



UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

UIT-T

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

X.880

(07/94)

**REDES DE DATOS Y COMUNICACIÓN
ENTRE SISTEMAS ABIERTOS**

**APLICACIONES DE INTERCONEXIÓN DE
SISTEMAS ABIERTOS – OPERACIONES
A DISTANCIA**

**TECNOLOGÍA DE LA INFORMACIÓN –
OPERACIONES A DISTANCIA:
CONCEPTOS, MODELO Y NOTACIÓN**

Recomendación UIT-T X.880

(Anteriormente «Recomendación del CCITT»)

PREFACIO

La UIT (Unión Internacional de Telecomunicaciones) es el organismo especializado de las Naciones Unidas en el campo de las telecomunicaciones. El UIT-T (Sector de Normalización de las Telecomunicaciones de la UIT) es un órgano permanente de la UIT. En el UIT-T, que es la entidad que establece normas mundiales (Recomendaciones) sobre las telecomunicaciones, participan unos 179 países miembros, 84 empresas de explotación de telecomunicaciones, 145 organizaciones científicas e industriales y 38 organizaciones internacionales.

Las Recomendaciones las aprueban los Miembros del UIT-T de acuerdo con el procedimiento establecido en la Resolución N.º 1 de la CMNT (Helsinki, 1993). Adicionalmente, la Conferencia Mundial de Normalización de las Telecomunicaciones (CMNT), que se celebra cada cuatro años, aprueba las Recomendaciones que para ello se le sometan y establece el programa de estudios para el periodo siguiente.

En ciertos sectores de la tecnología de la información que corresponden a la esfera de competencia del UIT-T, se preparan las normas necesarias en colaboración con la ISO y la CEI. El texto de la Recomendación UIT-T X.880 se aprobó el 1 de julio de 1994. Su texto se publica también, en forma idéntica, como Norma Internacional ISO/CEI 13712-1.

NOTA

En esta Recomendación, la expresión «Administración» se utiliza para designar, en forma abreviada, tanto una administración de telecomunicaciones como una empresa de explotación reconocida de telecomunicaciones.

© UIT 1995

Es propiedad. Ninguna parte de esta publicación puede reproducirse o utilizarse, de ninguna forma o por ningún medio, sea éste electrónico o mecánico, de fotocopia o de microfilm, sin previa autorización escrita por parte de la UIT.

RECOMENDACIONES DE LA SERIE UIT-T X

**REDES DE DATOS
Y COMUNICACIÓN DE SISTEMAS ABIERTOS**

(Febrero 1994)

ORGANIZACIÓN DE LAS RECOMENDACIONES DE LA SERIE X

Dominio	Recomendaciones
REDES PÚBLICAS DE COMUNICACIÓN DE DATOS	
Servicios y facilidades	X.1-X.19
Interfaces	X.20-X.49
Transmisión, señalización y conmutación	X.50-X.89
Aspectos de redes	X.90-X.149
Mantenimiento	X.150-X.179
Disposiciones administrativas	X.180-X.199
INTERCONEXIÓN DE SISTEMAS ABIERTOS	
Modelo y notación	X.200-X.209
Definiciones de los servicios	X.210-X.219
Especificaciones de los protocolos en modo con conexión	X.220-X.229
Especificación de los protocolos en modo sin conexión	X.230-X.239
Formularios PICS	X.240-X.259
Identificación de protocolos	X.260-X.269
Protocolos de seguridad	X.270-X.279
Objetos gestionados de red	X.280-X.289
Pruebas de conformidad	X.290-X.299
INTERFUNCIONAMIENTO ENTRE REDES	
Consideraciones generales	X.300-X.349
Sistemas móviles de transmisión de datos	X.350-X.369
Gestión	X.370-X.399
SISTEMAS DE TRATAMIENTO DE MENSAJES	X.400-X.499
DIRECTORIO	X.500-X.599
GESTIÓN DE REDES OSI Y ASPECTOS DE SISTEMAS	
Gestión de redes	X.600-X.649
Denominación, direccionamiento y registro	X.650-X.679
Notación de sintaxis abstracta N.º 1 (ASN.1)	X.680-X.699
GESTIÓN OSI	X.700-X.799
SEGURIDAD	X.800-X.849
APLICACIONES OSI	
Cometimiento, concurrencia y recuperación	X.850-X.859
Procesamiento de transacción	X.860-X.879
Operaciones a distancia	X.880-X.899
TRATAMIENTO ABIERTO DISTRIBUIDO	X.900-X.999

ÍNDICE

	<i>Página</i>
Introducción.....	iv
1 Alcance.....	1
2 Referencias normativas	1
2.1 Recomendaciones Normas Internacionales idénticas.....	1
2.2 Recomendaciones Normas Internacionales de contenido técnico equivalente.....	2
2.3 Referencias adicionales.....	2
3 Definiciones	2
3.1 Definiciones del modelo de referencia de OSI	2
3.2 Definiciones de ASN.1	2
3.3 Definiciones del servicio de operaciones a distancia	3
4 Abreviaturas	3
5 Convenios.....	3
6 Modelo de operaciones a distancia.....	4
7 Realización de operaciones a distancia	5
8 Conceptos de operaciones a distancia	6
8.1 Introducción	6
8.2 Operación.....	7
8.3 Error	8
8.4 Lote de operaciones	9
8.5 Lote de conexiones	9
8.6 Contrato de asociación.....	10
8.7 Clase de objeto de operaciones a distancia	11
8.8 Código.....	12
8.9 Prioridad.....	12
9 Protocolo genérico de operaciones a distancia.....	12
9.1 Introducción	12
9.2 Operaciones a distancia.....	12
9.3 Invocación.....	13
9.4 Retorno de resultado	14
9.5 Retorno de error	15
9.6 Rechazo.....	16
9.7 Problema de rechazo	18
9.8 Id de invocación.....	18
9.9 No Id de invocación	18
9.10 Errores	18
9.11 Vinculación.....	19
9.12 Desvinculación.....	19
10 Definiciones útiles.....	19
10.1 Introducción	19
10.2 Vinculación vacía.....	19
10.3 Desvinculación vacía	20
10.4 Denegación	20
10.5 No-op	20
10.6 Directa	20
10.7 Inversa	20
10.8 Acciones de consumidor	21

	<i>Página</i>
10.9 Acciones de suministrador	21
10.10 Todas las operaciones	21
10.11 Recodificación	22
10.12 Conmutación	22
10.13 Combinación	22
10.14 Una sola sintaxis abstracta ROS	23
10.15 Sintaxis abstracta de consumidor ROS	23
10.16 Sintaxis abstracta de suministrador ROS	23
Anexo A – Módulos ASN.1	24
Anexo B – Directrices para el uso de la notación.....	31
B.1 Ejemplos de operaciones y sus errores	31
B.2 Ejemplos de lotes de operaciones y utilización de conmutación	32
B.3 Ejemplos de operaciones de vinculación y desvinculación	33
B.4 Ejemplos de lotes de conexiones	34
B.5 Ejemplo de un contrato de asociación.....	34
B.6 Ejemplos de objetos ROS	34
B.7 Ejemplo de utilización directa e inversa	35
B.8 Ejemplos de acciones de consumidor{ }, acciones de suministrador{ } y todas las operaciones{ }	36
Anexo C – Sustitución gradual de las macros ROS	37
C.1 Introducción	37
C.2 Operación.....	37
C.3 Error	38
C.4 Vinculación.....	38
C.5 Desvinculación.....	38
Anexo D – Asignación de valores de identificador de objeto	39

Resumen

En la presente Recomendación | Norma Internacional se utiliza la notación ASN.1 para definir clases de objeto de información correspondientes a los conceptos fundamentales del servicio de operaciones a distancia (ROS). Esto, a su vez, proporciona una notación que permitirá a los proyectistas de aplicaciones especificar instancias de dichas clases. Esta Recomendación | Norma Internacional ofrece también un conjunto de definiciones para especificar el protocolo genérico entre objetos que comunican mediante operaciones a distancia. También contiene varias definiciones de utilidad general para los proyectistas de aplicaciones basadas en el ROS.

Introducción

El servicio de operaciones a distancia es un paradigma para la comunicación interactiva entre objetos. Como tal, se puede utilizar en el diseño y especificación de aplicaciones distribuidas. La interacción básica correspondiente es la invocación de una operación por un objeto (el invocador), su realización por otro (el realizador), posiblemente seguida de un informe del resultado de la operación que se devuelve al invocador.

Los conceptos del servicio de operaciones a distancia son abstractos y se pueden realizar de muchas maneras. Por ejemplo, los objetos cuyas interacciones emplean conceptos del servicio de operaciones a distancia pueden estar separados por una interfaz de soporte lógico o por una red de OSI.

Esta Recomendación | Norma Internacional describe los conceptos y modelos del servicio de operaciones a distancia. Utiliza la notación de sintaxis abstracta uno (ASN.1) para especificar clases de objetos de información que corresponden a los conceptos fundamentales del servicio de operaciones a distancia, tales como operación y error. A su vez, éste proporciona una notación de modo que los diseñadores pueden especificar ejemplos particulares de estas clases, por ejemplo, operaciones y errores particulares.

La presente Recomendación | Norma Internacional proporciona un conjunto genérico de unidades de datos de protocolo, que se pueden utilizar para aplicar los conceptos del servicio de operaciones a distancia entre objetos distantes entre sí. Estas unidades de datos de protocolo se utilizan en la realización de OSI del servicio de operaciones a distancia, que se especifican en las Recomendaciones | Normas Internacionales asociadas con ésta.

La presente Recomendación | Norma Internacional proporciona también varias definiciones de utilidad general para los diseñadores de aplicaciones basadas en el servicio de operaciones a distancia.

El Anexo A forma parte integrante de la presente Recomendación | Norma Internacional.

Los Anexos B, C y D no forman parte integrante de la presente Recomendación | Norma Internacional.

NORMA INTERNACIONAL

RECOMENDACIÓN UIT-T

TECNOLOGÍA DE LA INFORMACIÓN – OPERACIONES A DISTANCIA: CONCEPTOS, MODELO Y NOTACIÓN

1 Alcance

La presente Recomendación | Norma Internacional especifica el servicio de operaciones a distancia (ROS, *remote operations service*) utilizando la notación de sintaxis abstracta uno (ASN.1) para definir clases de objetos de información que corresponden a los conceptos fundamentales del ROS. A su vez, éste proporciona la notación que permitirá a los diseñadores de aplicaciones especificar ejemplos particulares de estas clases.

La presente Recomendación | Norma Internacional proporciona también un conjunto de definiciones para especificar el protocolo genérico entre objetos que comunican utilizando conceptos del ROS. Estas definiciones se utilizan en las Recomendaciones | Normas Internacionales asociadas con ésta para proporcionar las unidades de datos de protocolo, las primitivas de servicio y las definiciones de contexto de aplicación utilizadas en la realización de OSI del ROS.

Se proporcionan también varias definiciones de utilidad general para los diseñadores de aplicaciones basadas en el ROS.

No hay requisitos de conformidad con esta Recomendación | Norma Internacional.

2 Referencias normativas

Las siguientes Recomendaciones UIT-T y Normas Internacionales contienen disposiciones que, mediante su referencia en este texto, constituyen disposiciones de esta Especificación. Al efectuar esta publicación, estaban en vigor las ediciones indicadas. Todas las Recomendaciones y Normas son objeto de revisiones, con lo que se preconiza que los participantes en acuerdos basados en esta Especificación investiguen la posibilidad de aplicar las ediciones más recientes de las Recomendaciones y Normas Internacionales citadas a continuación. Los miembros de la CEI y de la ISO mantienen registros de las Normas Internacionales actualmente vigentes. La Oficina de Normalización de las Telecomunicaciones de la UIT mantiene una lista de las Recomendaciones del UIT-T actualmente vigentes.

