UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

UIT-T Q.816

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

(01/2001)

SERIE Q: CONMUTACIÓN Y SEÑALIZACIÓN

Interfaz Q3

Servicios de la red de gestión de las telecomunicaciones basados en CORBA

Recomendación UIT-T Q.816

(Anteriormente Recomendación del CCITT)

RECOMENDACIONES UIT-T DE LA SERIE Q

CONMUTACIÓN Y SEÑALIZACIÓN

SEÑALIZACIÓN EN EL SERVICIO MANUAL INTERNACIONAL	Q.1–Q.3
EXPLOTACIÓN INTERNACIONAL SEMIAUTOMÁTICA Y AUTOMÁTICA	Q.4–Q.59
FUNCIONES Y FLUJOS DE INFORMACIÓN PARA SERVICIOS DE LA RDSI	Q.60–Q.99
CLÁUSULAS APLICABLES A TODOS LOS SISTEMAS NORMALIZADOS DEL UIT-T	Q.100–Q.119
ESPECIFICACIONES DE LOS SISTEMAS DE SEÑALIZACIÓN N.° 4 Y N.° 5	Q.120–Q.249
ESPECIFICACIONES DEL SISTEMA DE SEÑALIZACIÓN N.° 6	Q.250–Q.309
ESPECIFICACIONES DEL SISTEMA DE SEÑALIZACIÓN R1	Q.310–Q.399
ESPECIFICACIONES DEL SISTEMA DE SEÑALIZACIÓN R2	Q.400–Q.499
CENTRALES DIGITALES	Q.500–Q.599
INTERFUNCIONAMIENTO DE LOS SISTEMAS DE SEÑALIZACIÓN	Q.600–Q.699
ESPECIFICACIONES DEL SISTEMA DE SEÑALIZACIÓN N.° 7	Q.700–Q.799
INTERFAZ Q3	**Q.800–Q.849**
SISTEMA DE SEÑALIZACIÓN DIGITAL DE ABONADO N.° 1	Q.850–Q.999
RED MÓVIL TERRESTRE PÚBLICA	Q.1000–Q.1099
INTERFUNCIONAMIENTO CON SISTEMAS MÓVILES POR SATÉLITE	Q.1100–Q.1199
RED INTELIGENTE	Q.1200–Q.1699
REQUISITOS Y PROTOCOLOS DE SEÑALIZACIÓN PARA IMT-2000	Q.1700–Q.1799
RED DIGITAL DE SERVICIOS INTEGRADOS DE BANDA ANCHA (RDSI-BA)	Q.2000–Q.2999

Para más información, véase la Lista de Recomendaciones del UIT-T.

Recomendación UIT-T Q.816

Servicios de la red de gestión de las telecomunicaciones basados en CORBA

Resumen

Esta Recomendación define un conjunto de servicios que junto con la UIT-T X.780 comprende un marco para interfaces de la RGT basadas en CORBA. Especifica los requisitos de protocolo, los requisitos de utilización de servicio de objetos comunes CORBA y los servicios de soporte específicos de la RGT. Se proporciona el módulo IDL CORBA que define las interfaces de los servicios de soporte específico de la RGT.

Orígenes

La Recomendación UIT-T Q.816, preparada por la Comisión de Estudio 4 (2001-2004) del UIT-T, fue aprobada por el procedimiento de la Resolución 1 de la AMNT el 25 de enero de 2001.

Palabras clave

Arquitectura de negociación de petición de objetos comunes [*common object request broker architecture* (CORBA)], lenguaje de definición de interfaces (IDL), servicios CORBA, procesamiento distribuido, interfaces RGT, objetos gestionados.

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. Este órgano estudia los aspectos técnicos, de explotación y tarifarios y publica Recomendaciones sobre los mismos, con miras a la normalización de las telecomunicaciones en el plano mundial.

La Asamblea Mundial de Normalización de las Telecomunicaciones (AMNT), que se celebra cada cuatro años, establece los temas que han de estudiar las Comisiones de Estudio del UIT-T, que a su vez producen Recomendaciones sobre dichos temas.

La aprobación de Recomendaciones por los Miembros del UIT-T es el objeto del procedimiento establecido en la Resolución 1 de la AMNT.

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.

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.

