {
        opcode          OPERATION.&operationCode
                                ({Operations})(CONSTRAINED BY  {-- identifié par invokeId --}
                                ! RejectProblem         : returnResult-unrecognizedInvocation),
        result          OPERATION.&ResultType ({Operations} {@opcode}
                                ! RejectProblem         : returnResult-mistypedResult)
        }
                        OPTIONAL
}
(CONSTRAINED BY  {-- doit être conforme à la définition ci-dessus --}
! RejectProblem         : general-mistypedPDU)
```

NOTE – Dans une réalisation pratique, l'emploi de règles de codage non autodélimitantes et non autoidentifiantes peut empêcher d'établir la distinction entre violations de contraintes de niveau externe et violations de contraintes de niveau interne.

9.4.2 Un seul paramètre ASN.1 doit être fourni pour constituer un type complètement défini, à savoir **Operations** qui définit les opérations dont il faut notifier l'achèvement avec succès.

9.4.3 La PDU résultante comporte les trois arguments suivants:

a) **invokeId** (*identificateur d'invocation*) – Cet argument identifie l'invocation particulière dont il faudra notifier l'achèvement avec succès. Il s'agira de l'identificateur **InvokeId** d'une invocation antérieure pour laquelle il n'aura pas encore été présenté de rapport de succès ou d'échec, faute de quoi il y aura génération d'une PDU de rejet **Reject{}**, l'argument **problem** de cette unité prenant la valeur d'exception **returnResult-unrecognizedInvocation** (*invocation non reconnue*). L'invocation antérieure en question sera une opération qui retourne un résultat, faute de quoi il y aura production d'une PDU de rejet **Reject{}**, l'argument **problem** de cette unité prenant la valeur d'exception **returnResult-resultResponseUnexpected** (*réponse de succès non prévue*).

b) **opcode** (*code d'opération*) – Cette composante, présente si et seulement si la composante **result** est elle-même présente, identifie au moyen de son code **&operationCode** l'une des opérations, et plus précisément celle qui est indiquée par l'identificateur **invokeId**, faute de quoi il y aura production d'une PDU de rejet **Reject{}**, l'argument **problem** de cette unité prenant la valeur d'exception **returnResult-unrecognizedInvocation** (*invocation non reconnue*).

c) **result** (résultat) – Cette composante sera présente et aura le type indiqué par le champ **&ResultType** de l'opération identifiée par la composante **opcode**, sauf si ce champ est omis, auquel cas cette composante sera elle-même omise. Faute de se conformer à cette condition, il y aura production d'une PDU de rejet **Reject{}**, l'argument **problem** de cette unité prenant la valeur d'exception **returnResult-mistypedResult** (*résultat de type incorrect*).

9.4.4 Si la PDU reçue est de type incorrect, l'entité réceptrice produira une PDU de rejet **Reject{}**, l'argument **problem** de cette unité prenant la valeur d'exception **general-mistypedPDU** (*général-PDU de type incorrect*).

9.5 Type ReturnError

9.5.1 Le type paramétré **ReturnError{}** (*retourner l'erreur*) fournit une PDU permettant de notifier l'échec d'exécution d'opérations données. Il est spécifié comme suit:

```
ReturnError {ERROR:Errors} ::= SEQUENCE
{
        invokeId        InvokeId
                                (CONSTRAINED BY   {-- doit être celui d'une opération en cours --}
                                ! RejectProblem   : returnError-unrecognizedInvocation)
                                (CONSTRAINED BY   {-- qui retourne une erreur --}
                                ! RejectProblem   : returnError-errorResponseUnexpected),
        errcode         ERROR.&errorCode
                                ({Errors}
                                ! RejectProblem   : returnError-unrecognizedError)
                                (CONSTRAINED BY   {-- doit être dans le champ &Errors de l'opération
associée --}
                                ! RejectProblem   : returnError-unexpectedError),
        parameter       ERROR.&ParameterType
                                ({Errors}{@errcode}
                                ! RejectProblem   : returnError-mistypedParameter)
                        OPTIONAL
}
(CONSTRAINED BY  { -- doit être conforme à la définition ci-dessus -- }
! RejectProblem        : general-mistypedPDU)
```

NOTE – Dans une réalisation pratique, l'emploi de règles de codage non autodélimitantes et non auto-identifiantes peut empêcher d'établir la distinction entre violations de contraintes de niveau externe et violations de contraintes de niveau interne.

9.5.2 Un seul paramètre ASN.1 doit être fourni pour constituer un type complètement défini, à savoir **Errors**, qui est l'ensemble des erreurs des opérations dont l'échec d'exécution pourra être notifié (voir 9.10).

9.5.3 La PDU résultante comporte jusqu'aux trois arguments suivants:

a) **invokeId** (*identificateur d'invocation*) – Cet argument identifie l'invocation particulière dont il faudra notifier l'échec d'exécution. Il s'agira de l'identificateur **InvokeId** d'une invocation antérieure pour laquelle il n'aura pas encore été présenté de rapport de succès ou d'échec, faute de quoi il y aura génération d'une PDU de rejet **Reject{}**, l'argument **problem** de cette unité prenant la valeur d'exception **returnError-unrecognizedInvocation** (*invocation non reconnue*). L'invocation antérieure en question sera une opération qui notifie une erreur, faute de quoi il y aura production d'une PDU de rejet **Reject{}**, l'argument **problem** de cette unité prenant la valeur d'exception **returnError-errorResponseUnexpected** (*réponse d'erreur non prévue*).

b) **errcode** (*code d'erreur*) – Cette composante identifie au moyen de son code **&errorCode** l'une des erreurs de l'ensemble **Errors**, faute de quoi il y aura production d'une PDU de rejet **Reject{}**, l'argument **problem** de cette unité prenant la valeur d'exception **returnError-unrecognizedError** (*erreur non reconnue*). L'erreur sera l'une de celles qui apparaissent dans le champ **&Errors** de l'opération identifiée par l'identificateur **invokeId**, faute de quoi il y aura production d'une PDU de rejet **Reject{}**, l'argument **problem** de cette unité prenant la valeur d'exception **returnError-unexpectedError** (*erreur non prévue*).

c) **parameter** (*paramètre*) – Cette composante sera présente et aura le type indiqué par le champ **&ParameterType** de l'erreur identifiée par la composante **errcode**, sauf si ce champ est omis, auquel cas cette composante sera elle-même omise. Faute de se conformer à cette condition, il y aura production d'une PDU de rejet **Reject{}**, l'argument problem de cette unité prenant la valeur d'exception **returnError-mistypedParameter** (*paramètre de type incorrect*).

9.5.4 Si la PDU reçue est de type incorrect, l'entité réceptrice produira une PDU de rejet **Reject{}**, l'argument **problem** de cette unité prenant la valeur d'exception **general-mistypedPDU** (*général-PDU de type incorrect*).

9.6 Type Reject

9.6.1 Le type **Reject** (*rejet*) fournit une PDU permettant de notifier l'utilisation erronée d'autres PDU **ROS{}**. Il est spécifié comme suit:

```
Reject ::= SEQUENCE
{
        invokeId        InvokeId,
        problem         CHOICE
            {
                general             [0]     GeneralProblem,
                invoke              [1]     InvokeProblem,
                returnResult        [2]     ReturnResultProblem,
                returnError         [3]     ReturnErrorProblem
            }
}
(CONSTRAINED BY  {-- doit être conforme à la définition ci-dessus --}
! RejectProblem        : general-mistypedPDU)
```

NOTE – Dans une réalisation pratique, l'emploi de règles de codage non autodélimitantes et non auto-identifiantes peut empêcher d'établir la distinction entre violation de contraintes de niveau externe et violations de contraintes de niveau interne.

9.6.2 La PDU résultante comporte les deux arguments suivants:

a) **invokeId** (*identificateur d'invocation*) – Il s'agit de l'identificateur **InvokeId** de la PDU rejetée, sauf si **invokeId** ne peut être déterminé, auquel cas le champ contient la valeur **noInvokeId** (voir 9.9).

b) **problem** (*problème*) – Cette composante identifie le problème particulier ayant entraîné le rejet de la PDU. Les quatre catégories de problèmes sont spécifiées aux 9.6.3 à 9.6.6.

9.6.3 **GeneralProblem** (*problème général*) est un problème fondamental se rapportant à la forme ou à la structure de la PDU d'opération distante **ROS{}**. Les différents messages possibles sont spécifiés comme suit:

```
GeneralProblem ::=  INTEGER
{
        unrecognizedPDU (0),
        mistypedPDU (1),
        badlyStructuredPDU (2)
}
```

et correspondent aux cas suivants:

a) **unrecognizedPDU** (*PDU non reconnue*) – L'étiquette de la PDU ne correspond à aucune des possibilités prévues au 9.2;

b) **mistypedPDU** (*PDU de type incorrect*) – La structure de la PDU n'est pas conforme à la définition appropriée;

c) **badlyStructuredPDU** (*PDU mal structurée*) – La structure de la PDU ne peut être déterminée à partir de la syntaxe abstraite prévue.

NOTE – Dans certaines correspondances, ces problèmes seront identifiés et traités dans le cadre de l'infrastructure de communication.

9.6.4 **InvokeProblem** (*problème d'invocation*) indique que l'une des composantes d'une PDU d'invocation **Invoke{}** était erronée. Les différents messages possibles sont spécifiés comme suit:

```
InvokeProblem ::= INTEGER
{
        duplicateInvocation (0),
        unrecognizedOperation (1),
        mistypedArgument (2),
        resourceLimitation (3),
        releaseInProgress (4),
        unrecognizedLinkedId (5),
        linkedResponseUnexpected (6),
        unexpectedLinkedOperation (7)
}
```

et correspondent aux cas suivants:

a) **duplicateInvocation** (*invocation dupliquée*) [voir 9.3.3 a)];

b) **unrecognizedOperation** (*opération non reconnue*) [voir 9.3.3 c)];

c) **mistypedArgument** (*argument de type incorrect*) [voir 9.3.3 d)];

d) **resourceLimitation** (*limitation des ressources*) – L'exécutant prévu ne désire pas exécuter l'opération en raison de la limitation des ressources;

e) **releaseInProgress** (*libération en cours*) – L'exécutant prévu ne désire pas exécuter l'opération car il est sur le point de libérer l'association;

f) **unrecognizedLinkedId** (*identificateur de lien non reconnu*) – [voir 9.3.3 b)];

g) **linkedResponseUnexpected** (*réponse liée non prévue*) – [voir 9.3.3 b)];

h) **unexpectedLinkedOperation** (*opération liée non prévue*) – [voir 9.3.3 c)].

9.6.5 **ReturnResultProblem** (*problème de retour de résultat*) indique que l'une des composantes d'une PDU de demande de retour de résultat **ReturnResult{}** était erronée. Les différents messages possibles sont spécifiés comme suit:

```
ReturnResultProblem ::= INTEGER
{
        unrecognizedInvocation (0),
        resultResponseUnexpected (1),
        mistypedResult (2)
}
```

et correspondent aux cas suivants:

a) **unrecognizedInvocation** (*invocation non reconnue*) [voir 9.4.3 a)];

b) **resultResponseUnexpected** (*retour de résultat non prévu*) [voir 9.4.3 a)];

c) **mistypedResult** (*résultat de type incorrect*) [voir 9.4.3 b) et 9.4.3 c)].

9.6.6 **ReturnErrorProblem** (*problème de retour d'erreur*) indique que l'une des composantes d'une PDU de demande de retour d'erreur **ReturnError{}** était erronée. Les différents messages possibles sont spécifiés comme suit:

```
ReturnErrorProblem ::= INTEGER
{
        unrecognizedInvocation (0),
        errorResponseUnexpected (1),
        unrecognizedError (2),
        unexpectedError (3),
        mistypedParameter (4)
}
```

et correspondent aux cas suivants:

a) **unrecognizedInvocation** (*invocation non reconnue*) [voir 9.5.3 a)];

b) **errorResponseUnexpected** (*retour de résultat non prévu*) [voir 9.5.3 a)];

c) **unrecognizedError** (*erreur non reconnue*) [voir 9.5.3 b)];

d) **unexpectedError** (*erreur non prévue*) [voir 9.5.3 b)];

e) **mistypedParameter** (*paramètre de type incorrect*) [voir 9.5.3 c)].

9.6.7 Si une telle PDU reçue va être de type incorrect, aucune nouvelle PDU de rejet **Reject{}** ne doit être produite.

9.7 Type RejectProblem

L'entier **RejectProblem** (*problème du rejet*) donne la valeur de code d'erreur générée en cas de transgression d'un type quelconque de contrainte portant sur la définition d'une PDU.

```
RejectProblem ::= INTEGER
{
        general-unrecognizedPDU (0),
        general-mistypedPDU (1),
        general-badlyStructuredPDU (2),
        invoke-duplicateInvocation (10),
        invoke-unrecognizedOperation (11),
        invoke-mistypedArgument (12),
        invoke-resourceLimitation (13),
        invoke-releaseInProgress (14),
        invoke-unrecognizedLinkedId (15),
        invoke-linkedResponseUnexpected (16),
        invoke-unexpectedLinkedOperation (17),
        returnResult-unrecognizedInvocation (20),
        returnResult-resultResponseUnexpected (21),
        returnResult-mistypedResult (22),
        returnError-unrecognizedInvocation (30),
        returnError-errorResponseUnexpected (31),
        returnError-unrecognizedError (32),
        returnError-unexpectedError (33),
        returnError-mistypedParameter (34)
}
```

9.7.1 Lorsqu'une erreur est signalée, le système doit réagir en envoyant une PDU de rejet **Reject{}**. Lorsque l'erreur signalée est indiquée par l'identificateur de problème du rejet «α-β», α et β étant deux chaînes de caractères, la composante **problem** de la PDU **Reject{}** prendra la valeur «α:β».

9.8 Type InvokeId

9.8.1 Le type **InvokeId** (*identificateur d'invocation*), qui définit les valeurs pouvant être utilisées pour identifier une invocation particulière d'une opération quelconque, est spécifié comme suit:

```
InvokeId ::= CHOICE
{
        present   INTEGER,
        absent    NULL
}
```

9.8.2 Lorsque l'identificateur **InvokeId** est présent, il aura une valeur de type entier **INTEGER**; lorsqu'il est absent, une valeur **NULL** (*néant*) est inscrite à sa place.

9.9 Valeur NoInvokeId

9.9.1 La valeur **noInvokeId** (*pas d'identificateur d'invocation*) est celle que prend le champ **InvokeId** lorsqu'une valeur entière **INTEGER** est soit inutile soit non disponible. Elle est spécifiée comme suit:

```
noInvokeId InvokeId ::= absent:NULL
```

9.9.2 L'ensemble **NoInvokeId** est une collection de valeurs du type **InvokeId** comprenant la seule valeur **noInvokeId**. Il est spécifié comme suit:

```
NoInvokeId InvokeId ::= {noInvokeId}
```

9.10 Ensemble Errors

L'ensemble paramétré d'erreurs **Errors{}** (*erreurs*), ayant pour paramètre ASN.1 un ensemble d'opérations, est l'ensemble de toutes les erreurs se trouvant dans les champs **&Errors** de ces opérations. Il est spécifié comme suit:

```
Errors {OPERATION:Operations} ERROR ::= {Operations.&Errors}
```

9.11 Type Bind

Le type paramétré **Bind{}** (*rattachement*), ayant pour paramètre ASN.1 une seule opération de rattachement, fournit la PDU permettant d'invoquer cette opération, d'en rapporter le résultat, ou d'en rapporter une erreur. Il est spécifié comme suit:

```
Bind {OPERATION:operation} ::= CHOICE
{
        bind-invoke     [16]    OPERATION.&ArgumentType({operation}),
        bind-result     [17]    OPERATION.&ResultType({operation}),
        bind-error      [18]    OPERATION.&Errors.&ParameterType({operation})
}
```

9.12 Type Unbind

Le type paramétré **Unbind{}** (*détachement*), ayant pour paramètre ASN.1 une seule opération de détachement, fournit la PDU permettant d'invoquer cette opération, d'en rapporter le résultat, ou d'en rapporter une erreur. Il est spécifié comme suit:

```
Unbind {OPERATION:operation} ::= CHOICE
{
        unbind-invoke     [19]    OPERATION.&ArgumentType({operation}),
        unbind-result     [20]    OPERATION.&ResultType({operation}),
        unbind-error      [21]    OPERATION.&Errors.&ParameterType({operation})
}
```

10 Définitions utiles

10.1 Introduction

Le présent article fournit une série de définitions pouvant servir aux concepteurs des applications basées sur les principes d'opérations distantes ROS. Ces définitions couvrent:

a) des opérations d'intérêt général (**emptyBind**, **emptyUnbind**, **no-op**) avec leurs erreurs associées;

b) des définitions paramétrées qui fournissent les ensembles d'opérations mises en œuvre dans certains lots d'opérations (**ConsumerPerforms{}**, **SupplierPerforms{}**, **AllOperations{}**) avec certaines définitions auxiliaires;

c) une définition paramétrée qui permet de dériver une opération d'une autre en en modifiant le code d'opération (**recode{}**);

d) des définitions paramétrées qui permettent de définir des lots d'opérations en intervertissant les rôles tenus dans un autre, ou en combinant plusieurs autres lots (**switch{}**, **combine{}**);

e) des types paramétrés qui peuvent servir à définir des syntaxes abstraites correspondant à un lot d'opérations (**ROS-SingleAS{}**, **ROS-ConsumerAS{}**, **ROS-SupplierAS{}**).

10.2 Opération emptyBind

L'opération **emptyBind** (*rattachement vide*) est l'opération de rattachement la plus simple, et tient lieu d'opération de rattachement par défaut lorsqu'une opération de rattachement n'a pas été explicitement spécifiée pour un lot de connexion donné. Cette opération n'a ni argument ni valeurs de résultat, et n'a qu'une seule erreur possible, **refuse** (*refus*, voir 10.4), qui correspond au refus de l'association. L'opération est synchrone, c'est-à-dire qu'aucune autre opération synchrone ne peut être invoquée tant que le résultat de l'opération de rattachement n'a pas été reçu en retour. L'opération **emptyBind** est spécifiée comme suit:

```
emptyBind OPERATION ::= {ERRORS {refuse} SYNCHRONOUS TRUE}
```

10.3 Opération empty Unbind

L'opération **emptyUnbind** (*détachement vide*) est l'opération de détachement la plus simple, et tient lieu d'opération de détachement par défaut lorsqu'une opération de détachement n'a pas été explicitement spécifiée pour un lot de connexion donné. Cette opération n'a ni argument ni valeurs de résultat ou d'erreur. L'opération est synchrone, c'est-à-dire qu'elle ne peut être invoquée tant que le résultat de toute autre opération synchrone en cours n'a pas été reçu en retour. L'opération **emptyUnbind** est spécifiée comme suit:

```
emptyUnbind OPERATION ::= {SYNCHRONOUS TRUE}
```

10.4 Notification refuse

L'erreur **refuse** (*refus*) est une notification de refus d'une demande quelconque, sans qu'un motif ne soit avancé. Elle est spécifiée comme suit:

```
refuse ERROR ::= {CODE  local:-1}
```

10.5 Opération no-op

10.5.1 L'opération **no-op** (*opération nulle*) ne fait rien. Elle est spécifiée comme suit:

```
no-op OPERATION ::=
{
        ALWAYS RESPONDS FALSE
        CODE  local:-1
}
```

10.5.2 L'opération ne renvoie aucune notification en retour.

10.6 Ensemble Forward

L'ensemble paramétré d'opérations **Forward{}** (*direct*), ayant pour paramètre ASN.1 un ensemble d'opérations **OperationSet**, est formé par l'ensemble lui-même, étendu en ajoutant aux opérations d'origine toutes les opérations ayant la même directionnalité (sens direct) qui leur sont indirectement liées. Il est spécifié comme suit:

```
Forward {OPERATION:OperationSet} OPERATION ::=
{
        OperationSet |
        OperationSet.&Linked.&Linked |
        OperationSet.&Linked.&Linked.&Linked.&Linked
}
```

On suppose qu'il n'existe pas d'opérations au-delà du quatrième étage de lien qui ne soient déjà présentes à des niveaux précédents. Si, pour un ensemble **OperationSet** donné, l'hypothèse ne tient pas, il faudra alors recourir à une autre méthode pour identifier l'ensemble complet des opérations mises en jeu.

10.7 Ensemble Reverse

L'ensemble paramétré d'opérations **Reverse{}** (*inverse*), ayant pour paramètre ASN.1 un ensemble d'opérations **OperationSet**, est formé par l'ensemble de toutes les opérations ayant la directionnalité inverse (sens inverse) qui sont directement ou indirectement liées aux opérations de l'ensemble d'origine. Il est spécifié comme suit:

```
Reverse {OPERATION:OperationSet} OPERATION ::= {Forward{{OperationSet.&Linked}}}
```

On suppose qu'il n'existe pas d'opérations au-delà du cinquième étage de lien qui ne soient déjà présentes à des niveaux précédents. Si, pour un ensemble **OperationSet** donné, l'hypothèse ne tient pas, il faudra alors recourir à une autre méthode pour identifier l'ensemble complet des opérations mises en jeu.

10.8 Ensemble ConsumerPerforms

L'ensemble paramétré d'opérations **ConsumerPerforms{}** (*le client exécute*), ayant pour paramètre ASN.1 un lot d'opérations **package**, est formé par l'ensemble de toutes les opérations que le client peut exécuter. Il est spécifié comme suit:

```
ConsumerPerforms {OPERATION-PACKAGE:package} OPERATION ::=
{
        Forward{{package.&Consumer}} |
        Forward{{package.&Both}} |
        Reverse{{package.&Supplier}} |
        Reverse{{package.&Both}}
}
```

Cette spécification ne peut être utilisée que si les hypothèses associées à la définition des ensembles **Forward{}** et **Reverse{}** et portant sur les opérations liées sont valides.

10.9 Ensemble SupplierPerforms

L'ensemble paramétré d'opérations **SupplierPerforms{}** (*le serveur exécute*), ayant pour paramètre ASN.1 un lot d'opérations package, est formé par l'ensemble de toutes les opérations que le serveur peut exécuter. Il est spécifié comme suit:

```
SupplierPerforms {OPERATION-PACKAGE:package} OPERATION ::=
{
        Forward{{package.&Supplier}} |
        Forward{{package.&Both}} |
        Reverse{{package.&Consumer}} |
        Reverse{{package.&Both}}
}
```

Cette spécification ne peut être utilisée que si les hypothèses associées à la définition des ensembles **Forward{}** et **Reverse{}** et portant sur les opérations liées sont valides.

10.10 Ensemble AllOperations

L'ensemble paramétré d'opérations **AllOperations{}** (*toutes opérations*), ayant pour paramètre ASN.1 un lot d'opérations **package**, est formé par l'ensemble de toutes les opérations mise en jeu dans le lot. Il est spécifié comme suit:

```
AllOperations {OPERATION-PACKAGE:package} OPERATION ::=
{
        ConsumerPerforms {package} | SupplierPerforms {package}
}
```

Cette spécification ne peut être utilisée que si les hypothèses associées à la définition des ensembles **Forward{}** et **Reverse{}** et portant sur les opérations liées sont valides.

10.11 Opération recode

L'opération paramétrée **recode{}** (*recoder*), ayant pour premier paramètre ASN.1 une opération, est identique en tous points à cette opération excepté son champ **&code** qui porte la valeur du second paramètre ASN.1 **code**. Elle est spécifiée comme suit:

```
recode {OPERATION:operation, Code:code} OPERATION ::=
{
        ARGUMENT                operation.&ArgumentType
                OPTIONAL                operation.&argumentTypeOptional
        RETURN RESULT           operation.&returnResult
        RESULT                  operation.&ResultType,
                OPTIONAL                operation.&resultTypeOptional
        ERRORS                  {operation.&Errors}
        ALWAYS RESPONDS         operation.&alwaysReturns
        LINKED                  {operation.&Linked}
        SYNCHRONOUS             operation.&synchronous
        INVOKE PRIORITY         {operation.&InvokePriority}
        RESULT-PRIORITY         {operation.&ResultPriority}
        CODE                    code
}
```

10.12 Lot d'opérations switch

Le lot d'opérations paramétré **switch{}** (*intervertir*), ayant pour premier paramètre ASN.1 un lot d'opérations **package**, est identique en tous points à ce lot, sauf que les rôles de client et de serveur y sont intervertis et que son champ **&id** porte la valeur du second paramètre ASN.1 **id**. Il est spécifié comme suit:

```
switch {OPERATION-PACKAGE:package, OBJECT IDENTIFIER:id} OPERATION-PACKAGE ::=
{
        OPERATIONS              {package.&Both}
        SUPPLIER INVOKES        {package.&Supplier}
        CONSUMER INVOKES        {package.&Consumer}
        ID                      id
}
```

10.13 Lot d'opérations combine

10.13.1 Le lot d'opérations paramétré **combine{}** (*combiner*) est la combinaison de plusieurs lots d'opérations en un seul. Il est spécifié comme suit:

```
combine {OPERATION-PACKAGE:ConsumerConsumes, OPERATION-PACKAGE:ConsumerSupplies,
OPERATION-PACKAGE:base} OPERATION-PACKAGE ::=
{
        OPERATIONS              {ConsumerConsumes.&Both | ConsumerSupplies.&Both}
        SUPPLIER INVOKES        {ConsumerConsumes.&Supplier   | ConsumerSupplies.&Consumer}
        CONSUMER INVOKES        {ConsumerConsumes.&Consumer | ConsumerSupplies.&Supplier}
        ID                      base.&id
}
```

10.13.2 Les paramètres ASN.1 nécessaires à la définition complète du lot d'opérations sont les suivants:

a) **ConsumerConsumes** (*client client*) – Ensemble des lots d'opérations dans lesquels le client du lot résultant final tiendra le rôle de client;

b) **ConsumerSupplies** (*client serveur*) – Ensemble des lots d'opérations dans lesquels le client du lot résultant final tiendra le rôle de serveur;

c) **base** – Lot d'opérations, généralement incomplètement défini, dont le champ d'identification **&id** sert à définir le lot résultant final.

NOTE – Aucune opération ne sera généralement définie dans le lot d'opérations base; même si c'était le cas, ces opérations n'apparaîtraient pas dans le lot résultant final, à moins qu'elles ne soient citées dans les ensembles **ConsumerConsumes** ou **ConsumerSupplies**.

10.14 Type ROS-SingleAS

Le type paramétré **ROS-SingleAS{}** (*syntaxe abstraite commune de ROS*) est la base d'une notation de syntaxe abstraite permettant d'invoquer toutes les opérations d'un lot **package** et d'en rapporter les résultats, en utilisant les identificateurs d'un ensemble **InvokeIdSet**. Il est spécifié comme suit:

```
ROS-SingleAS {InvokeId:InvokeIdSet, OPERATION-PACKAGE:package} ::=
        ROS {{InvokeIdSet}, {AllOperations{package}}, {AllOperations{package}}}
```

10.15 Type ROS-ConsumerAS

Le type paramétré **ROS-ConsumerAS{}** (*syntaxe abstraite de client ROS*) est la base d'une notation de syntaxe abstraite permettant, dans un lot **package**, d'invoquer toutes les opérations exécutables par le client et de rapporter les résultats de toutes les opérations exécutables par le serveur, en utilisant les identificateurs d'un ensemble **InvokeIdSet**. Il est spécifié comme suit:

```
ROS-ConsumerAS {InvokeId:InvokeIdSet, OPERATION-PACKAGE:package} ::=
        ROS {{InvokeIdSet}, {ConsumerPerforms{package}}, {SupplierPerforms{package}}}
```

10.16 Type ROS-SupplierAS

Le type paramétré **ROS-SupplierAS{}** (*syntaxe abstraite de serveur ROS*) est la base d'une notation de syntaxe abstraite permettant, dans un lot **package**, d'invoquer toutes les opérations exécutables par le serveur et de rapporter les résultats de toutes les opérations exécutables par le client, en utilisant les identificateurs d'un ensemble **InvokeIdSet**. Il est spécifié comme suit:

```
ROS-SupplierAS {InvokeId:InvokeIdSet, OPERATION-PACKAGE:package} ::=
        ROS {{InvokeIdSet}, {SupplierPerforms{package}}, {ConsumerPerforms{package}}}
```

Annexe A

Modules ASN.1

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

```
Remote-Operations-Information-Objects {joint-iso-itu-t remote-operations(4) informationObjects(5) version1(0)}
DEFINITIONS ::=
BEGIN
-- exporter tout
IMPORTS emptyBind, emptyUnbind FROM Remote-Operations-Useful-Definitions {joint-iso-itu-t remote-operations(4)
useful-definitions(7) version1(0)};

OPERATION ::= CLASS
{
        &ArgumentType               OPTIONAL,
        &argumentTypeOptional       BOOLEAN OPTIONAL,
        &returnResult               BOOLEAN DEFAULT TRUE,
        &ResultType                 OPTIONAL,
        &resultTypeOptional         BOOLEAN OPTIONAL,
        &Errors                     ERROR OPTIONAL,
        &Linked                     OPERATION OPTIONAL,
        &synchronous                BOOLEAN DEFAULT FALSE,
        &alwaysReturns              BOOLEAN DEFAULT TRUE,
        &InvokePriority             Priority OPTIONAL,
        &ResultPriority             Priority OPTIONAL,
        &operationCode              Code UNIQUE OPTIONAL
}
WITH SYNTAX
{
        [ARGUMENT                   &ArgumentType    [OPTIONAL   &argumentTypeOptional]]
        [RESULT                     &ResultType      [OPTIONAL   &resultTypeOptional]]
        [RETURN RESULT              &returnResult]
        [ERRORS                     &Errors]
        [LINKED                     &Linked]
        [SYNCHRONOUS                &synchronous]
        [ALWAYS RESPONDS            &alwaysReturns]
        [INVOKE PRIORITY            &InvokePriority]
        [RESULT-PRIORITY            &ResultPriority]
        [CODE                       &operationCode]
}
ERROR ::= CLASS
{
        &ParameterType              OPTIONAL,
        &parameterTypeOptional      BOOLEAN OPTIONAL,
        &ErrorPriority              Priority OPTIONAL,
        &errorCode                  Code UNIQUE OPTIONAL
}
WITH SYNTAX
{
        [PARAMETER                  &ParameterType   [OPTIONAL   &parameterTypeOptional]]
        [PRIORITY                   &ErrorPriority]
        [CODE                       &errorCode]
}
OPERATION-PACKAGE ::= CLASS
{
        &Both                       OPERATION OPTIONAL,
        &Consumer                   OPERATION OPTIONAL,
        &Supplier                   OPERATION OPTIONAL,
        &id                         OBJECT IDENTIFIER UNIQUE OPTIONAL
}
-- suite page suivante
```

```
WITH SYNTAX
{
        [OPERATIONS                     &Both]
        [CONSUMER INVOKES               &Supplier]
        [SUPPLIER INVOKES               &Consumer]
        [ID                             &id]
}
CONNECTION-PACKAGE ::= CLASS
{
        &bind                           OPERATION DEFAULT emptyBind,
        &unbind                         OPERATION DEFAULT emptyUnbind,
        &responderCanUnbind             BOOLEAN DEFAULT FALSE,
        &unbindCanFail                  BOOLEAN DEFAULT FALSE,
        &id                             OBJECT IDENTIFIER UNIQUE OPTIONAL
}
WITH SYNTAX
{
        [BIND                           &bind]
        [UNBIND                         &unbind]
        [RESPONDER UNBIND               &responderCanUnbind]
        [FAILURE TO UNBIND              &unbindCanFail]
        [ID                             &id]
}
CONTRACT ::= CLASS
{
        &connection                     CONNECTION-PACKAGE OPTIONAL,
        &OperationsOf                   OPERATION-PACKAGE OPTIONAL,
        &InitiatorConsumerOf            OPERATION-PACKAGE OPTIONAL,
        &InitiatorSupplierOf            OPERATION-PACKAGE OPTIONAL,
        &id                             OBJECT IDENTIFIER UNIQUE OPTIONAL
}
WITH SYNTAX
{
        [CONNECTION                     &connection]
        [OPERATIONS OF                  &OperationsOf]
        [INITIATOR CONSUMER OF          &InitiatorConsumerOf]
        [RESPONDER CONSUMER OF          &InitiatorSupplierOf]
        [ID                             &id]
}
ROS-OBJECT-CLASS ::= CLASS
{
        &Is                             ROS-OBJECT-CLASS OPTIONAL,
        &Initiates                      CONTRACT OPTIONAL,
        &Responds                       CONTRACT OPTIONAL,
        &InitiatesAndResponds           CONTRACT OPTIONAL,
        &id                             OBJECT IDENTIFIER UNIQUE
}
WITH SYNTAX
{
        [IS                             &Is]
        [BOTH                           &InitiatesAndResponds]
        [INITIATES                      &Initiates]
        [RESPONDS                       &Responds]
        ID                              &id
}
Code ::= CHOICE
{
        local                           INTEGER,
        global                          OBJECT IDENTIFIER
}
Priority ::= INTEGER (0..MAX)
END -- fin des spécifications d'objets informationnels
```

```
Remote-Operations-Generic-ROS-PDUs {joint-iso-itu-t remote-operations(4) generic-ROS-PDUs(6) version1(0)}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
-- exporter tout
IMPORTS OPERATION, ERROR FROM Remote-Operations-Information-Objects {joint-iso-itu-t remote-operations(4)
informationObjects(5) version1(0)};