2.1 Recomendaciones | Normas Internacionales idénticas

- Recomendación X.680 del UIT-T (1994) | ISO/CEI 8824-1:1994, *Tecnología de la información – Notación de sintaxis abstracta uno (ASN.1): Especificación de la notación básica*.
- Recomendación X.681 del UIT-T (1994) | ISO/CEI 8824-2:1994, *Tecnología de la información – Notación de sintaxis abstracta uno (ASN.1): Especificación de objetos de información*.
- Recomendación X.682 del UIT-T (1994) | ISO/CEI 8824-3:1994, *Tecnología de la información – Notación de sintaxis abstracta uno (ASN.1): Especificación de constricciones*.
- Recomendación X.683 del UIT-T (1994) | ISO/CEI 8824-4:1994, *Tecnología de la información – Notación de sintaxis abstracta uno (ASN.1): Parametrización de especificaciones ASN.1*.
- Recomendación X.200 del UIT-T (1994) | ISO/CEI 7498-1:1994, *Tecnología de la información – Modelo de referencia básico: El modelo básico*.
- Recomendación X.881 del UIT-T (1994) | ISO/CEI 13712-2:1994, *Tecnología de la información – Operaciones a distancia: Realizaciones de OSI – Definición de servicio del elemento de servicio de operaciones a distancia*.
- Recomendación X.882 del UIT-T (1994) | ISO/CEI 13712-3:1994, *Tecnología de la información – Operaciones a distancia: Realizaciones de OSI – Especificación de protocolo del elemento de servicio de operaciones a distancia*.

2.2 Recomendaciones | Normas Internacionales de contenido técnico equivalente

- Recomendación X.219 del CCITT (1988), *Operaciones a distancia: Modelo, notación y definición de servicio*.

ISO/CEI 9072-1:1989, *Information processing systems – Text communication – Remote Operations – Part 1: Model, notation and service definition*.
- Recomendación X.229 del CCITT (1988), *Operaciones a distancia: Especificación de protocolo*.

ISO/CEI 9072-2:1989, *Information processing systems – Text communication – Remote Operations – Part 2: Protocol specification*.

2.3 Referencias adicionales

- Recomendación X.407 del CCITT (1988), *Sistemas de tratamiento de mensajes: Convenios de definición de servicio abstracto*.

3 Definiciones

3.1 Definiciones del modelo de referencia de OSI

La presente Recomendación | Norma Internacional utiliza los siguientes términos definidos en la Rec. X.200 del UIT-T | ISO/CEI 7498-1:

- a) sintaxis abstracta;
- b) unidad de datos de protocolo;
- c) calidad de servicio.

3.2 Definiciones de ASN.1

La presente Recomendación | Norma Internacional utiliza los siguientes términos definidos en la Rec. X.680 del UIT-T | ISO/CEI 8824-1:

- a) tipo (de datos);
- b) valor (de datos).

La presente Recomendación | Norma Internacional utiliza los siguientes términos definidos en la Rec. X.681 del UIT-T | ISO/CEI 8824-2:

- a) campo;
- b) objeto (de información);
- c) clase de objeto (de información);
- d) conjunto de objetos (de información).

La presente Recomendación | Norma Internacional utiliza los siguientes términos definidos en la Rec. X.682 del UIT-T | ISO/CEI 8824-3:

- a) constricción;
- b) valor de excepción.

La presente Recomendación | Norma Internacional utiliza el siguiente término definido en la Rec. X.683 del UIT-T | ISO/CEI 8824-4.

- parametrizado.

3.3 Definiciones del servicio de operaciones a distancia

La presente Recomendación | Norma Internacional define los siguientes términos:

- 3.3.1 argumento:** Un valor de datos que acompaña la invocación de una operación.
- 3.3.2 asociación:** Relación entre un par de objetos, que sirve como contexto para la invocación y realización de operaciones.
- 3.3.3 contrato de asociación:** Una especificación de los cometidos de un par de objetos comunicantes que pueden tener una asociación entre sí.
- 3.3.4 asimétrico:** Describe un lote de operaciones (o contrato de asociación), en el que los conjuntos de operaciones que las dos partes son capaces de realizar difieren.
- 3.3.5 lote de conexión:** Una especificación de los cometidos de un par de los objetos comunicantes en el establecimiento y liberación dinámicos de asociaciones entre ellos.
- 3.3.6 contrato:** Conjunto de requisitos impuestos a uno o más objetos que prescribe un comportamiento colectivo.
- 3.3.7 error:** Un informe de la realización infructuosa de una operación.
- 3.3.8 operación enlazada:** Operación invocada durante la realización de otra operación por el realizador (de la última) y destinada a ser realizada por el invocador (de la última).
- 3.3.9 objeto:** Un modelo (posiblemente una parte autónoma) de un sistema, caracterizado por su estado inicial y su comportamiento producido por interacciones externas en interfaces bien definidas.
- 3.3.10 operación:** Función que un objeto (el invocador) puede solicitar de otro (el realizador).
- 3.3.11 lote de operaciones:** Conjunto de operaciones conexas utilizadas para especificar los cometidos de un par de objetos comunicantes, cada operación puede ser invocada por uno o ambos objetos del par y realizada por el socio.
- 3.3.12 parámetro (de un error):** Un valor de datos que pueda acompañar al informe de un error.
- 3.3.13 resultado:** Un valor de datos que pueda acompañar al informe de la realización fructuosa de una operación.
- 3.3.14 objeto ROS:** Objeto cuyas interacciones con otros objetos se describen utilizando conceptos del ROS.
- 3.3.15 simétrico:** Describe un lote de operaciones (o contrato de asociación) en el cual ambas partes son capaces de realizar el mismo conjunto de operaciones.
- 3.3.16 síncrona:** Característica de una operación que, una vez invocada, su invocador no puede invocar otra operación síncrona (con el mismo realizador previsto) hasta que se haya informado el resultado.

4 Abreviaturas

A los efectos de esta Recomendación | Norma Internacional se utilizan las siguientes abreviaturas:

ASN.1	Notación de sintaxis abstracta uno (<i>abstract syntax notation one</i>)
PDU	Unidad de datos de protocolo (<i>protocol data unit</i>)
QOS	Calidad de servicio (<i>quality of service</i>)
RO (o ROS)	Operaciones a distancia (<i>remote operations</i>)

5 Convenios

La presente Recomendación | Norma Internacional emplea la ASN.1 para definir:

- Clases de objetos de información correspondientes a los conceptos ROS. Se proporciona también notación con la cual los diseñadores de aplicaciones ROS pueden especificar ejemplos particulares de estas clases.
- Objetos de información particulares de estas clases.
- Las PDU del protocolo ROS genérico.
- Tipos de datos necesarios en estas definiciones.

Muchas de estas definiciones están parametrizadas, de modo que sus usuarios deben suministrar parámetros reales para completarlas.

6 Modelo de operaciones a distancia

Operaciones a distancia (ROS) es un paradigma para comunicación interactiva entre objetos. Los objetos cuyas interacciones se describen y especifican utilizando ROS son **objetos ROS**. La interacción básica es la invocación de una operación por un objeto ROS (el invocador) y su realización por otro (el realizador).

La realización completa de la operación (fructuosa o infructuosamente) puede conducir a la devolución al invocador, de un informe de su resultado por el realizador. Esto se ilustra en la Figura 1.

Un informe de una operación completada fructuosamente es un **resultado**; un informe de una operación completada infructuosamente es un **error**.

Durante la realización de una operación, el realizador puede invocar **operaciones enlazadas**, destinadas a ser realizadas por el invocador de la operación original.

Para el interfuncionamiento correcto, ciertas propiedades de la operación deben ser conocidas por el invocador y el realizador. Estas propiedades comprenden:

- si han de devolverse informes, y en caso afirmativo, cuáles;
- los tipos de valores, si hubiere alguno, que han de transportarse con las invocaciones de la operación o devoluciones de ésta;
- las operaciones, si hubiera alguna, que pueden enlazarse con ésta;
- el valor de código que ha de utilizarse para distinguir esta operación de las otras que pudieran ser invocadas.

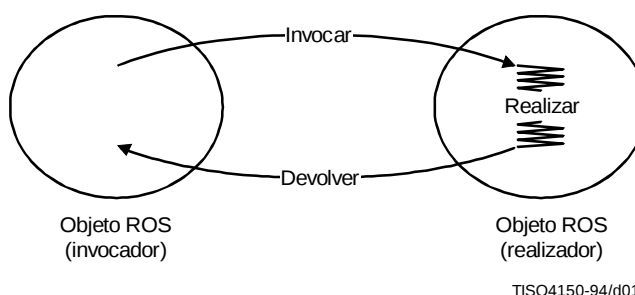


Figura 1 – Invocación, realización y devolución de resultado de una operación

Las capacidades de interfuncionamiento de (pares de) objetos ROS de alguna **clase de objeto ROS** se definen en términos de conjuntos de operaciones conexas denominadas **lotes de operaciones**. Un lote puede ser **simétrico**, en cuyo caso es definido por un solo conjunto de operaciones que cada objeto ROS en el par puede invocar (del otro). Alternativamente, un lote puede ser **asimétrico**, en cuyo caso es definido por dos conjuntos de operaciones, aquéllas que un objeto ROS del par puede invocar y aquéllas que el otro puede invocar. Para definir un lote asimétrico, los dos objetos ROS se denominan (de manera arbitraria) el **consumidor** y el **suministrador**, respectivamente.

NOTA 1 – Aunque en general estas etiquetas son arbitrarias, a menudo ocurrirá que se puede hacer una asignación intuitiva, dado que frecuentemente en un par de objetos uno suministrará un servicio que el otro consume.

Un par de objetos ROS debe tener una asociación entre ellos para servir como un contexto para la invocación y realización de operaciones. Cada asociación es regida por un **contrato de asociación**. Un contrato se especifica en términos del conjunto de lotes que (colectivamente) determinan las operaciones que pueden ser invocadas durante la asociación. Si la especificación del contrato incluye uno o más lotes asimétricos, el contrato es asimétrico. Para especificar un contrato de asociación asimétrico, los dos objetos ROS que establecen una asociación entre sí se denominan el **iniciador** y el **respondedor**.

Una asociación puede establecerse y suprimirse por medios «fuera de línea». Como otra posibilidad, una asociación puede ser establecida y liberada dinámicamente. Una opción, que se describe en la presente Recomendación | Norma Internacional, por la cual una asociación puede ser establecida y liberada dinámicamente, es mediante la invocación y realización de operaciones **de vinculación** y **desvinculación** especiales, respectivamente. El contrato para asociaciones de la segunda variedad incluye un **lote de conexión**, que comprende las operaciones de vinculación y desvinculación particulares que han de utilizarse.

NOTA 2 – El mecanismo para el establecimiento y liberación de asociaciones puede ser efectuado por otros medios descritos en otras Recomendaciones | Normas Internacionales.

Una asociación requiere una relación entre los objetos, cuya existencia corresponde al acuerdo mutuo de los objetos con los términos de algún contrato de asociación.

NOTA 3 – Esta Especificación no trata de los medios por los cuales estas relaciones son establecidas o terminadas.

En lo que antecede, los únicos objetos que, según se ve, participan en una operación han sido el invocador y el realizador. Sin embargo, en general el invocador y el realizador de una operación no están unidos directamente entre sí, sino conectados por algún medio por el cual se pueden transportar invocaciones y devoluciones. Esta visión ampliada se ilustra en la Figura 2.