PROPIEDAD INTELECTUAL

La UIT señala a la atención la posibilidad de que la utilización o aplicación de la presente Recomendación suponga el empleo de un derecho de propiedad intelectual reivindicado. La UIT no adopta ninguna posición en cuanto a la demostración, validez o aplicabilidad de los derechos de propiedad intelectual reivindicados, ya sea por los miembros de la UIT o por terceros ajenos al proceso de elaboración de Recomendaciones.

En la fecha de aprobación de la presente Recomendación, la UIT no ha recibido notificación de propiedad intelectual, protegida por patente, que puede ser necesaria para aplicar esta Recomendación. Sin embargo, debe señalarse a los usuarios que puede que esta información no se encuentre totalmente actualizada al respecto, por lo que se les insta encarecidamente a consultar la base de datos sobre patentes de la TSB.

ÍNDICE

Página

Recomendación UIT-T Q.816

Servicios de la red de gestión de las telecomunicaciones basados en CORBA

1 Ámbito de aplicación

La arquitectura de la RGT definida en UIT-T M.3010 (2000) introduce conceptos a partir del procesamiento distribuido e incluye la utilización de múltiples protocolos de gestión. Las especificaciones iniciales de las interfaces de la RGT para interfaces intra-RGT e inter-RGT se desarrollaron utilizando la notación de las directrices para la definición de objetos gestionados (GDMO) a partir de la gestión de sistemas OSI utilizando como protocolo el protocolo de información de gestión común (CMIP). La interfaz inter-RGT (X) incluye el CMIP y el CORBA GIOP/IIOP como posibles opciones en la capa de aplicación.

CORBA, una tecnología de procesamiento distribuido, se está considerando para su utilización en la arquitectura de comunicación de la RGT fundamentalmente debido a su aceptación por la industria de la tecnología de la información. Esta aceptación se espera que mejore la disponibilidad de las interfaces basadas en CORBA debido a las herramientas para un mejor desarrollo y a los muy amplios conocimientos en el desarrollo de las interfaces basadas en CORBA. Esta tecnología, desarrollada por el Grupo de gestión de objetos (OMG) también la están considerando múltiples industrias. Las especificaciones que utilizan esta tecnología permiten soportar interfaces de programación de aplicación normalizadas (API) y vinculaciones de lenguaje a lenguajes de programación, y también facilitan la portabilidad del soporte lógico. Las soluciones de interoperabilidad que ofrece la negociación de petición de objetos combinada con los protocolos entre ORB consideran la interoperabilidad entre el cliente y el servidor. Mientras que el CMIP y los modelos de información proporcionan soluciones para la interoperabilidad entre los sistemas gestor y de agente, CORBA define interacciones entre objetos en las que los objetos pueden estar distribuidos.

1.1 Objeto

Diversos grupos están desarrollando especificaciones de gestión de red que utilizan las técnicas de modelado CORBA con la notación IDL junto con servicios CORBA. El objeto de esta Recomendación es definir los requisitos de protocolo y los servicios comunes para su utilización en al especificación de interfaces de gestión de red interoperables basadas en CORBA. Los requisitos que se exigen a las interfaces "X" son diferentes de los utilizados "dentro de" una RGT, interfaces "Q". Esta Recomendación cubre todas las interfaces de la RGT en las que se puede utilizar CORBA. Se espera que no todas las capacidades y servicios definidos aquí sean necesarios para todas las interfaces de la RGT. Esto implica que el marco se puede utilizar para interfaces entre sistemas de gestión en todos los niveles de abstracción (entre y dentro de administraciones), así como entre sistemas de gestión y elementos de red.

Se pretende que esta Recomendación la utilicen diversos grupos para la especificación de interfaces de gestión de red. Se consideran algunos factores: la versión de CORBA a utilizar, el conjunto de servicios de objetos comunes CORBA y servicios adicionales. Esta Recomendación, junto con las directrices de modelado de objetos definidas en UIT-T X.780, constituye un *marco* para las interfaces de la RGT basadas en CORBA. La utilización de un marco común en las interfaces de gestión de telecomunicaciones tiene diversas ventajas. Por ejemplo:

– facilitar la reutilización de modelos que se han desarrollado para cumplir los requisitos genéricos de telecomunicaciones;

– adaptar los servicios CORBA para su utilización por la industria de las telecomunicaciones;

– facilitar la definición de nuevos servicios para la RGT, reutilizando la semántica del amplio conjunto de modelos existente y

– armonizar el planteamiento de modelado entre grupos que utilizan una única fuente como en UIT-T X.720, UIT-T X.721 y UIT-T X.722 para el CMIP.