ROS {InvokeId:InvokeIdSet, OPERATION:Invokable, OPERATION:Returnable} ::= CHOICE
{
        invoke          [1]   Invoke {{InvokeIdSet}, {Invokable}},
        returnResult    [2]   ReturnResult {{Returnable}},
        returnError     [3]   ReturnError {{Errors{{Returnable}}}},
        reject          [4]   Reject
}
(CONSTRAINED BY  {-- doit être conforme à la définition ci-dessus --}
! RejectProblem        : general-unrecognizedPDU)

Invoke {InvokeId:InvokeIdSet, OPERATION:Operations} ::= SEQUENCE
{
        invokeId        InvokeId    (InvokeIdSet)
                                    (CONSTRAINED BY  {-- doit être non ambigu --}
                                    ! RejectProblem        : invoke-duplicateInvocation),
        linkedId        CHOICE {
                                        present   [0]  IMPLICIT present <   InvokeId,
                                        absent    [1]  IMPLICIT NULL
                                            }
                                    (CONSTRAINED BY  {-- doit identifier une opération en cours  --}
                                    ! RejectProblem        : invoke-unrecognizedLinkedId)
                                    (CONSTRAINED BY  {-- ayant une ou plusieurs opérations liées --}
                                    ! RejectProblem        : invoke-linkedResponseUnexpected)
                                        OPTIONAL,
        opcode          OPERATION.&operationCode
                                    ({Operations}
                                    ! RejectProblem        : invoke-unrecognizedOperation),
        argument        OPERATION.&ArgumentType
                                    ({Operations} {@opcode}
                                    ! RejectProblem        : invoke-mistypedArgument)
                                        OPTIONAL
}
(CONSTRAINED BY  {-- doit être conforme à la définition ci-dessus --}
! RejectProblem        : general-mistypedPDU)
(
        WITH COMPONENTS
        {...,
            linkedId ABSENT
        }
|       WITH COMPONENTS
        {...,
            linkedId PRESENT,
            opcode
            (CONSTRAINED BY  {-- doit être dans le champ &Linked de l'opération associée --}
            ! RejectProblem        : invoke-unexpectedLinkedOperation)
        }
)
-- suite page suivante
```

```
ReturnResult {OPERATION:Operations}::= SEQUENCE
{
        invokeId        InvokeId
                                (CONSTRAINED BY {-- doit être celui d'une opération en cours --}
                                ! RejectProblem         : returnResult-unrecognizedInvocation)
                                (CONSTRAINED BY {-- qui retourne un résultat --}
                                ! RejectProblem         : returnResult-resultResponseUnexpected),
        result          SEQUENCE
        {
        opcode          OPERATION.&operationCode
                                ({Operations})(CONSTRAINED BY {-- identifiées par invokeId --}
                                ! RejectProblem         : returnResult-unrecognizedInvocation),
        result          OPERATION.&ResultType
                                ({Operations} {@opcode}
                                ! RejectProblem         : returnResult-mistypedResult)
        }
                        OPTIONAL
}
(CONSTRAINED BY {-- doit être conforme à la définition ci-dessus --}
! RejectProblem         : general-mistypedPDU)

ReturnError {ERROR:Errors} ::= SEQUENCE
{
        invokeId        InvokeId
                                (CONSTRAINED BY {-- doit être celui d'une opération en cours --}
                                ! RejectProblem         : returnError-unrecognizedInvocation)
                                (CONSTRAINED BY {-- qui retourne une erreur --}
                                ! RejectProblem         : returnError-errorResponseUnexpected),
        errcode         ERROR.&errorCode
                                ({Errors}
                                ! RejectProblem : returnError-unrecognizedError)
                                (CONSTRAINED BY {-- doit être dans le champ &Errors de l'opération associée --}
                                ! RejectProblem : returnError-unexpectedError),
        parameter       ERROR.&ParameterType
                                ({Errors}{@errcode}
                                ! RejectProblem : returnError-mistypedParameter) OPTIONAL
}
(CONSTRAINED BY {-- doit être conforme à la définition ci-dessus --}
! RejectProblem         : general-mistypedPDU)

Reject ::= SEQUENCE
{
        invokeId        InvokeId,
        problem         CHOICE
            {
                        general         [0]  GeneralProblem,
                        invoke          [1]  InvokeProblem,
                        returnResult    [2]  ReturnResultProblem,
                        returnError     [3]  ReturnErrorProblem
            }
}
(CONSTRAINED BY {-- doit être conforme à la définition ci-dessus --}
! RejectProblem         : general-mistypedPDU)

GeneralProblem ::= INTEGER
{
        unrecognizedPDU (0),
        mistypedPDU (1),
        badlyStructuredPDU (2)
}
```

-- suite page suivante

```
InvokeProblem ::= INTEGER
{
        duplicateInvocation (0),
        unrecognizedOperation (1),
        mistypedArgument (2),
        resourceLimitation (3),
        releaseInProgress (4),
        unrecognizedLinkedId (5),
        linkedResponseUnexpected (6),
        unexpectedLinkedOperation (7)
}

ReturnResultProblem ::= INTEGER
{
        unrecognizedInvocation (0),
        resultResponseUnexpected (1),
        mistypedResult (2)
}

ReturnErrorProblem ::= INTEGER
{
        unrecognizedInvocation (0),
        errorResponseUnexpected (1),
        unrecognizedError (2),
        unexpectedError (3),
        mistypedParameter (4)
}

RejectProblem ::= INTEGER
{
        general-unrecognizedPDU (0),
        general-mistypedPDU (1),
        general-badlyStructuredPDU (2),
        invoke-duplicateInvocation (10),
        invoke-unrecognizedOperation (11),
        invoke-mistypedArgument (12),
        invoke-resourceLimitation (13),
        invoke-releaseInProgress (14),
        invoke-unrecognizedLinkedId (15),
        invoke-linkedResponseUnexpected (16),
        invoke-unexpectedLinkedOperation (17),
        returnResult-unrecognizedInvocation (20),
        returnResult-resultResponseUnexpected (21),
        returnResult-mistypedResult (22),
        returnError-unrecognizedInvocation (30),
        returnError-errorResponseUnexpected (31),
        returnError-unrecognizedError (32),
        returnError-unexpectedError (33),
        returnError-mistypedParameter (34)
}

InvokeId ::= CHOICE
{
        present    INTEGER,
        absent     NULL
}

noInvokeId InvokeId ::= absent:NULL

NoInvokeId InvokeId ::= {noInvokeId}

Errors {OPERATION:Operations} ERROR ::= {Operations.&Errors}
```

-- suite page suivante

```
Bind {OPERATION:operation} ::= CHOICE
{
        bind-invoke    [16]    OPERATION.&ArgumentType({operation}),
        bind-result    [17]    OPERATION.&ResultType ({operation}),
        bind-error     [18]    OPERATION.&Errors.&ParameterType ({operation})
}

Unbind {OPERATION:operation} ::= CHOICE
{
        unbind-invoke    [19]    OPERATION.&ArgumentType({operation}),
        unbind-result    [20]    OPERATION.&ResultType ({operation}),
        unbind-error     [21]    OPERATION.&Errors.&ParameterType ({operation})
}

END -- fin des définitions génériques des unités de données de protocole ROS
```

```
Remote-Operations-Useful-Definitions {joint-iso-itu-t remote-operations(4) useful-definitions(7) version1(0)}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
-- exporter tout
IMPORTS OPERATION, ERROR, OPERATION-PACKAGE, Code FROM Remote-Operations-Information-Objects
{joint-iso-itu-t remote-operations(4) informationObjects(5) version1(0)}
InvokeId, ROS{}, FROM Remote-Operations-Generic-ROS-PDUs {joint-iso-itu-t remote-operations(4)
generic-ROS-PDUs(6) version1(0)};

emptyBind OPERATION ::= {ERRORS {refuse} SYNCHRONOUS TRUE}

emptyUnbind OPERATION ::= { SYNCHRONOUS TRUE }

refuse ERROR ::= {CODE  local:-1}

no-op OPERATION ::=
{
        ALWAYS RESPONDS FALSE
        CODE  local:-1
}

Forward {OPERATION:OperationSet} OPERATION ::=
{
        OperationSet |
        OperationSet.&Linked.&Linked |
        OperationSet.&Linked.&Linked.&Linked.&Linked
}

Reverse  {OPERATION:OperationSet} OPERATION ::=
        {Forward{{OperationSet.&Linked}}}

ConsumerPerforms {OPERATION-PACKAGE:package} OPERATION ::=
{
        Forward{{package.&Consumer}} |
        Forward{{package.&Both}} |
        Reverse{{package.&Supplier}} |
        Reverse{{package.&Both}}
}