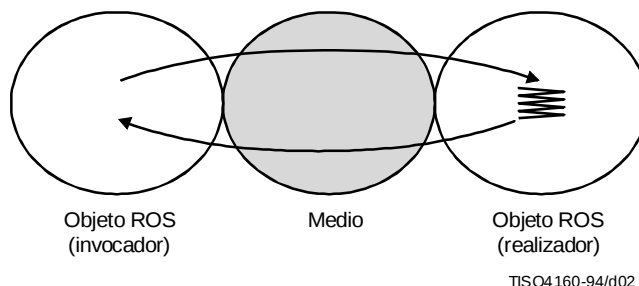


Figura 2 – Visión ampliada

El medio puede introducir retardo y la posibilidad de fallo e inexactitud en el transporte de las invocaciones y devoluciones de resultado y en el establecimiento, liberación y mantenimiento de asociaciones. Puede introducir también la posibilidad de que la seguridad de la asociación y sus operaciones sea amenazada. La extensión de éstos (junto con otros factores), se describen como la calidad de servicio.

Se puede considerar que los contratos de asociación tienen tres partes, y la tercera parte es el medio. La obligación del medio bajo el contrato es satisfacer los requisitos de calidad de servicio.

NOTA 4 – En el futuro, los requisitos máximos y mínimos aceptables de calidad de servicio pueden formar parte de la especificación de operaciones, lotes de operaciones y del propio contrato de asociación directamente. Diferentes aspectos de la calidad de servicio son aplicables a estos diferentes niveles de especificación.

7 Realización de operaciones a distancia

Una **realización** de ROS comprende la definición de un medio adecuado para transportar invocaciones y devoluciones de resultado entre objetos ROS. Este medio puede comprender, por ejemplo:

- a) una transferencia de mensaje o procedimiento que requiere una capacidad que permita al invocador y al realizador de una operación llevarla a cabo en módulos de soporte lógico desarrollados separadamente dentro de un solo sistema de computador;
- b) una capacidad de comunicaciones, que permita al invocador y al realizador de una operación llevarla a la práctica en sistemas de computador separados.

Una realización puede ser para fines generales, en cuyo caso se puede emplear para sustentar cualquier contrato de asociación. Otras son para fines especiales, y acomodan solamente algún contrato o contratos particulares.

La Figura 3 muestra un método para realizar ROS por medios de comunicación que es probable se utilicen ampliamente.

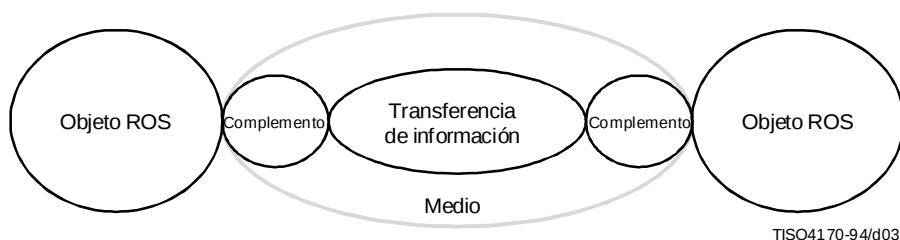


Figura 3 – Método para la realización de operaciones de ROS

En este método, el medio se compone de objetos de **complemento**, uno para cada objeto ROS, y un objeto de transferencia de información. El objeto de complemento asociado con cada objeto ROS parece desempeñar el cometido del objeto ROS socio. Sin embargo, no invoca ni efectúa realmente ninguna operación, sino que simplemente transforma invocaciones y devoluciones de resultado en PDU y viceversa, según proceda. Estas PDU son intercambiadas entre los complementos por medio del objeto de transferencia de información.

De este modo, para invocar una operación, el invocador invoca la operación del complemento asociado, que forma una PDU que describe la invocación. El complemento utiliza la capacidad de transferencia de información para transferir la PDU al otro complemento. El segundo complemento interpreta la PDU e invoca la operación apropiada del objeto ROS con el cual está asociado, el realizador. Cuando la operación se ha efectuado, el realizador transporta cualquier devolución de resultado a su complemento asociado, que forma una PDU que la describe. Después el complemento utiliza la capacidad de transferencia de información para transferir la PDU al otro complemento. El segundo complemento interpreta la PDU e informa la devolución al invocador.

En la cláusula 9 se define un conjunto de las PDU adecuadas.

En una realización ROS de esta clase se puede utilizar distintas capacidades de transferencia de información. De particular importancia son las capacidades de transferencia de información de OSI. En las Recomendaciones | Normas Internacionales asociadas, Rec. UIT-T X.881 | ISO/CEI 13712-2 y Rec. UIT-T X.882 | ISO/CEI 13712-3, se describen algunas de estas realizaciones.

8 Conceptos de operaciones a distancia

8.1 Introducción

8.1.1 En esta cláusula se definen las clases de objeto de información que corresponden a los conceptos básicos de ROS, especificando las características que tienen los objetos de estas clases. Se definen las siguientes clases de objeto de información:

- OPERACIÓN (que describe operaciones) (OPERATION)
- ERROR (que describe errores) (ERROR)
- LOTE DE OPERACIONES (que describe lotes de operaciones) (OPERATION-PACKAGE)
- LOTE DE CONEXIÓN (que describe lotes de conexión) (CONNECTION-PACKAGE)
- CONTRATO (que describe contratos de asociación) (CONTRACT)
- CLASE DE OBJETO ROS (que describe clases de objetos ROS) (ROS-OBJECT-CLASS)

8.1.2 Las clases de objeto de información se definen utilizando la ASN.1, que proporciona notación que está a disposición de los diseñadores de aplicaciones ROS para especificar ejemplos particulares de estas clases. Se insta a los diseñadores, aunque no están obligados, a utilizar este método de especificación. Si se emplea algún otro método, la especificación resultante incluirá o hará referencia a una descripción de cómo se puede obtener una utilización válida de la notación proporcionada.

NOTA – Varias especificaciones existentes emplean la anotación MACRO DE ASN.1 (definida en anteriores versiones de esta Recomendación | Norma Internacional: véase la Rec. X.219 del CCITT | ISO/CEI 9072-1) para especificar operaciones, errores y otras clases de objetos de información pertinentes a ROS. En el Anexo C se describe cómo la utilización de estas macros puede transformarse en la notación proporcionada. Estas macros no deben utilizarse para nuevas aplicaciones.

8.2 Operación

8.2.1 Una operación es una función que un objeto (el invocador) puede pedir a otro (el realizador). La clase de objeto de información OPERATION, a la cual pertenecen todas las operaciones, se especifica como sigue y los distintos campos se describen en 8.2.2 a 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 El campo **&ArgumentType** (tipo de argumento) especifica el tipo de datos del argumento de la operación. Si en alguna operación se omite el campo, la operación toma el valor de ningún argumento.

8.2.3 El campo **&argumentTypeOptional** (tipo de argumento facultativo) que puede estar presente solamente si el campo **&ArgumentType** está presente, especifica si el tipo de datos del argumento de la operación puede omitirse facultativamente. Si este campo está ausente o toma el valor FALSE (falso), el valor de **&ArgumentType** no puede ser omitido de la PDU Invoke{ } (invocación) (véase 9.3).

8.2.4 El campo **&returnResult** (devolución de resultado) especifica si se devuelve un resultado en el caso de que la operación se realice satisfactoriamente, y forma el valor TRUE (verdadero) si es así, y FALSE en los demás casos.

8.2.5 El campo **&ResultType** (tipo de resultado) especifica el tipo de datos del valor devuelto con el resultado de la operación. Si se omite, la operación devuelve el valor ningún resultado. Se omitirá si el campo **&returnResult** es FALSE.

8.2.6 El campo **&resultTypeOptional** (tipo de resultado facultativo), que puede estar presente solamente si está presente el campo **&ResultType**, especifica si el tipo de datos del valor devuelto como resultado de la ejecución de la operación puede omitirse facultativamente. Si este campo está ausente o toma el valor FALSE, el valor de **&ResultType** no puede ser omitido de la PDU **&ReturnResult** (véase 9.4).

8.2.7 El campo **&Errors** (errores) especifica un conjunto de errores, cualquiera de los cuales puede ser devuelto para informar una ejecución infructuosa de la operación. Si se omite este campo, la ejecución infructuosa de la operación no es posible o no se ha informado.

8.2.8 El campo **&alwaysReturns** (devuelve siempre) especifica si el resultado de la ejecución de la operación se devuelve siempre, y toma el valor TRUE si es así y FALSE en los demás casos. Si este campo se pone a TRUE, por lo menos debe estar presente uno de los campos **&returnResult** o **&Errors**.

8.2.9 El campo **&Linked** (enlazado) si está presente, especifica un conjunto de operaciones, cualquiera de las cuales puede ser invocada como operaciones enlazadas durante la ejecución de la operación. Si este campo se omite, no puede enlazarse ninguna operación con una invocación de ésta.

8.2.10 El campo `&synchronous` (síncrono) especifica si la operación es síncrona o no, y toma el valor `TRUE` si lo es, y `FALSE` en los demás casos. Si el campo `&returnResult` es `FALSE`, este campo tomará también el valor `FALSE`. Cuando el campo síncrono se pone a `TRUE`, ello implica que ninguna otra operación *síncrona* puede ser invocada por este lado hasta que se ha devuelto el resultado de esta operación.

NOTA – La combinación de los campos `&alwaysReturns` y `&synchronous` sustituye el anterior concepto de «clases de operaciones» definido en la Rec. X.219 del CCITT | ISO/CEI 9072-1.

8.2.11 El campo `&InvokePriority` (prioridad de invocación) especifica los niveles de prioridad permitidos (véase 8.9) en que esta operación puede ser invocada.

8.2.12 El campo `&ResultPriority` (prioridad de resultado) especifica los niveles de prioridad permitidos (véase 8.9) en que el resultado de esta operación puede ser devuelto. Si el campo `&returnResult` es `FALSE`, este campo se omitirá.

8.2.13 El campo `&operationCode` (código de operación) está presente, especifica el valor de código (véase 8.8) que se utiliza para identificar esta operación, por ejemplo, cuando ha de invocarse.

NOTA – Una operación que no tiene `&operationCode` no puede ser invocada utilizando la PDU `Invoke{}` (véase 9.3). Por consiguiente, en la práctica todas las operaciones deben tener `&operationCodeS` asignados, salvo cuando se prevé utilizarlas en algunas circunstancias especiales, por ejemplo, una operación de vinculación.

8.3 Error

8.3.1 Un error es un informe de la ejecución infructuosa de una operación. La clase de objeto de información `ERROR` a la cual pertenecen todos los errores, se especifica como sigue y los distintos campos se describen en 8.3.2 a 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 El campo `&ParameterType` (tipo de parámetro) especifica el tipo de dato del parámetro del error. Si en algún error se omite el campo, ese error toma el valor ningún parámetro.

8.3.3 El campo `¶meterTypeOptional` (tipo de parámetro facultativo), que puede estar presente solamente si está presente el campo `&ParameterType`, especifica si el tipo de datos del valor devuelto como el parámetro que califica el error puede estar presente facultativamente. Si este campo está ausente o toma el valor `FALSE`, el valor de `&ParameterType` no puede omitirse de la PDU `ReturnError { }` (Devolución de error) (véase 9.5).

8.3.4 El campo `&ErrorPriority` (prioridad de error) especifica los niveles de prioridad permitidos (véase 8.9) en que puede devolverse el error.

8.3.5 El campo `&errorCode` (código de error), si está presente, especifica el valor de código (véase 8.8) que se utiliza para identificar este error, por ejemplo, cuando se devuelve.

NOTA – Un error que no tiene `&errorCode` no puede devolverse utilizando la PDU `&ReturnError { }` definida a continuación. Por consiguiente, en la práctica todos los errores deben tener sus campos `&errorCode` asignados, salvo cuando se prevé utilizarlos en alguna circunstancia especial, por ejemplo, un error de vinculación.