La reutilización de un marco común y un modelo de información genérico para una diversidad de tecnologías de red y aplicaciones de gestión de red acelerará la introducción de nuevos servicios de red manteniendo bajo el coste del desarrollo de sistemas de gestión de red.

La industria de las telecomunicaciones ha invertido gran cantidad de tiempo y energía en el desarrollo de modelos de información para el protocolo de gestión de red CMIP. Un objetivo fundamental de este marco consiste en reutilizar estos modelos de información permitiendo su traducción al lenguaje de definición de interfaces (IDL) CORBA con pequeños cambios en la semántica (véase UIT-T X.780). En consecuencia, se espera obtener a partir de los modelos CMIP modelos iniciales de información IDL.

Además de aprovechar los modelos de información CMIP, otro objetivo del marco consiste en aprovechar CORBA. El marco nivela las funciones definidas en las especificaciones CORBA, incluido un conjunto de servicios de objetos comunes. Asimismo, el marco intenta reutilizar los planteamientos CORBA y los parámetros de diseño siempre que concuerden. Finalmente, aunque es importante reutilizar los modelos existentes, también es importante que el marco soporte el desarrollo de nuevos modelos. Este marco no precisa el desarrollo de un modelo GDMO antes del desarrollo de un modelo de IDL. De hecho, desarrollar un nuevo modelo de IDL para su utilización en este marco resulta inmediato y se han proporcionado directrices para hacerlo.

1.2 Aplicación

Puesto que CORBA se introduce en la RGT, son posibles diferentes opciones que varían entre la utilización de pasarelas que realizan traslaciones entre sistemas que utilizan diferentes protocolos de gestión de red a casos en los que CORBA está soportado desde un principio por los sistemas en comunicación. La aplicación de este marco está destinada a casos en los que tanto el sistema gestionado como el sistema de gestión proporcionan interfaces CORBA.

El marco no considera otros casos de interfuncionamiento que precisen sistemas "pasarela" en los que sean necesarias conversiones del protocolo y del modelo de información para lograr la interoperabilidad. En particular, este marco no está diseñado específicamente para soportar la construcción de pasarelas entre aplicaciones de gestión de red CORBA y CMIP, incluso aunque la semántica de modelos existentes se mantenga en este marco. Sin embargo, un sistema de gestión podría tener que soportar múltiples protocolos para interfuncionar en diferentes entornos.

Ya se ha desarrollado un planteamiento de pasarela que ha sido normalizado por el grupo conjunto de gestión entre dominios (JIDM). Este planteamiento de pasarela proporciona una correspondencia uno a uno de todos los constructivos y capacidades disponibles con el CMIP y las GDMO. Sin embargo, muchos servicios y capacidades CORBA no se reutilizan con este planteamiento puesto que el problema resuelto consiste en facilitar el interfuncionamiento con sistemas que se han desarrollado utilizando el CMIP. En cambio, el entorno problemático para la aplicación de este marco consiste en soportar interfaces de gestión de red CORBA originales basados en normas. Este planteamiento aprovecha los beneficios que ofrece CORBA al ser una tecnología utilizada por múltiples industrias.

La Recomendación UIT-T X.780 [1] acompaña a esta Recomendación y define las directrices de modelado de objetos, superclases para todos los objetos gestionados y para factorías de objetos gestionados para su utilización con este marco y un conjunto normalizado de notificaciones. Juntas la Recomendación X.780 y la presente Recomendación definen un *marco* para interfaces de la RGT basadas en CORBA. Asimismo, UIT-T M.3120 proporcionar una versión de IDL CORBA del modelo de información de red genérico definido originalmente en UIT-T M.3100. La versión de IDL sigue las directrices de modelado de objetos de UIT-T X.780 y está diseñada para ajustarse a los servicios definidos aquí.