SupplierPerforms {OPERATION-PACKAGE:package} OPERATION ::=
{
        Forward{{package.&Supplier}} |
        Forward{{package.&Both}} |
        Reverse{{package.&Consumer}} |
        Reverse{{package.&Both}}
}

AllOperations {OPERATION-PACKAGE:package} OPERATION ::=
{
        ConsumerPerforms {package} |
        SupplierPerforms {package}
}
-- suite page suivante
```

```
recode {OPERATION:operation, Code:code} OPERATION ::=
{
        ARGUMENT                operation.&ArgumentType
                OPTIONAL                operation.&argumentTypeOptional
        RETURN RESULT           operation.&returnResult
        RESULT                  operation.&ResultType
                OPTIONAL                operation.&resultTypeOptional
        ERRORS                  {operation.&Errors}
        ALWAYS RESPONDS         operation.&alwaysReturns
        LINKED                  {operation.&Linked}
        SYNCHRONOUS             operation.&synchronous
        INVOKE PRIORITY         {operation.&InvokePriority}
        RESULT-PRIORITY         {operation.&ResultPriority}
        CODE                    code
}

switch {OPERATION-PACKAGE:package, OBJECT IDENTIFIER:id} OPERATION-PACKAGE ::=
{
        OPERATIONS              {package.&Both}
        SUPPLIER INVOKES        {package.&Supplier}
        CONSUMER INVOKES        {package.&Consumer}
        ID                      id
}

combine {OPERATION-PACKAGE:ConsumerConsumes, OPERATION-PACKAGE:ConsumerSupplies,
OPERATION-PACKAGE:base} OPERATION-PACKAGE ::=
{
        OPERATIONS              {ConsumerConsumes.&Both | ConsumerSupplies.&Both}
        SUPPLIER INVOKES        {ConsumerConsumes.&Supplier    | ConsumerSupplies.&Consumer}
        CONSUMER INVOKES        {ConsumerConsumes.&Consumer | ConsumerSupplies.&Supplier}
        ID                      base.&id
}

ROS-SingleAS {InvokeId:InvokeIdSet, OPERATION-PACKAGE:package} ::=
        ROS {{InvokeIdSet}, {AllOperations{package}}, {AllOperations{package}}}

ROS-ConsumerAS {InvokeId:InvokeIdSet, OPERATION-PACKAGE:package} ::=
        ROS {{InvokeIdSet}, {ConsumerPerforms{package}}, {SupplierPerforms{package}}}

ROS-SupplierAS {InvokeId:InvokeIdSet, OPERATION-PACKAGE:package} ::=
        ROS {{InvokeIdSet}, {SupplierPerforms{package}}, {ConsumerPerforms{package}}}

END -- fin des définitions utiles.
```

Annexe B

Directives pour l'utilisation de la notation

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

La présente annexe contient, à l'intention des concepteurs des protocoles d'application, des exemples et des directives relatifs à l'utilisation des classes d'objets informationnels correspondant aux concepts de base des services d'opérations distantes ROS, et à l'application de l'ensemble des définitions paramétrées utiles.

B.1 Exemples d'opérations et d'erreurs

Ce paragraphe fournit quelques exemples d'objets informationnels appartenant aux classes **OPERATION** et **ERROR**.

```
operationExample1 OPERATION ::=
    {
    ARGUMENT                ArgumentType1
    RESULT                  ResultType1
    ERRORS                  {errorExample1 | errorExample2}
    LINKED                  {operationExample2}
    CODE                    local:1
    }

operationExample2 OPERATION ::=
    {
    ARGUMENT                ArgumentType2
    RESULT                  ResultType2 OPTIONAL TRUE
    LINKED                  {operationExample4}
    ALWAYS RESPONDS         FALSE
    CODE                    local:2
    }

operationExample3 OPERATION ::=
    {
    ARGUMENT                ArgumentType3
    ERRORS                  {errorExample3}
    SYNCHRONOUS             TRUE
    CODE                    local:3
    }

operationExample4 OPERATION ::=
    {
    ARGUMENT                ArgumentType4
    RETURN RESULT           FALSE
    ALWAYS RESPONDS         FALSE
    CODE                    local:4
    }
```

L'opération distante asynchrone **operationExample1**, qui possède un argument du type **ArgumentType1**, renvoie toujours une réponse (ce qui découle implicitement de l'absence du mot clé **ALWAYS RESPONDS**), qui, en cas de succès d'exécution de l'opération, est une valeur de résultat du type **ResultType1**, ou, en cas d'échec de l'opération, est l'une des deux erreurs **errorExample1** ou **errorExample2** selon les circonstances de l'échec. L'opération **operationExample2** est liée à la présente opération, ce qui signifie qu'elle peut être invoquée en réponse à la présente opération à tout moment avant l'achèvement de cette dernière. L'opération **operationExample1** est identifiée par son code, l'entier 1.

L'opération distante asynchrone **operationExample2** possède un argument du type **ArgumentType2**. Il s'agit d'une opération soit qui n'échoue jamais, soit dont l'éventuel échec n'est pas notifié. Lorsque l'exécutant notifie le succès d'exécution de cette opération, il peut optionnellement omettre la valeur résultat qui est de type **ResultType2**. L'opération **operationExample4** est liée à la présente opération, ce qui signifie qu'elle peut être invoquée en réponse à la présente opération à tout moment avant l'achèvement de cette dernière. L'opération **operationExample2** est identifiée par l'entier 2.

L'opération distante synchrone **operationExample3**, identifiée par l'entier 3 et possédant un argument du type **ArgumentType3**, renvoie toujours un rapport d'exécution. Si l'opération a été exécutée avec succès, une indication est renvoyée en ce sens, mais aucune valeur de résultat n'est retournée. Si l'opération échoue, l'exécutant renvoie l'erreur **errorExample3**.

NOTE – Comme il s'agit d'une opération synchrone, le concepteur d'application doit s'assurer que cette fonction envoie toujours un rapport en retour, surtout s'il prévoit d'invoquer d'autres opérations *synchrones*.

L'opération asynchrone **operationExample4**, identifiée par l'entier 4 et possédant un argument du type **ArgumentType4**, ne renvoie pas de rapport d'exécution.

Les entités suivantes sont des instances de la classe (d'objets informationnels) **ERROR** utilisées pour rapporter l'échec d'exécution (de certaines) des opérations définies ci-dessus:

```
errorExample1 ERROR ::=
   {
   PARAMETER        ParameterType1
   CODE             local:1
   }

errorExample2 ERROR ::=
   {
   PARAMETER        ParameterType2 OPTIONAL TRUE
   CODE             local:2

errorExample3 ERROR
   {
   CODE             local:3
   }
```

L'erreur **errorExample1**, identifiée par l'entier 1, peut servir à notifier l'échec d'exécution des opérations **operationExample1** et **operationExample2**. Cette erreur comporte un paramètre de diagnostic d'erreur dont la valeur est du type **ParameterType1**.

L'erreur **errorExample2**, qui peut servir à notifier l'échec d'exécution de l'opération **operationExample1**, comporte un paramètre de diagnostic d'erreur dont la valeur est du type **ParameterType2**. Cette valeur peut être omise au choix de l'exécutant. Elle est identifiée par l'entier 2.

L'erreur **errorExample3**, identifiée par l'entier 3, sert à notifier l'échec d'exécution de l'opération **operationExample3**. Cette erreur ne comporte aucun paramètre (c'est-à-dire aucun diagnostic d'erreur).

B.2 Exemples de lots d'opérations et de l'utilisation de l'opérateur switch

Les opérations et les erreurs du B.1 peuvent être regroupées de la manière suivante en un même lot d'opérations **package1**:

```
package1 OPERATION-PACKAGE ::=
   {
   CONSUMER INVOKES     {operationExample1 | operationExample3}
   SUPPLIER INVOKES     {operationExample2}
   ID                   objectIdentifierOfPackage1
   }
```

L'un des deux objets ROS en interaction, arbitrairement désigné comme le «client» CONSUMER, peut invoquer les opérations **operationExample1** et **operationExample3**. L'autre objet ROS, désigné comme le «serveur» SUPPLIER, ne peut invoquer que l'opération **operationExample2**. Cette combinaison particulière de fonctions est globalement identifiée par la valeur **objectIdentifierOfPackage1**.

NOTE – Les termes «client» et «serveur», qui distinguent les deux objets ROS en interaction, doivent être définis par référence à des considérations extérieures aux opérations distantes. Le cas le plus simple est de les définir par référence à leurs rôles dans la formation du contrat d'association (voir B.5).

Une autre combinaison de la même collection d'opérations, définie comme suit, forme le lot **package2**:

```
package2 OPERATION-PACKAGE ::=
   {
   BOTH                 {operationExample3}
   CONSUMER INVOKES     {operationExample1}
   ID                   objectIdentifierOfPackage2
   }
```

Dans le lot **package2**, chacun des deux objets ROS peut invoquer l'opération **operationExample3**, le client peut invoquer l'opération **operationExample1**, et aucun des deux ne peut invoquer l'opération **operationExample2**.

Un troisième lot d'opérations, **package3**, peut être dérivé du premier au moyen de l'opérateur **switch{}**:

```
package3 OPERATION-PACKAGE ::= switch{OPERATION:package1, objectIdentifierOfPackage3}
```

L'utilisation de l'opérateur **switch{}** intervertit les opérations qui peuvent être invoquées par le «client» et par le «serveur» dans le lot **package1**, et affecte une nouvelle valeur d'identificateur d'objet à cette combinaison. Le développement complet de **package3** serait le suivant:

```
package3 OPERATION-PACKAGE ::=
   {
   CONSUMER INVOKES      {operationExample2}
   SUPPLIER INVOKES      {operationExample1 | operationExample3}
   ID                    objectIdentifierOfPackage3
   }
```

B.3 Exemples d'opérations de rattachement et de détachement

Une association peut être établie dynamiquement par l'invocation d'une opération de rattachement, par exemple l'opération suivante:

```
bindExample1 OPERATION ::=
   {
   ARGUMENT              BindArgumentType1
   RESULT                BindResultType1
   ERRORS                {bindError1}
   SYNCHRONOUS           TRUE
   }

bindError1 ERROR ::=
   {
   PARAMETER    BindErrorType1  OPTIONAL  TRUE
   }
```

L'opération synchrone **bindExample1** sert à établir une association entre deux objets ROS. L'invocation de l'opération comporte un argument de type **BindArgumentType1**, et l'établissement de l'association avec succès entraîne le retour d'une valeur de résultat du type **BindResultType1**. L'échec de l'établissement de l'association entraîne le renvoi d'une erreur **bindError1**, comportant optionnellement le paramètre diagnostic **BindErrorType1**.

La libération de l'association est réalisée par l'invocation d'une autre opération synchrone **unBindExample1** définie comme suit:

```
unBindExample1 OPERATION ::=
   {
   ARGUMENT              UnBindArgumentType1
   RESULT                UnBindResultType1  OPTIONAL  TRUE
   ERRORS                {unBindError1}
   SYNCHRONOUS           TRUE
   }

unBindError1 ERROR ::=
   {
   PARAMETER    UnBindErrorType1  OPTIONAL  TRUE
   }
```

B.4 Exemples de lots de connexion

Les exemples bindExample1 et **unBindExample1** permettent de définir un lot de connexion **connectionPackage1**, pouvant servir à établir et libérer dynamiquement une association entre deux objets ROS:

```
connectionPackage1 CONNECTION-PACKAGE ::=
    {
    BIND                        bindExample1
    UNBIND                      unBindExample1
    RESPONDER UNBIND            TRUE
    FAILURE TO UNBIND           TRUE
    ID                          objectIdentifierOfConnectionPackage1
    }
```

Le lot **connectionPackage1**, globalement identifié par la valeur **objectIdentifierOfConnectionPackage1** lorsque l'association entre les deux objets ROS est annoncée ou dynamiquement établie, utilise les opérations **bindExample1** et **unBindExample1** pour respectivement établir et libérer l'association. Il permet au répondeur de l'association de la libérer, et autorise le maintien de l'association après échec de l'opération **unBindExample1**.

On peut également citer le lot de connexion très simple suivant qui utilise par défaut les opérations **emptyBind** et **emptyUnbind** (voir 10.2 et 10.3) pour respectivement établir et libérer une association:

```
simpleConnectionPackage CONNECTION-PACKAGE ::=
    {
    ID      objectIdentifierOfSimpleConnectionPackage
    }
```

Dans ce lot de connexion, seul l'initiateur de l'association peut invoquer l'opération de rattachement, et l'association est libérée même en cas d'échec de l'opération de détachement.

B.5 Exemples de contrat d'association

Le contrat d'association **contract1** défini par:

```
contract1 CONTRACT ::=
    {
    CONNECTION                  connectionPackage1
    INITIATOR CONSUMER OF       {package1}
    ID                          objectIdentifierOfContract1
    }
```

indique que le lot de connexion **connectionPackage1** est utilisé pour établir et libérer l'association, et que l'objet ROS initiateur de l'association tient le rôle de «client» dans le lot d'opérations **package1**. Ce contrat est identifié par la valeur **objectIdentifierOfContract1**.

B.6 Exemples d'objets ROS

Soit l'objet ROS **object1** défini comme suit:

```
object1 ROS-OBJECT-CLASS ::=
    {
    INITIATES       {contract1}
    ID              objectIdentifierofROSObject1
    }
```

object1, qui est identifié par la valeur **objectIdentifierOfROSObject1**, appartient à un ensemble d'objets pouvant interagir avec d'autres objets ROS en lançant le contrat d'association **contract1**.

De même, **object2**, défini par:

```
object2 ROS-OBJECT-CLASS ::=
    {
    RESPONDS        {contract1}
    ID              objectIdentifierOf ROSObject2
    }
```

appartient à un ensemble d'objets pouvant interagir avec d'autres objets ROS en répondant à des interactions offrant le contrat d'association **contract1**.

B.7 Exemple d'utilisation des opérateurs Forward{} et Reverse{}

L'ensemble d'objets informationnels **ConsumerInvokes**, dérivé comme suit de l'ensemble **package1.&Supplier** défini au B.2:

```
ConsumerInvokes OPERATION ::= {package1.&Supplier}
```

produit l'ensemble **{operationExample1 | operationExample2}**, qui énumère les opérations pouvant être invoquées par le «client». De même:

```
SupplierLinkedInvokes OPERATION ::= {ConsumerInvokes.&Linked}
```

produit l'ensemble **{operationExample2}**, qui est l'ensemble des opérations pouvant être invoquées dans le sens «inverse», c'est-à-dire par le «serveur», alors que l'expression:

```
ConsumerLinkedInvokes OPERATION ::= {SupplierLinkedInvokes.&Linked}
```

produit l'ensemble **{operationExample4}**, qui est l'ensemble des opérations indirectement liées à l'ensemble formé par **package1.&Consumer** pouvant être invoquées par le «client».

L'ensemble d'opérations **Forward{OPERATION:ConsumerInvokes}** est l'ensemble constitué des opérations de l'ensemble d'origine **ConsumerInvokes** plus toutes les opérations de même «directionnalité» qui leur sont indirectement liées, c'est-à-dire dans notre exemple, les opérations pouvant être invoquées par le «client». Nous avons donc:

```
Forward {OPERATION:ConsumerInvokes} OPERATION ::=
        {operationExample1 | operationExample3 | operationExample4}
```

Par ailleurs, l'ensemble d'opérations **Reverse{OPERATION:ConsumerInvokes}** est l'ensemble des opérations appartenant à l'ensemble **package1.&Consumer.&Linked**, et qui peuvent être invoquées par le «serveur» en réponse aux opérations de l'ensemble **ConsumerInvokes**, plus toutes les opérations de même «directionnalité» qui leur sont indirectement liées, c'est-à-dire dans notre exemple, les opérations pouvant être invoquées par le «serveur». Nous avons donc:

```
Reverse{OPERATION:ConsumerInvokes} OPERATION ::=
                                Forward{OPERATION:SupplierLinkedInvokes}
```

ce qui, une fois explicité, donne:

```
Reverse{OPERATION:ConsumerInvokes} OPERATION ::= {operationExample2}
```

Par ailleurs, pour l'ensemble d'opérations **SupplierInvokes** obtenu de la manière suivante:

```
SupplierInvokes OPERATION ::= {package1.&Consumer}
```

nous obtenons **{operationExample2}**.

Nous avons donc:

```
Forward{OPERATION:SupplierInvokes} OPERATION ::= {operationExample2}

Reverse{OPERATION:SupplierInvokes} OPERATION ::= {operationExample4}
```

B.8 Exemple d'utilisation des opérateurs ConsumerPerforms{}, SupplierPerforms{} et AllOperations{}

A partir des exemples du B.7 et du lot d'opérations **package1**, le «client» peut effectuer toutes les opérations de l'ensemble:

```
ConsumerPerforms{OPERATION-PACKAGE:package1} OPERATION ::=
        {Forward{OPERATION:ConsumerInvokes} |
        {Reverse{OPERATION:SupplierInvokes}}
```

ce qui produit l'ensemble d'opérations **{operationExample1 | operationExample3 | operationExample4}**.

De même, et toujours dans le lot **package1**, le «serveur» peut effectuer les opérations de l'ensemble:

```
SupplierPerforms{OPERATION-PACKAGE:package1} OPERATION ::=
{{Forward{OPERATION:SupplierInvokes} |
        {Reverse{OPERATION:ConsumerInvokes}}
```

qui est l'ensemble **{operationExample2}**.

L'intérêt des opérateurs **ConsumerPerforms{}** et **SupplierPerforms{}** est de permettre au concepteur d'une application, dans une situation complexe faisant intervenir de nombreux liens imbriqués entre les opérations d'un lot donné, de faire ressortir les opérations de même «directionnalité», c'est-à-dire celles qui peuvent être invoquées par le client et celles qui peuvent être invoquées par le serveur, indépendamment de la nature de ces liens.

L'ensemble **AllOperations{OPERATION-PACKAGE:package1}** énumère toutes les opérations intervenant implicitement (c'est-à-dire par le biais de liens) ou explicitement dans le lot **package1** et qui peuvent être invoquées par l'un ou l'autre des objets en communication dans une instance quelconque d'utilisation de ce lot d'opérations, à savoir:

```
AllOperations{OPERATION-PACKAGE:package1} OPERATION ::=
        {ConsumerPerforms{package1} |
                                {SupplierPerforms{package1}}
```

ce qui produit l'ensemble d'opérations **{operationExample1 | operationExample2 | operationExample3 | operationExample4}**.

Annexe C

Migration des macro-instructions ROS

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

Les versions précédentes des systèmes ROS comportaient des macro-instructions ASN.1 destinées aux concepteurs d'applications ROS pour leur permettre de spécifier leurs opérations, erreurs, opérations de rattachement, éléments de service d'application, etc. De plus, dans un domaine étroitement lié aux systèmes ROS, la Recommandation X.407 du CCITT comportait d'autres macro-instructions que les concepteurs pouvaient utiliser pour la spécification en termes d'objets des applications réparties. Les macro-instructions sont en train d'être écartées de l'ASN.1, et, dans la présente Spécification, la notation ROS utilise à la place la notation de «remplacement des macro-instructions» de l'ASN.1, y compris les classes d'objets informationnels avec leurs paramétrages.

C.1 Introduction

Toutefois, il existe un certain nombre de spécifications en vigueur définies par des macro-instructions. La présente annexe indique par conséquent la manière de transformer des notations de macro-instructions en notation de remplacement. Très souvent, la notation de remplacement est plus exhaustive que la macro-instruction qu'elle remplace, mais cet aspect des choses n'est pas mis en relief dans la présente annexe, qui ne traite que de la compréhension de l'utilisation des macro-instructions existantes. Les macro-instructions ne devront pas être utilisées pour les nouvelles spécifications.

La présente annexe consacre un paragraphe à chaque macro-instruction existante. Ces paragraphes décrivent l'objet de la macro-instruction, en donnent un exemple d'utilisation avec l'équivalent en notation nouvelle, puis expliquent tout point resté obscur dans l'exemple. Dans les exemples donnés, le passage d'une notation à l'autre doit s'effectuer en supprimant les expressions **doublement soulignées** et en insérant les expressions en ***italique gras***.

La «définition en deux étapes», une des approches d'utilisation des macro-instructions, a été largement employée dans les spécifications existantes. Dans cette approche, les «identificateurs» des différentes sortes d'objets informationnels (les opérations ou les codes d'erreur par exemple) sont affectés dans la deuxième étape. Toutefois, l'interprétation de telles spécifications par le lecteur nécessite parfois de prendre en compte des facteurs normalement non pertinents, comme la similitude d'écriture de références ASN.1 distinctes ou la proximité des définitions ASN.1. Ceci n'est pas acceptable dans la notation de remplacement. La présente annexe ne décrit par conséquent que la transformation de la deuxième ou unique étape de la définition utilisant les macro-instructions.

C.2 Macro-instruction OPERATION

La macro-instruction **OPERATION** servait à spécifier des opérations (à l'exception des opérations de rattachement et de détachement). Les modifications à apporter pour passer de la définition d'une instance appelée get de la macro-instruction **OPERATION** à un élément de la classe d'objets informationnels **OPERATION** sont indiquées ci-dessous. Dans la notation en macro-instructions à gauche, les symboles doublement soulignés sont supprimés et les symboles en italique gras sont insérés pour passer à la nouvelle notation (reprise au propre à droite).

```
get OPERATION ::=
{       ARGUMENT    Descriptor
        RESULT      Information
        ERRORS {unknown , / noAccess}
        LINKED  {getPassword}
        ::= CODE  local:73
}
```

⇒

```
get OPERATION ::=
{
        ARGUMENT    Descriptor
        RESULT      Information
        ERRORS      {unknown | noAccess}
        LINKED      {getPassword}
        CODE        local:73
}
```

NOTES

1 L'omission de la déclaration **RESULT** dans la macro-notation équivaut à écrire **RETURN RESULT FALSE** en notation nouvelle.

2 La présence de la déclaration **RESULT** sans type de données dans la macro-notation équivaut à omettre la déclaration **RESULT** en notation nouvelle (la déclaration **RETURN RESULT TRUE** est la valeur par défaut et peut donc être omise).

3 Il découle de la supposition faite plus haut concernant la définition en deux étapes que les noms des opérations liées et erreurs individuelles commencent par une minuscule.

4 Les champs d'opération suivants, qui sont autorisés par la définition de la classe d'objets informationnels, ne pouvaient pas être spécifiés par la macro-notation, mais si nécessaire, pouvaient l'être textuellement: **&synchronous**, **&InvokePriority**, **&ResultPriority**.

5 Grâce aux champs **&argumentTypeOptional** et **&resultTypeOptional**, il est possible de déclarer dans la nouvelle notation si, respectivement, la valeur de l'argument ou du résultat peut être omise sur option de l'utilisateur.

C.3 Macro-instruction ERROR

La macro-instruction **ERROR** servait à spécifier des erreurs (à l'exception des erreurs liées aux opérations de rattachement et de détachement).

```
unknown ERROR ::=
{       PARAMETER Descriptor
        ::= CODE local:14
}
```

⇒

```
unknown ERROR ::=
{
        PARAMETER       Descriptor
        CODE            local:14
}
```

NOTES

1 Le champ **&ErrorPriority** ne pouvait pas être spécifié par la macro-notation, mais si nécessaire, pouvait l'être textuellement.

2 Grâce au champ **&parameterTypeOptional**, il est possible de déclarer dans la nouvelle notation si la valeur du paramètre, lorsqu'il y en a un de défini pour accompagner un rapport d'erreur, peut être omise sur option de l'utilisateur.

C.4 Macro-instruction Bind

La macro-instruction **BIND** servait à spécifier des opérations de rattachement.

```
Hallohallo ::= BIND OPERATION::=
{       ARGUMENT        HowAreYou
        RESULT          Fine-AndYou
        BIND-ERRORS { GgoAway}
}
```

⇒

```
hallo OPERATION::=
{
        ARGUMENT        HowAreYou
        RESULT          Fine-AndYou
        ERRORS          {goAway}
}
```

NOTE – L'absence du mot clé **CODE** dans la nouvelle notation indique qu'il s'agit d'une opération spéciale qui ne peut être invoquée au moyen de l'unité de données de protocole **Invoke{}**.

C.5 Macro-instruction Unbind

La macro-instruction **UNBIND** servait à spécifier des opérations de détachement.

```
Byebye ::= UNBIND OPERATION ::=
{       ARGUMENT        SeeYouSoon
        RESULT          LetsDoLunch
        BIND-ERRORS     { DdontGo}
}
```

⇒

```
bye  OPERATION ::=
{
        ARGUMENT        SeeYouSoon
        RESULT          LetsDoLunch
        ERRORS          {dontGo}
}
```

NOTE – L'absence du mot clé CODE dans la nouvelle notation indique qu'il s'agit d'une opération spéciale qui ne peut être invoquée à l'aide de l'unité de données de protocole **Invoke{}**.

Annexe D

Affectation de valeurs d'identificateurs d'objets

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

Les valeurs d'identificateurs d'objets suivantes sont affectées dans la présente Recommandation | Norme internationale:

Article	Valeur d'identificateur d'objet
Annexe A	**{joint-iso-itu-t remote-operations(4) informationObjects(5) version1(0)}**
	{joint-iso-itu-t remote-operations(4) generic-ROS-PDUs(6) version1(0)}
	{joint-iso-itu-t remote-operations(4) useful-definitions(7) version1(0)}