8.4 Lote de operaciones

8.4.1 Un lote de operaciones es una especificación de los cometidos de un par de objetos comunicantes, en términos de las operaciones que pueden invocar entre sí. Cuando el lote es asimétrico, se emplean los términos «consumidor» y «suministrador» para los dos objetos participantes. La clase de objeto de información OPERATION-PACKAGE a la cual pertenecen todos estos lotes, se especifica como sigue, y los distintos campos se describen en 8.4.2 a 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 El campo &Both (ambos) especifica un conjunto de OPERATIONS que ambos objetos serán capaces de ejecutar. Se puede omitir este campo.

8.4.3 El campo &Consumer (consumidor) especifica un conjunto de OPERATIONS que uno de los objetos, conocidos como el consumidor, será capaz de ejecutar. Este campo se puede omitir, en cuyo caso el consumidor sólo será capaz de ejecutar las operaciones especificadas por el campo &Both.

8.4.4 El campo &Supplier (suministrador) especifica un conjunto de OPERATIONS que uno de los objetos, conocidos como el suministrador, será capaz de ejecutar. Este campo se puede omitir, en cuyo caso el suministrador sólo será capaz de realizar las operaciones especificadas por el campo &Both.

NOTA – Se proporciona un lote de operaciones parametrizado, switch { } (conmutación) (véase 10.12) para poder obtener un lote de operaciones conmutando los cometidos de algún otro.

8.4.5 El campo &id, si está presente, especifica el valor de IDENTIFICADOR DE OBJETO (OBJECT IDENTIFIER) que se utiliza para identificar este lote, por ejemplo, si se está anunciando o negociando.

NOTA – Un lote que no tiene &id no puede ser anunciado ni negociado.

8.4.6 Todas las operaciones incluidas (directa o indirectamente) por los campos &Both, &Consumer, y &Supplier tendrán valores distintos de &operationCode.

8.4.7 Todos los valores incluidos en el campo &Errors de todas las operaciones incluidas (véase 8.4.6) tendrán valores distintos de &errorCode.

8.5 Lote de conexiones

8.5.1 Un lote de conexiones es una especificación de las operaciones y calidad de servicio que participan en el establecimiento y liberación dinámicos de una asociación. El lote de conexiones se especificará solamente si se utilizan las operaciones vinculación y desvinculación para establecer y liberar dinámicamente una asociación, respectivamente.

La clase de objeto de información CONNECTION-PACKAGE a la cual pertenecen todos estos lotes de conexión, se especifica como sigue y los distintos campos se describen en 8.5.2 a 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 El campo &bind (vinculación) especifica una OPERATION que se ha de ejecutar como parte del establecimiento de una asociación. Esta operación debe tener sus campos &returnResult y &alwaysReturns puestos a TRUE y tener un solo error, presente en el campo &Errors. Si la asociación se establece satisfactoriamente, la operación vinculación devolverá un resultado. Si la asociación no se establece satisfactoriamente, la operación vinculación informará su error. Si el campo &bind no se incluye explícitamente, el lote de conexiones incluirá la operación emptyBind (vinculación vacía) (véase 10.2).

8.5.3 El campo &unbind (desvinculación) especifica una OPERATION que ha de ejecutarse como parte de la liberación de una asociación. Esta operación debe tener sus campos &returnResult y &alwaysReturns puestos a TRUE, y debe tener un solo error, definido por el campo &Errors. Si la asociación se libera satisfactoriamente, la operación desvinculación devolverá un resultado. Si la asociación no se libera satisfactoriamente, la operación desvinculación no devolverá un resultado e informará su error. Si el campo &unbind no está incluido explícitamente, el lote de conexiones incluirá la operación emptyUnbind (desvinculación vacía) (véase 10.3).

8.5.4 El campo &responderCanUnbind (respondedor puede desvincular) indica si el respondedor de la asociación (y el iniciador) puede invocar o no la operación &unbind.

8.5.5 El campo &unbindCanFail (desvinculación puede fracasar) indica si es posible o no que la asociación exista aún después que la operación &unbind ha señalado un error.

8.5.6 El campo &id, si está presente, identifica el valor de IDENTIFICADOR DE OBJETO que se utiliza para identificar este lote de conexión, por ejemplo, si está siendo anunciado o negociado.

NOTA – Un lote de conexiones que no tiene &id no puede ser anunciado ni negociado.

8.6 Contrato de asociación

8.6.1 Un contrato de asociación es una especificación de los cometidos de un par de objetos comunicantes que puedan establecer una asociación entre sí. La clase de objeto de información CONTRACT a la cual pertenecen todos estos contratos, se especifica como sigue y los distintos campos se describen en 8.6.2 a 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 presencia del campo **&connection** (conexión) indica que la asociación regida por este contrato de asociación es establecida y liberada dinámicamente por medio de las respectivas operaciones de vinculación y desvinculación especificadas como parte de un lote de conexiones, y el contenido del campo especifica el lote de conexiones participante.

8.6.3 El campo **&OperationsOf** (operaciones de), si está presente, especifica uno o más lotes de operaciones aplicables mientras existe la asociación, y cuáles son simétricos y en cuáles el iniciador de la asociación puede desempeñar los cometidos de consumidor y de suministrador.

8.6.4 El campo **&InitiatorConsumerOf** (iniciador consumidor de), si está presente, especifica uno o más lotes de operaciones aplicables mientras existe la asociación en cuáles se considerará que el iniciador de la asociación desempeña el cometido de consumidor.

NOTA – Se proporciona un lote de operaciones parametrizadas **switch{}** (véase 10.12) para poder obtener un lote de operaciones conmutando los cometidos de algún otro.

8.6.5 El campo **&InitiatorSupplierOf** (iniciador suministrador de), si está presente, especifica uno o más lotes de operaciones aplicables mientras existe la asociación y en cuáles se considerará que el iniciador de la asociación desempeña el cometido de suministrador.

8.6.6 El campo **&id**, si está presente, especifica el valor de IDENTIFICADOR DE OBJETO que se utiliza para identificar este contrato de asociación, por ejemplo, si está siendo anunciado o negociado.

NOTA – Un contrato de asociación que no tiene **&id** no puede ser anunciado ni negociado.

8.7 Clase de objeto de operaciones a distancia

8.7.1 Una clase de objeto ROS define las capacidades de un conjunto de objetos ROS que tienen en común su capacidad de interactuar con otros objetos ROS utilizando un conjunto determinado de contratos. La clase de objeto de información ROS-OBJECT-CLASS se especifica como sigue, y los distintos campos se describen en 8.7.2 a 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 El campo **&Is**, si está presente, especifica un conjunto de clases de objeto ROS que son superclases de la clase que se define. Los objetos ROS de la última clase son capaces de sustentar todos los contratos implicados por su pertenencia a cada una de las superclases especificadas, así como las presentes explícitamente en los campos definidos en 8.7.3 a 8.7.5 a continuación.

8.7.3 El campo **&InitiatesAndResponds** (inicia y responde) especifica un conjunto de CONTRACTS para los cuales los objetos ROS de la clase serán capaces de actuar como iniciador y respondedor. Este campo se puede omitir.

8.7.4 El campo **&Initiates** (inicia) especifica un conjunto de CONTRACTS para los cuales los objetos ROS de la clase serán capaces de ejecutar el cometido de iniciador. Este campo se puede omitir, en cuyo caso los objetos ROS sólo serán capaces de iniciar los contratos de asociación especificados por el campo **&InitiatesAndResponds**.

8.7.5 El campo **&Responds** (responde) especifica un conjunto de CONTRACTS para los cuales el objeto de la clase será capaz de ejecutar el cometido de respondedor. Este campo se puede omitir, en cuyo caso el objeto sólo será capaz de responder a los contratos de asociación especificados por el campo **&InitiatesAndResponds**.

8.7.6 El campo **&id** especifica el valor de IDENTIFICADOR DE OBJETO que se utiliza para identificar esta clase de objeto, por ejemplo, si se está anunciando o negociando.

8.8 Código

El tipo Code (código) proporciona los valores para los campos &operationsCode de operaciones y los campos (&errorCode) de errores. Se especifica como sigue:

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

8.9 Prioridad

El tipo Priority (prioridad) se especifica como sigue:

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

Este parámetro define la prioridad asignada a la transferencia de la invocación correspondiente (de una operación) o la devolución de su resultado con respecto a otras invocaciones (y sus devoluciones) que han de ser intercambiadas entre los dos objetos ROS.

Mientras más bajo es el valor, más alta es la prioridad.

9 Protocolo genérico de operaciones a distancia

9.1 Introducción

Esta subcláusula proporciona un conjunto de definiciones que se pueden utilizar para especificar protocolos que realizan los conceptos ROS. Las definiciones primarias son:

- a) un conjunto parametrizado de PDU para invocaciones y devoluciones de invocaciones (ROS{});
- b) una PDU parametrizada para la invocación y devolución de una operación de vinculación (Bind{});
- c) una PDU parametrizada para la invocación y devolución de una operación de desvinculación (Unbind{}).

Además, hay varias definiciones secundarias que dependen de éstas.

9.2 Operaciones a distancia

9.2.1 El tipo parametrizado ROS{} proporciona una base para la definición de una sintaxis abstracta que contiene las PDU para la invocación de operaciones, para la devolución de resultados y errores, y para el rechazo de las PDU inválidas. Se especifica como sigue:

```
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 { -- must conform to the above definition -- }
! RejectProblem : general-unrecognizedPDU)
```

NOTA – En la práctica, el uso de reglas de codificación que no se autodelimitan y autoidentifican puede no permitir la distinción entre las violaciones de restricción a nivel externo e interno.

9.2.2 Los parámetros ASN.1 que se deben suministrar para producir un tipo totalmente definido son los siguientes:

- a) InvokeIdSet (conjunto de id de invocación). – Este conjunto de valores de tipo InvokeId (id de invocación) define los valores disponibles para la identificación de invocaciones y, por tanto, para la correlación de informes ulteriores. Si en alguna realización la función de correlación se puede efectuar por otros medios, se puede fijar al conjunto de valores NoInvokeId (ningún id de invocación) (véase 9.9);

NOTAS

1 El designador de aplicación puede escoger entre dejar este parámetro como ENTERO sin restricción, especificar la gama exacta o propagarlo como uno de los parámetros de la sintaxis abstracta (véase 10.14, 10.15 y 10.16).

2 Una realización OSI no admite el valor noInvokeId (véase 9.9.1) para las PDU Invoke, ReturnResult y ReturnError.

- b) el parámetro Invokable (invocable) especifica un conjunto de objetos que describe aquellas operaciones que pueden ser invocadas;
- c) el parámetro Returnable (restituible) especifica un conjunto de objetos que describe aquellas operaciones para las cuales puede ser necesario generar respuestas.

NOTA – Este parámetro se proporciona para recalcar la asimetría de la sintaxis abstracta (véase 10.15 y 10.16). Puede derivarse del conjunto de objetos de información Invokable. Se incluirán en este parámetro las operaciones del conjunto de objetos de información Invokable cuyo campo &alwaysReturns no es FALSE.

9.2.3 Si el receptor de una PDU no la reconoce como una de las definidas, se genera una PDU Reject{} (rechazo) cuyo componente problem (problema) toma el valor de excepción: general-unrecognisedPDU (PDU no reconocida-general).

9.3 Invocación

9.3.1 El tipo parametrizado Invoke{} (invocación) proporciona una PDU para la invocación de operaciones. Se especifica como sigue:

```

Invoke {InvokeId:InvokeIdSet, OPERATION:Operations} ::= SEQUENCE
{
    invokeId      InvokeId      (InvokeIdSet)
                        (CONSTRAINED BY {-- must be unambiguous --}
                        ! RejectProblem      : invoke-duplicateInvocation),
    linkedId      CHOICE {
                        present [0]    IMPLICIT present < InvokeId,
                        absent  [1]    IMPLICIT NULL
                        }
                        (CONSTRAINED BY {-- must identify an outstanding operation --}
                        ! RejectProblem      : invoke-unrecognizedLinkId)
                        (CONSTRAINED BY {-- which has one or more linked operations --}
                        ! RejectProblem      : invoke-linkedResponseUnexpected)
                        OPTIONAL,
    opcode        OPERATION.&operationCode
                        ({Operations}
                        ! RejectProblem      : invoke-unrecognizedOperation)
    argument      OPERATION.&ArgumentType
                        ({Operations} {:@opcode}
                        ! RejectProblem      : invoke-mistypedArgument)
                        OPTIONAL
}
(CONSTRAINED BY {-- must conform to the above definition --}
! RejectProblem      : general-mistypedPDU)
(
    WITH COMPONENTS
    {...,
      linkedId ABSENT
    }
|
    WITH COMPONENTS
    {...,
      linkedId PRESENT,
      opcode
      (CONSTRAINED BY {-- must be in the &Linked field of the associated operation --}
      ! RejectProblem      : invoke-unexpectedLinkedOperation)
    }
)

```

NOTA – En la práctica, el uso de reglas que no se autodelimitan y autoidentifican puede impedir la distinción entre violaciones de construcción a nivel externo e interno.

9.3.2 Los parámetros ASN.1 que se deben suministrar para producir un tipo totalmente definido son los siguientes:

- a) InvokeIdSet – Este conjunto de valores del tipo InvokeId define los valores disponibles para la identificación de invocaciones y, por tanto, para la correlación de informes ulteriores [véase 9.2.2 a)];
- b) Operations – Las que pueden ser invocadas.

9.3.3 La PDU resultante tiene hasta cuatro componentes, como sigue:

- a) **invokeId** – Este componente identifica la invocación particular. Será uno de los permitidos por el parámetro **InvokeIdSet**. **InvokeId** no será uno que ya está en uso (la regla para determinar esto es una característica de la correspondencia específica); en los demás casos, se generará una PDU **Reject{}** (véase 9.6), cuyo componente **problem** toma el valor de excepción: **invoke-duplicateInvocation**.
- b) **linkedId** (id enlazado) – Este componente, si está presente, indica que esta invocación está enlazada a una invocación anterior en el sentido opuesto. Será el **InvokeId** de una invocación anterior para la cual no se ha informado resultado. Si no es así, se generará una PDU **Reject{}**, cuyo componente **problem** toma el valor de excepción: **invoke-unrecognisedLinkedId**. La invocación anterior será de una operación que permite operaciones enlazadas; si no, se generará una PDU **Reject{}**, cuyo componente **problem** tomará el valor de excepción: **invoke-linkedResponseUnexpected**.

NOTA – Para una realización OSI no se utiliza el valor **absent:NULL** para el **linkedID**.

- c) **opcode** (código de operaciones) – Este componente identificará, por medio de su **&operationCode** (código de operación), una de las **operations**; si no, se generará una PDU **Reject{}** cuyo componente **problem** toma el valor de excepción: **invoke-unrecognisedOperation**. Si el componente **linkedId** está presente, la anterior invocación será de una operación que admitió que se enlazase con ella esta operación particular; si no, se generará una PDU **Reject{}**, cuyo componente **problem** toma el valor de excepción: **invoke-unexpectedLinkedOperation**.
- d) **argument** (argumento) – Este componente estará presente y es del tipo que forma el campo **&ArgumentType** (tipo de argumento) de la operación identificada por el componente **opcode**, a menos que se omita el campo o el campo **&argumentTypeOptional** indique que el campo **&ArgumentType** puede omitirse facultativamente, en cuyo caso este componente se omitirá. Si esta condición no se cumple, se generará una PDU **Reject{}**, cuyo componente **problem** toma el valor de excepción: **invoke-mistypedArgument**.

9.3.4 Si el receptor de esta PDU encuentra que está mal tipificada, se genera una PDU **Reject{}** cuyo componente **problem** toma el valor de excepción: **general-mistypedPDU**.

9.4 Retorno de resultado

9.4.1 El tipo parametrizado **ReturnResult{}** proporciona una PDU para informar la ejecución satisfactoria de operaciones. Se especifica como sigue:

```
ReturnResult {OPERATION:Operations} ::= SEQUENCE
{
    invokeId      InvokeId
                  (CONSTRAINED BY {-- must be that for an outstanding operation --}
                  ! RejectProblem      : returnResult-unrecognizedInvocation)
                  (CONSTRAINED BY {-- which returns a result --}
                  ! RejectProblem      : returnResult-resultResponseUnexpected),
    result        SEQUENCE
    {
        opcode     OPERATION.&operationCode
                  ({Operations})(CONSTRAINED BY {-- identified by invokeId --}
                  ! RejectProblem      : returnResult-unrecognizedInvocation),
        result      OPERATION.&ResultType ({Operations} {@opcode}
                  ! RejectProblem      : returnResult-mistypedResult)
    }
    OPTIONAL
}
(CONSTRAINED BY {-- must conform to the above definition --}
! RejectProblem : general-mistypedPDU)
```

NOTA – En la práctica, el uso de reglas que no se autodelimitan y autoidentifican puede impedir la distinción entre violaciones de construcción a nivel externo e interno.

9.4.2 Se debe suministrar un solo parámetro ASN.1 para producir un tipo totalmente definido, a saber, **Operations**, aquellas cuyo resultado puede ser devuelto.

9.4.3 La PDU resultante tiene hasta tres componentes, como sigue:

- invokeId** – Este componente identifica la invocación particular cuya ejecución satisfactoria se está informando. Será el InvokeId de una invocación previa para la cual no se ha informado aún el resultado; si no es así, se generará una PDU Reject{}, cuyo componente problem toma el valor de excepción: returnResult-unrecognisedInvocation. La invocación anterior será de una operación que devuelve un resultado; si no, se generará una PDU Reject{}, cuyo componente problem toma el valor de excepción: returnResult-resultResponseUnexpected.
- opcode** – Este componente, está presente solamente si el componente result está presente, e identificará, por medio de su &operationCode, una de las operations, específicamente, la indicada por el invokeId; si no, se generará una PDU Reject{}, cuyo componente problem toma el valor de excepción: returnResult-unrecognizedInvocation.
- result** – Este componente estará presente y es del tipo que forma el campo &ResultType de la operación identificada por el componente opcode, a menos que ese campo se omita, en cuyo caso este componente se omitirá. Si no se cumple esta condición, se generará una PDU Reject{}, cuyo componente problem toma el valor de excepción: general-mistypedPDU.

9.4.4 Si el receptor de esta PDU encuentra que está mal tipificada, se genera una PDU Reject{} cuyo componente problem toma el valor de excepción: general-mistypedPDU.

9.5 Retorno de error

9.5.1 El tipo parametrizado ReturnError{} proporciona una PDU para informar la ejecución infructuosa de operaciones. Se especifica como sigue:

```
ReturnError {ERROR:Errors} ::= SEQUENCE
{
    invokeId      InvokeId
                  (CONSTRAINED BY {-- must be that for an outstanding operation --}
                  ! RejectProblem   : returnError-unrecognizedInvocation)
    errcode       ERROR.&errorCode
                  (CONSTRAINED BY {-- which returns an error --}
                  ! RejectProblem   : returnError-errorResponseUnexpected),
    parameter     ERROR.&parameterType
                  ({Errors}
                  ! RejectProblem   : returnError-unrecognizedError)
                  (CONSTRAINED BY {-- must be in the &Errors field of the associated
operation --}
                  ! RejectProblem   : returnError-unexpectedError),
    OPTIONAL
    parameter     ERROR.&parameterType
                  ({Errors}{@errcode}
                  ! RejectProblem   : returnError-mistypedParameter)
}
(CONSTRAINED BY { -- must conform to the above definition -- }
! RejectProblem   : general-mistypedPDU)
```

NOTA – En la práctica, el uso de reglas que no se autodelimitan y autoidentifican puede impedir la distinción entre violaciones de construcción a nivel externo e interno.

9.5.2 Se debe suministrar un solo parámetro ASN.1 para producir un tipo totalmente definido, a saber, errors, que es el conjunto de errores para aquellas operaciones cuyos errores pueden ser devueltos (véase 9.10).

9.5.3 La PDU resultante tiene hasta tres componentes, como sigue:

- invokeId** – Este componente identifica la invocación particular cuya ejecución infructuosa se está informando. Será el InvokeId de una invocación anterior para la cual no se ha informado aún el resultado; si no es así, se generará una PDU Reject{}, cuyo componente problem toma el valor de excepción: returnError-unrecognisedInvocation. La invocación anterior será de una operación que devuelve un error; si no, se generará una PDU Reject{} cuyo componente problem toma el valor de excepción: returnError-errorResponseUnexpected.

- b) **errcode** (código de error) – Este componente identificará, por medio de su **&errorCode**, uno de los errores de Errors; si no, se generará una PDU Reject{}, cuyo componente **problem** toma el valor de excepción: **returnError-unrecognisedError**. El error será uno de los que aparecen en el campo **&Errors** de la operación identificada por el **invokeId**; si no, se generará una PDU Reject{}, cuyo componente **problem** toma el valor de excepción: **returnError-unexpectedError**.
- c) **parameter** (parámetro) – Este componente estará presente y es del tipo que forma el campo **&ParameterType** (tipo de parámetro) del error identificado por el componente **errcode**, a menos que este campo se omita, en cuyo caso este componente se omitirá. Si esta condición no se cumple, se generará una PDU Reject{} cuyo componente **problem** toma el valor de excepción: **returnError-mistypedParameter**.

9.5.4 Si el receptor de esta PDU encuentra que está mal tipificada, se genera una PDU Reject{} cuyo componente **problem** toma el valor de excepción: **general-mistypedPDU**.

9.6 Rechazo

9.6.1 El tipo Reject (rechazo) proporciona una PDU para informar la utilización errónea de otras PDU ROS{}. Se especifica como sigue:

Reject ::= SEQUENCE				
{				
invokeId	InvokeId,			
problem	CHOICE			
{				
	general	[0]	GeneralProblem,	
	invoke	[1]	InvokeProblem,	
	returnResult	[2]	ReturnResultProblem,	
	returnError	[3]	ReturnErrorProblem	
}				
}				
(CONSTRAINED BY {-- must conform to the above definition --}				
! RejectProblem	: general-mistypedPDU)			

NOTA – En la práctica, el uso de reglas que no se autodelimitan y autoidentifican puede impedir la distinción entre violaciones de construcción a nivel externo e interno.

9.6.2 La PDU resultante tiene dos componentes, como sigue:

- a) **invokeId** – Este componente es el **InvokeId** que aparece en la PDU que se rechaza, salvo que si el **invokeId** no puede ser determinado, en su lugar se envía el valor **noInvokeId** (véase 9.9).
- b) **problem** – Este componente identifica el problema particular con la PDU rechazada. Hay cuatro categorías posibles, como se especifica en 9.6.3 a 9.6.6.

9.6.3 Un **GeneralProblem** (problema general) es un problema fundamental con la forma o estructura de una PDU ROS{}. Las posibilidades se especifican como sigue:

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

e indican lo siguiente:

- a) **unrecognisedPDU** (PDU no reconocida) – La etiqueta de la PDU indica que no es una de las alternativas permitidas por 9.2;
- b) **mistypedPDU** (PDU mal tipificada) – La estructura de la PDU no se conforma con la definición apropiada;
- c) **badlyStructuredPDU** (PDU mal estructurada) – La estructura de la PDU no se puede determinar sobre la base de la sintaxis abstracta prevista.

NOTA – En algunas correspondencias, estos problemas se identificarán y tratarán dentro de la infraestructura de comunicaciones.

9.6.4 Un `InvokeProblem` (problema de invocación) indica que algún componente de una PDU `Invoke{}` era erróneo. Las posibilidades se especifican como sigue:

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

e indican lo siguiente:

- a) `duplicateInvocation` (invocación duplicada) [véase 9.3.3 a)];
- b) `unrecognisedOperation` (operación no reconocida) [véase 9.3.3 c)];
- c) `mistypedArgument` (argumento mal tipificado) [véase 9.3.3 d)];
- d) `resourceLimitation` (limitación de recursos). El realizador previsto no desea ejecutar la operación debido a limitación de recursos;
- e) `releaseInProgress` (liberación en curso). El realizador previsto no desea ejecutar la operación porque está a punto de liberar la asociación;
- f) `unrecognizedLinkId` (id enlazado no reconocido) [véase 9.3.3 b)];
- g) `linkedResponseUnexpected` (respuesta enlazada inesperada) [véase 9.3.3 b)];
- h) `unexpectedLinkedOperation` (operación enlazada inesperada) [véase 9.3.3 c)].

9.6.5 `ReturnResultProblem` (problema de retorno de resultado) indica que algún componente de una PDU `ReturnResult{}` era erróneo. Las posibilidades se especifican como sigue:

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

e indican lo siguiente:

- a) `unrecognisedInvocation` (invocación no reconocida) [véase 9.4.3 a)];
- b) `resultResponseUnexpected` (respuesta de resultado inesperada) [véase 9.4.3 a)];
- c) `mistypedResult` (resultado mal tipificado) [véanse 9.4.3 b) y 9.4.3 c)].

9.6.6 `ReturnErrorProblem` (problema de retorno de error) indica que algún componente de una PDU `ReturnError{}` era erróneo. Las posibilidades se especifican como sigue:

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

e indican lo siguiente:

- a) `unrecognisedInvocation` (invocación no reconocida) [véase 9.5.3 a)];
- b) `errorResponseUnexpected` (respuesta de error inesperada) [véase 9.5.3 a)];
- c) `unrecognisedError` (error no reconocido) [véase 9.5.3 b)];
- d) `unexpectedError` (error inesperado) [véase 9.5.3 b)];
- e) `mistypedParameter` (parámetro mal tipificado) [véase 9.5.3 c)].

9.6.7 Si el receptor de esta PDU encuentra que está mal tipificada, no generará una nueva PDU Reject{ }.

9.7 Problema de rechazo

El entero RejectProblem (problema de rechazo) describe el valor de código de error generado cuando se viola algún tipo o restricción en una definición de 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-unrecognizedLinkId (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 La reacción a la señalización de un error es el envío de una PDU Reject{ }. Cuando el error señalado es denotado por el identificador de RejectProblem « α - β » para algunas cadenas α y β , el componente problem de la PDU Reject{ } tomará el valor « α : β ».

9.8 Id de invocación

9.8.1 El tipo InvokeId define los valores que se pueden utilizar para identificar una invocación particular de una operación. Se especifica como sigue:

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

9.8.2 Cuando InvokeId está presente, es un valor de tipo INTEGER (entero). Cuando está ausente, se utiliza en su lugar un valor NULL (nulo).

9.9 No Id de invocación

9.9.1 noInvokeId es el valor de InvokeId utilizado cuando un valor INTEGER no se necesita o no está disponible. Se especifica como sigue:

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

9.9.2 NoInvokeId es un conjunto de valores de tipo InvokeId que consiste solamente en el valor noInvokeId. Se especifica como sigue:

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

9.10 Errores

El conjunto de errores parametrizado Errors{ }, dado un conjunto de operations como su parámetro ASN.1, es el conjunto de todos los errores en el campo &Errors de estas Operations. Se especifica como sigue:

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

9.11 Vinculación

El tipo parametrizado Bind{} (vinculación) dada una sola operación de vinculación como su parámetro ASN.1, proporciona las PDU para la invocación de esa operación, para la devolución de resultado de esa operación y para el informe de un error. Se especifica como sigue:

```
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 Desvinculación

El tipo parametrizado Unbind{} (desvinculación), dada una sola operación de desvinculación como su parámetro ASN.1, proporciona las PDU para la invocación de esa operación, para la devolución del resultado de esa operación y para el informe de un error. Se especifica como sigue:

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

10 Definiciones útiles

10.1 Introducción

En esta subcláusula figura un conjunto de definiciones que pueden ser útiles a los diseñadores de aplicaciones basadas en ROS y que comprenden:

- operaciones generalmente útiles (emptyBind, emptyUnbind, no-op) y sus errores asociados;
- definiciones parametrizadas que entregan los conjuntos de operaciones que intervienen en algún lote de operaciones (ConsumerPerforms{}, SupplierPerforms{}, AllOperations{}) y algunas definiciones auxiliares;
- una definición parametrizada que permite derivar una operación de otra cambiando el código de operación (recode{});
- definiciones parametrizadas que permiten definir lotes de operaciones mediante la conmutación de cometidos de otro, o combinando varios otros (switch{}, combine{});
- tipos parametrizados que se pueden utilizar para definir sintaxis abstractas que corresponden a un lote de operaciones (ROS-SingleAS{}, ROS-ConsumerAS{}, ROS-SupplierAS{}).

10.2 Vinculación vacía

La operación emptyBind (vinculación vacía) proporciona la operación de vinculación más simple, que es la operación por defecto si no se especifica explícitamente una operación de vinculación para algún lote de conexiones. Esta operación no tiene argumento ni valores de resultado y tiene un solo error posible, refuse (denegación) (véase 10.4) que corresponde a una denegación de la asociación. La operación es síncrona, lo que significa que no se pueden invocar operaciones síncronas hasta que se haya devuelto el resultado de la operación de vinculación. La operación emptyBind se especifica como sigue:

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

10.3 Desvinculación vacía

La operación `emptyUnbind` (desvinculación vacía) proporciona la operación de desvinculación más simple que es la operación por defecto si no se especifica explícitamente una operación de desvinculación para algún lote de conexiones. Esta operación no tiene argumentos, ni valores de error o de resultado. La operación es síncrona, lo que significa que la operación de desvinculación no puede invocarse hasta que se haya devuelto el resultado de cualquier operación síncrona pendiente. La operación `emptyUnbind` se especifica como sigue:

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

10.4 Denegación

El error `refuse` (denegación) proporciona la denegación de alguna petición, sin proporcionar ninguna razón. Se especifica como sigue:

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

10.5 No-op

10.5.1 La operación `no-op` (no operación) no hace nada. Se especifica como sigue:

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

10.5.2 Esta operación no retorna resultado.

10.6 Directa

El conjunto de operaciones parametrizadas `Forward{}` (Directa), dado un `OperationSet` (conjunto de operaciones) como su parámetro ASN.1, es el conjunto de operaciones ampliado formado mediante la adición de cualesquiera operaciones indirectamente enlazadas con operaciones del conjunto y con la misma direccionalidad. Se especifica como sigue:

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

Se supone que no hay operaciones en el nivel quinto o ulterior de enlace que no estén también presentes en niveles anteriores. Si para algún `OperationSet` esta hipótesis no es válida, entonces se requerirá algún otro método para identificar el conjunto completo de operaciones implicadas.

10.7 Inversa

El conjunto de operaciones parametrizado `Reverse{}` (Inversa) dado un `OperationSet` como su parámetro ASN.1, consiste en todas las operaciones directa o indirectamente enlazadas con operaciones del conjunto, con direccionalidad opuesta. Se especifica como sigue:

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

Se supone que no hay operaciones en el nivel sexto o ulterior de enlace que no estén también presentes en niveles anteriores. Si para algún `OperationSet` esta hipótesis no es válida, se requerirá algún otro método para identificar el conjunto completo de operaciones implicadas.

10.8 Acciones de consumidor

El conjunto de operaciones parametrizado `ConsumerPerforms{}` (acción de consumidor) dado un lote de operaciones `package` como su parámetro ASN.1, es el que contiene todas las operaciones que el consumidor tiene que ser capaz de realizar. Se especifica como sigue:

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

Sólo se puede utilizar si son válidas las hipótesis asociadas con `Forward{}` y `Reverse{}` en relación con las operaciones enlazadas.

10.9 Acciones de suministrador

El conjunto de operaciones parametrizado `SupplierPerforms{}` (acciones del suministrador), dado un lote de operaciones `package` como su parámetro ASN.1, es el que contiene todas las operaciones que el suministrador tiene que ser capaz de realizar. Se especifica como sigue:

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

Sólo se puede utilizar si son válidas las hipótesis asociadas con `Forward{}` y `Reverse{}` en relación con las operaciones enlazadas.

10.10 Todas las operaciones

El conjunto de operaciones parametrizadas `AllOperations{}` (todas las operaciones), dado un lote de operaciones `package` como su parámetro ASN.1, es el que contiene todas las operaciones que participan en el lote. Se especifica como sigue:

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

Sólo se puede utilizar si son válidas las hipótesis asociadas con `Forward{}` y `Reverse{}` en relación con las operaciones enlazadas.

10.11 Recodificación

La operación parametrizada `recode{ }` (recodificación), dada una `operation` como su primer parámetro ASN.1, es idéntica en todos los aspectos a esa `operation`, salvo que su campo `&code` (código) tiene el valor del segundo parámetro ASN.1, `code`. Se especifica como sigue:

```

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 Conmutación

El lote parametrizado `switch{ }` (conmutación), dado un `package` como su primer parámetro ASN.1, es idéntico en todos los aspectos a ese `package`, salvo que los cometidos de consumidor y de suministrador se invierten y su campo `&id` tiene el valor del segundo parámetro ASN.1. Se especifica como sigue:

```

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

```

10.13 Combinación

10.13.1 El lote de operaciones parametrizadas `combine{ }` (combinación) combina varios lotes de operaciones en uno. Se especifica como sigue:

```

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 Los parámetros ASN.1 que se deben suministrar para producir un lote de operaciones completamente definido son los siguientes:

- ConsumerConsumes (consumidor consume) – El conjunto de lotes de operaciones en los cuales el consumidor del lote de operaciones resultante desempeñará el cometido de consumidor;
- ConsumerSupplies (consumidor suministra) – El conjunto de lotes de operaciones en los cuales el consumidor del lote de operaciones resultante desempeñará el cometido de suministrador;
- base (base) – Un lote de operaciones, definido típicamente de manera incompleta, cuyo campo `&id` se utiliza para definir el lote de operaciones resultante.

NOTA – El lote base no tendrá normalmente operaciones definidas; incluso si las tiene, no aparecerán en el lote de operaciones resultante, a menos que aparezcan también en ConsumerConsumes o SupplierSupplies.

10.14 Una sola sintaxis abstracta ROS

El tipo parametrizado ROS-SingleAS{} (una sola sintaxis abstracta ROS) es la base de una sintaxis abstracta que permite la invocación en el informe de todas las operaciones en algún package, utilizando identificadores de invocación del InvokeIdSet. Se especifica como sigue:

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

10.15 Sintaxis abstracta de consumidor ROS

El tipo parametrizado ROS-ConsumerAS{} (sintaxis abstracta de consumidor ROS) es la base de una sintaxis abstracta que permite la invocación de todas las operaciones realizadas por el consumidor, y el informe de todas las operaciones realizadas por el suministrador en algún package, utilizando los identificadores de invocación de InvokeIdSet. Se especifica como sigue:

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

10.16 Sintaxis abstracta de suministrador ROS

El tipo parametrizado ROS-SupplierAS{} (sintaxis abstracta de suministrador ROS) es la base de una sintaxis abstracta que permite la invocación de todas las operaciones realizadas por el suministrador, y el informe de todas las operaciones realizadas por el consumidor en algún package, utilizando identificadores de invocación de InvokeIdSet. Se especifica como sigue:

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

Anexo A

Módulos ASN.1

(Este anexo es parte integrante de la presente Recomendación | Norma Internacional)

```

Remote-Operations-Information-Objects {joint-iso-itu-t remote-operations(4) informationObjects(5) version1(0)}
DEFINITIONS ::=
BEGIN
-- exports everything
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
}
-- continued on the next page

```

```

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 -- end of Information Object specifications

```

```

Remote-Operations-Generic-ROS-PDUs {joint-iso-itu-t remote-operations(4) generic-ROS-PDUs(6) version1(0)}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
-- exports everything
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 {-- must conform to the above definition --}
! RejectProblem : general-unrecognizedPDU)

Invoke {InvokeId:InvokeIdSet, OPERATION:Operations} ::= SEQUENCE
{
    invokeId      InvokeId      (InvokeIdSet)
                                (CONSTRAINED BY {-- must be unambiguous --}
                                ! RejectProblem : invoke-duplicateInvocation),
    linkedId      CHOICE {
                                present [0] IMPLICIT present < InvokeId,
                                absent  [1] IMPLICIT NULL
                                }
                                (CONSTRAINED BY {-- must identify an outstanding operation --}
                                ! RejectProblem : invoke-unrecognizedLinkId)
                                (CONSTRAINED BY {-- which has one or more linked operations --}
                                ! RejectProblem : invoke-linkedResponseUnexpected)
                                OPTIONAL,
    opcode        OPERATION.&operationCode
                                ({Operations}
                                ! RejectProblem : invoke-unrecognizedOperation),
    argument      OPERATION.&ArgumentType
                                ({Operations} {@opcode}
                                ! RejectProblem : invoke-mistypedArgument)
                                OPTIONAL
}
(CONSTRAINED BY {-- must conform to the above definition --}
! RejectProblem : general-mistypedPDU)
(
    WITH COMPONENTS
    {...,
    linkedId ABSENT
    }
|
    WITH COMPONENTS
    {...,
    linkedId PRESENT,
    opcode
    (CONSTRAINED BY {-- must be in the &Linked field of the associated operation --}
    ! RejectProblem : invoke-unexpectedLinkIdOperation)
    }
)
-- continued on the next page

```

```

ReturnResult {OPERATION:Operations} ::= SEQUENCE
{
    invokeId      InvokeId
                  (CONSTRAINED BY {-- must be that for an outstanding operation --}
                  ! RejectProblem      : returnResult-unrecognizedInvocation)
                  (CONSTRAINED BY {-- which returns a result --}
                  ! RejectProblem      : returnResult-resultResponseUnexpected),
    result        SEQUENCE
    {
        opcode     OPERATION.&operationCode
                  ({Operations})(CONSTRAINED BY {-- identified by invokeId --}
                  ! RejectProblem      : returnResult-unrecognizedInvocation),
        result      OPERATION.&ResultType
                  ({Operations} {@opcode}
                  ! RejectProblem      : returnResult-mistypedResult)
    }
    OPTIONAL
}
(CONSTRAINED BY {-- must conform to the above definition --}
! RejectProblem : general-mistypedPDU)

ReturnError {ERROR:Errors} ::= SEQUENCE
{
    invokeId      InvokeId
                  (CONSTRAINED BY {-- must be that for an outstanding operation --}
                  ! RejectProblem      : returnError-unrecognizedInvocation)
                  (CONSTRAINED BY {-- which returns an error --}
                  ! RejectProblem      : returnError-errorResponseUnexpected),
    errcode       ERROR.&errorCode
                  ({Errors}
                  ! RejectProblem : returnError-unrecognizedError)
                  (CONSTRAINED BY {-- must be in the &Errors field of the associated operation --}
                  ! RejectProblem : returnError-unexpectedError),
    parameter     ERROR.&ParameterType
                  ({Errors} {@errcode}
                  ! RejectProblem : returnError-mistypedParameter) OPTIONAL
}
(CONSTRAINED BY {-- must conform to the above definition --}
! RejectProblem : general-mistypedPDU)

Reject ::= SEQUENCE
{
    invokeId      InvokeId,
    problem        CHOICE
    {
        general    [0] GeneralProblem,
        invoke      [1] InvokeProblem,
        returnResult [2] ReturnResultProblem,
        returnError [3] ReturnErrorProblem
    }
}
(CONSTRAINED BY {-- must conform to the above definition --}
! RejectProblem : general-mistypedPDU)

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

```

-- continued on the next page

```

InvokeProblem ::= INTEGER
{
    duplicateInvocation (0),
    unrecognizedOperation (1),
    mistypedArgument (2),
    resourceLimitation (3),
    releaseInProgress (4),
    unrecognizedLinkId (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-unrecognizedLinkId (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}

-- continued on the next page

```

```

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 -- end of generic ROS PDU definitions

```

```

Remote-Operations-Useful-Definitions {joint-iso-itu-t remote-operations(4) useful-definitions(7) version1(0)}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
-- exports everything
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}
}

-- continued on the next page

```

```

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 -- end of useful definitions.

```

Anexo B

Directrices para el uso de la notación

(Este anexo no es parte integrante de la presente Recomendación | Norma Internacional)

El presente anexo proporciona ejemplos y directrices para los diseñadores de protocolos de aplicación sobre la utilización de las clases de objetos de información correspondientes a los conceptos básicos de ROS y sobre la aplicación del conjunto de definiciones útiles parametrizadas.

B.1 Ejemplos de operaciones y sus errores

Esta cláusula contiene algunos ejemplos de objetos de información pertenecientes a la clase OPERATION y 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
}
```

La operación distante asíncrona operationExample1, que transporta un argumento de tipo ArgumentType1, siempre responde (como lo implica la ausencia de las palabras clave ALWAYS RESPONDS), devolviendo, en el caso de la ejecución satisfactoria de la operación, un valor de resultado de tipo ResultType1, o, si la operación no tiene éxito, uno de los errores errorExample1 o errorExample2, según las circunstancias del fallo. La operación operationExample2 está «enlazada» con esta operación, lo que significa que se puede invocar operationExample2 en respuesta a esta operación en cualquier momento antes de completarse esta operación. Esta operationExample1 es identificada por su código, el entero 1.

La operación distante asíncrona operationExample2 transporta un argumento ArgumentType2. Es una operación que nunca falla o, si falla, no se informa el fallo. Cuando se informa el resultado satisfactorio de esta operación, el valor de resultado de tipo ResultType2 puede ser omitido facultativamente por el realizador. La operación operationExample4 está «enlazada» con esta operación, lo que significa que se puede invocar operationExample4 en respuesta a operationExample2 en cualquier momento antes de completarse la segunda. Esta operación es identificada por el valor entero 2.

La operación distante síncrona operationExample3, identificada por el valor entero 3 y que transporta un argumento de tipo ArgumentType3, siempre informa su resultado. Si la operación se realiza con éxito, se devuelve una indicación pero no el valor del resultado. Si la operación falla, se devuelve un error errorExample3.

NOTA – Como ésta es una operación síncrona, el diseñador de aplicaciones debe estar seguro de que esta operación siempre retorna un resultado, particularmente si el invocador prevé invocar otras operaciones síncronas.

La operación asíncrona `operationExample4`, identificada por el valor entero 4, y cuya invocación está acompañada por un argumento de tipo `ArgumentType4`, no retorna resultado.

A continuación se dan ejemplos de la clase (de objeto de información) `ERROR` que se utilizan para informar la ejecución infructuosa de (algunas de) las operaciones definidas anteriormente:

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

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

errorExample3 ERROR
{
  CODE           local:3
}
```

`errorExample1`, que se puede utilizar para informar la ejecución infructuosa de `operationExample1` u `operationExample2`, es identificada por el valor entero 1. Un valor de parámetro de diagnóstico de error del tipo `ParameterType1` acompaña a este error.

`errorExample2`, que se puede utilizar para informar la compleción infructuosa de `operationExample1`, está acompañado por un valor del tipo `ParameterType2`. A discreción del realizador, este valor puede omitirse algunas veces. Es identificado por el entero 2.

`errorExample3`, que es identificado por el valor entero 3, se utiliza para informar la ejecución infructuosa de `operationExample3`. No se devuelve ningún valor de parámetro (es decir, diagnóstico de error) con este error.

B.2 Ejemplos de lotes de operaciones y utilización de conmutación

Las operaciones y errores definidos en B.1 se pueden agrupar en un lote de operaciones, `package1`, como sigue:

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

De los dos objetos ROS que interactúan, uno de ellos, denominado arbitrariamente el «consumidor», puede invocar `operationExample1` y `operationExample3`. El otro objeto ROS, designado el «suministrador», sólo puede invocar `operationExample2`. Esta combinación particular es identificada globalmente por el valor `objectIdentifierOfPackage1`.

NOTA – Los términos «consumidor» y «suministrador», que distinguen los dos objetos ROS que interactúan, tienen que definirse con referencia a algo fuera de las operaciones a distancia. El caso más simple es definirlos con respecto a sus cometidos al formar un contrato de asociación (véase B.5).

Una combinación diferente del mismo conjunto de operaciones forma `package2`, definido como sigue:

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

En `package2`, cualquiera de los dos objetos puede invocar `operationExample3`, el consumidor puede invocar `operationExample1`, mientras que ninguno de los lados puede invocar `operationExample2`.

Un tercer lote de operaciones, package3, se puede derivar de package1 utilizando la definición parametrizada switch{}:

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

El uso de switch{} invierte las operaciones que pueden ser invocadas por el «consumidor» y el «suministrador» en package1 y asigna un nuevo valor de identificador de objeto a esta combinación. De este modo, package3 escrito completamente es como sigue:

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

B.3 Ejemplos de operaciones de vinculación y desvinculación

Se puede establecer dinámicamente una asociación invocando una operación bind, un ejemplo de la cual es:

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

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

La operación síncrona bindExample1 se utiliza para establecer una asociación entre dos objetos ROS. La invocación de la operación va acompañada por un valor de argumento de tipo BindArgumentType1 y el establecimiento satisfactorio de la vinculación va acompañado de un valor de resultado de tipo BindResultType1. Cuando la asociación no se establece satisfactoriamente, se devolverá un error bindError1 con el parámetro de diagnóstico acompañante BindErrorType1 presente facultativamente.

La liberación de la asociación se realiza mediante la invocación de otra operación síncrona unBindExample1 definida como sigue:

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

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

B.4 Ejemplos de lotes de conexiones

Los ejemplos `bindExample1` y `unBindExample1` se pueden utilizar para definir un lote de conexiones `connectionPackage1`, que se puede utilizar para establecer o liberar dinámicamente una asociación entre dos objetos ROS.

```

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

```

`connectionPackage 1`, que es identificado globalmente por el valor `objectIdentifierOfConnectionPackage1` cuando se establece o anuncia dinámicamente la asociación entre los dos objetos ROS, utiliza las operaciones `bindExample1` y `unBindExample1` para establecer y terminar la asociación, respectivamente. Permite al respondedor del establecimiento de la asociación desvincular y permite que la asociación permanezca, incluso si la operación `unBindExample1` falla.

A continuación se da un ejemplo de un lote de conexiones muy simple que suple a la utilización de las operaciones `emptyBind` y `emptyUnbind` (véanse 10.2 y 10.3) para establecer y terminar, respectivamente una asociación:

```

simpleConnectionPackage CONNECTION-PACKAGE ::=
{
  ID    objectIdentifierOfSimpleConnectionPackage
}

```

Para este lote de conexión, sólo el iniciador del establecimiento de la asociación puede invocar la vinculación y la asociación es liberada incluso si falla la tentativa de desvinculación.

B.5 Ejemplo de un contrato de asociación

El contrato de asociación `contract1`, definido como:

```

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

```

muestra que el lote de conexión `connectionPackage1` se utiliza para establecer y terminar la asociación y que el objeto ROS que inicia el establecimiento de la asociación desempeña el cometido de «consumidor» en el lote de operaciones `package1`. Este contrato es identificado por el valor `objectIdentifierOfContract1`.

B.6 Ejemplos de objetos ROS

Se puede definir un objeto ROS, `object1`:

```

object1 ROS-OBJECT-CLASS ::=
{
  INITIATES    {contract1}
  ID           objectIdentifierofROSOBJECT1
}

```

`object1`, que es identificado por `objectIdentifierOfROSOBJECT1`, es uno de un conjunto de objetos que puede interactuar con otros objetos ROS iniciando la interacción que utiliza el contrato de asociación `contract1`.

De manera similar, object2, definido como:

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

es uno de un conjunto de objetos que puede interactuar con otros objetos ROS respondiendo a la iniciación de las interacciones ofrecidas por el contrato de asociación contract1.

B.7 Ejemplo de utilización directa e inversa

El conjunto de objetos de información ConsumerInvokes, derivado de package1.&Supplier definido en B.2 como sigue:

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

origina el conjunto {operationExample1 | operationExample2} que enumera las operaciones que pueden ser invocadas por el «consumidor». A su vez:

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

produce {operationExample2} que es el conjunto de operaciones que puede ser invocado en dirección «inversa» es decir, por el «suministrador», mientras:

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

produce {operationExample4} que es el conjunto de operaciones enlazado indirectamente con el conjunto formado por package1.&Consumer y es invocado por el «consumidor».

El conjunto de operaciones Forward {OPERATION: ConsumerInvokes} es el conjunto de operaciones que incluye el conjunto de operaciones original ConsumerInvokes junto con cualesquiera operaciones enlazadas indirectamente con la misma «direccionalidad», es decir, en nuestro ejemplo, las que pueden ser invocadas por el «consumidor». De este modo:

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

Por otra parte, el conjunto de operaciones Reverse {OPERATION: ConsumerInvokes} es el conjunto de operaciones que son una parte del conjunto formado por package1.&Consumer.&Linked, que son las operaciones que pueden ser invocadas por el «suministrador» en respuesta a las del conjunto de operaciones dadas por ConsumerInvokes, junto con las enlazadas indirectamente con ella y con la misma «direccionalidad», esto es, de «consumidor» a «suministrador». Es decir:

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

que, cuando se escribe explícitamente, es:

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

Por otra parte, para el conjunto de operaciones SupplierInvokes derivado como sigue:

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

se obtiene {operationExample2}.

De este modo, se tiene:

```
Forward{OPERATION: SupplierInvokes} OPERATION ::= {operationExample2}
Reverse{OPERATION: SupplierInvokes} OPERATION ::= {operationExample4}
```

B.8 Ejemplos de acciones de consumidor{}, acciones de suministrador{} y todas las operaciones{}

Con la ayuda de los ejemplos mostrados en B.7, dado el lote de operaciones package1, el «consumidor» puede realizar todas las operaciones del conjunto:

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

que produce el conjunto de operaciones {operationExample1 | operationExample3 | operationExample4}.

De manera similar, en package1, todas las operaciones que el «suministrador» debe ser capaz de ejecutar son:

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

que es el conjunto {operationExample2}.

La utilidad de las construcciones ConsumerPerforms{} y SupplierPerforms{} es que permite a un diseñador de aplicaciones ver, en una situación compleja que consiste en muchos enlaces jerarquizados entre las operaciones en ese lote, aquéllas con la misma «direccionalidad», es decir, las que pueden ser invocadas por el consumidor y las que pueden ser invocadas por el suministrador, con respecto a la naturaleza de sus enlaces.

El conjunto AllOperations {OPERATION-PACKAGE:package1} enumera todas las operaciones citadas implícitamente (es decir, por enlaces) y explícitamente por package1 que pueden ser invocadas por cualquiera de los dos lados en algún caso de la utilización de este lote:

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

que produce el conjunto de operaciones {operationExample1 | operationExample2 | operationExample3 | operationExample4}.

Anexo C

Sustitución gradual de las macros ROS

(Este anexo no es parte integrante de la presente Recomendación | Norma Internacional)

En versiones anteriores de ROS, se proporcionaron macros ASN.1 para que los diseñadores de aplicaciones ROS pudieran especificar sus operaciones, errores, operaciones de vinculación, elementos de servicio de aplicación, etc. Además, y estrechamente asociada con ROS, la Recomendación X.407 del CCITT proporcionaba otras macros que los diseñadores podían utilizar en Especificaciones, basadas en objetos, de aplicaciones distribuidas. La capacidad de las macros se está eliminando por etapas de la ASN.1 y, en consecuencia, la notación ROS proporcionada por la presente Especificación utiliza en cambio la notación «sustitución de macros» y de ASN.1, que incluye clases de objetos de información y parametrización.

C.1 Introducción

Hay, sin embargo, un número importante de especificaciones en existencia que utilizan el método de macros. En consecuencia, este anexo muestra cómo la utilización de estas macros puede transformarse en la utilización de la notación de sustitución. En muchos casos, la notación de sustitución es más general que la macro que sustituye; no obstante, estas características no se señalan generalmente en este apéndice, que sólo se relaciona con la comprensión de la utilización de las macros existentes. Las macros no deben utilizarse para nuevas especificaciones.

Este anexo está organizado con una subcláusula dedicada a cada macro existente. La subcláusula describe la finalidad de la macro, da un ejemplo de su uso junto con el equivalente en la notación más nueva y explica cualesquiera aspectos del ejemplo que no están claros. En los ejemplos que utilizan la notación de macros, los símbolos subrayados deben suprimirse para formar la nueva notación, mientras que han de insertarse los símbolos en *cursivas negritas*.

En las Especificaciones existentes se ha empleado ampliamente un método particular de utilización de las macros, «la definición en dos etapas». En este método, los «identificadores» para distintas clases de objetos de información (por ejemplo, el código de operación o de error) se asigna en la segunda etapa. Sin embargo, para que el lector pueda interpretar estas especificaciones, algunas veces tiene que tener en cuenta factores normalmente no pertinentes, tales como la similitud de ortografía de distintas referencias ASN.1 y la proximidad de definiciones ASN.1. Esto no es aceptable con respecto a la notación de sustitución. En consecuencia, este anexo sólo describe la correspondencia de la segunda o única etapa de la definición que utiliza macros.

C.2 Operación

La macro OPERATION se utilizó para especificar operaciones (salvo para las operaciones de vinculación y desvinculación). Los cambios, según pasamos de definir un caso, denominado get, de la macro OPERATION a un miembro de la clase objeto de información OPERATION, se muestran a continuación según pasamos, respectivamente, de derecha a izquierda. Si se observa la notación macro en el lado izquierdo, los símbolos con subrayado doble deben suprimirse de la antigua notación mientras que los símbolos en cursivas negritas han de insertarse en la antigua notación para formar la nueva notación (como se ve en el lado derecho).

```
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
}
```

NOTAS

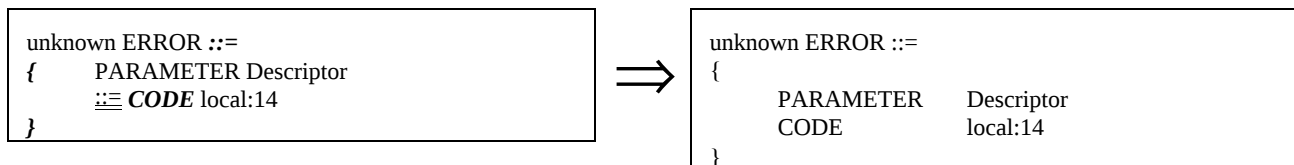
- 1 La omisión de la cláusula RESULT en la notación macro $\Rightarrow$ RETURN RESULT FALSE en la nueva notación.
- 2 La presencia de RESULT sin tipo de datos en la notación macro $\Rightarrow$ omisión de la cláusula RESULT en la nueva notación (las palabras clave RETURN RESULT TRUE son las supletorias y por tanto se omiten).
- 3 Como resultado de la hipótesis anterior en relación con la definición en dos fases, los nombres de errores individuales y operaciones enlazadas comienzan con letras minúsculas.

4 Los siguientes campos de operaciones permitidos por la definición de clase de objeto de información no pueden ser especificados con la macro, pero, si es necesario, tendrían que especificarse en texto: &synchronous, &InvokePriority, &ResultPriority.

5 Es posible indicar en la nueva notación, mediante el uso de los campos &argumentTypeOptional y &resultTypeOptional, si se puede omitir, respectivamente, el valor de argumento o resultado, como una opción del usuario.

C.3 Error

La macro ERROR se utilizaba para especificar errores (salvo para las operaciones de vinculación y desvinculación).

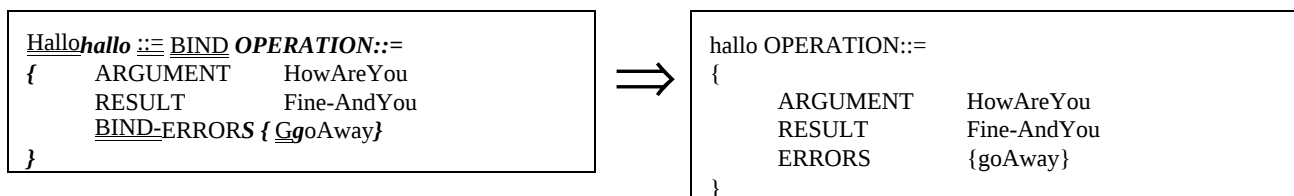


NOTAS

- 1 El campo de &ErrorPriority no puede especificarse con la macro, pero si es necesario, se especificará en texto.
- 2 Es posible indicar en la nueva notación, mediante el uso del campo ¶meterTypeOptional, si se puede omitir el valor de parámetro, si hubiere alguno, definido para acompañar el informe de un error, como una opción del usuario.

C.4 Vinculación

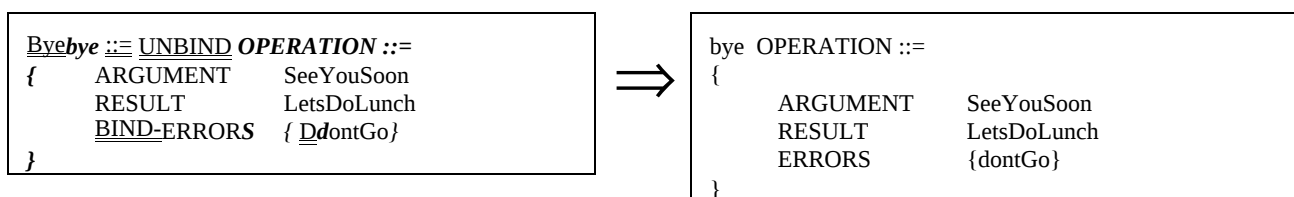
La macro BIND se utilizaba para especificar operaciones de vinculación.



NOTA – La ausencia de la palabra clave CODE en la nueva notación indica que ésta es una operación especial que no puede ser invocada utilizando la PDU Invoke{ }.

C.5 Desvinculación

La macro UNBIND se utilizaba para especificar operaciones de desvinculación.



NOTA – La ausencia de la palabra clave CODE en la nueva notación indica que ésta es una operación especial que no puede ser invocada utilizando la PDU Invoke{ }.

Anexo D**Asignación de valores de identificador de objeto**

(Este anexo no es parte integrante de la presente Recomendación | Norma Internacional)

En esta Recomendación | Norma Internacional se asignan los siguientes valores de identificador de objeto:

Cláusula	Valor de identificador de objeto
Anexo 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)}