1.3 Estructura de la Recomendación

Esta Recomendación tiene la estructura siguiente:

Cláusula 1	Introducción, estructura de la Recomendación, actualizaciones y lista de ediciones
Cláusula 2	Referencias
Cláusula 3	Definiciones de términos y abreviaturas utilizadas en el resto de la Recomendación
Cláusula 4	Requisitos para los servicios de la RGT basadas en CORBA. Se trata de los objetivos de diseño que tiene que cumplir
Cláusula 5	Requisitos de la versión de servicio de la ORB CORBA. También se proporciona una visión de conjunto de los servicios
Cláusula 6	Requisitos para la utilización de servicios de objetos comunes CORBA para interfaces de gestión de red
Cláusula 7	Definición de los servicios de soporte específicos de la RGT. Las interfaces de IDL para los servicios de soporte se definen en el anexo A
Cláusula 8	Directrices para el cumplimiento y la conformidad
Anexo A	Servicios soporte del marco IDL
Apéndice I	Casos de interfuncionamiento entre modelos que utilizan el marco UIT y los modelos que cumplen ADSL/ATMF.
Apéndice II	Filtrado de eventos estructurados originales y trasladados
Apéndice III	Bibliografía.

1.4 Convenios del documento

Se incluyen algunos convenios en esta Recomendación para que el lector sea consciente del objeto del texto. Aunque la mayor parte de la Recomendación es normativa, párrafos que establecen sucintamente requisitos obligatorios que debe cumplir el sistema de gestión (que gestiona y/o gestionado) están precedidos de una "R" en negrita entre paréntesis, seguida de un nombre corto que indica el sujeto del requisito, y de un número. Por ejemplo:

(R) EJEMPLO-1 Ejemplo de requisito obligatorio.

Los requisitos que pueden ser implementados optativamente por un sistema de gestión están precedidos por una "O" en lugar de una "R". Por ejemplo:

(O) OPCIÓN-1 Ejemplo de requisito opcional.

Las declaraciones de requisitos se utilizan para crear perfiles de cumplimiento y conformidad.

En esta Recomendación se incluyen muchos ejemplos de IDL CORBA y de IDL que especifican los servicios específicos de la RGT y que soportan tipos de datos, incluidos en un anexo normativo. Un IDL se escribe en un carácter "courier" de 9 puntos:

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

En la subcláusula siguiente se presentan instrucciones para extraer el IDL de una versión electrónica de esta Recomendación y compilarla.

1.5 Compilación del IDL

Una ventaja de utilizar el IDL para especificar interfaces de gestión de redes es que el IDL puede ser "compilado" en un código de programación por las herramientas que acompañan a una ORB. Se automatiza así realmente el desarrollo de alguno de los códigos necesarios para el interfuncionamiento de aplicaciones de gestión de red. Esta Recomendación tiene un anexo que contiene el código que los implementadores desearon extraer y compilar. El anexo A es normativo y debe ser utilizado por los desarrolladores que implementan sistemas conformes a esta norma. El IDL en esta Recomendación ha sido comprobado con dos compiladores para asegurar su corrección. Debe utilizarse un compilador que soporte la versión CORBA especificada en 5.2.

El anexo A se ha presentado en un formato que lo haga fácil de cortar y pegar en un fichero de texto en lenguaje claro que pueda luego ser compilado. A continuación se dan informaciones sobre el modo de conseguirlo.

1) Cortar y pegar parece funcionar mejor a partir de la versión Microsoft® Word® de esta Recomendación. Cortar y pegar a partir del formato del fichero Adobe® Acrobat® parece incluir más encabezamientos de página y pies de página, que no pueden ser compilados.
2) Todo el anexo A, que comienza por la línea "/* This IDL code..." hasta el final debe almacenarse en un fichero llamado "itut_q816.idl" en un directorio en el que será encontrado por el compilador de IDL.
3) No es necesario suprimir los encabezamientos incorporados en el anexo A. Han sido encapsulados en comentarios IDL y serán ignorados por el compilador.
4) Los comentarios que empiezan por la secuencia especial "/**" son reconocidos por los compiladores que convierten IDL en HTML. Estos comentarios a menudo tienen instrucciones de formato especiales para estos computadores. Quienes trabajen con el IDL pueden desear generar HTML ya que los ficheros HTML resultantes tienen enlaces que permiten la navegación rápida a través de los ficheros.
5) Al anexo A se le ha dado un formato con espacios de tabulación a intervalos de 8 espacios y cambios de renglón fijos que deben permitir a cualquier editor de textos trabajar con el texto.

2 Referencias

Las siguientes Recomendaciones del UIT-T y otras referencias contienen disposiciones que, mediante su referencia en este texto, constituyen disposiciones de la presente Recomendación. Al efectuar esta publicación, estaban en vigor las ediciones indicadas. Todas las Recomendaciones y otras referencias son objeto de revisiones por lo que se preconiza que los usuarios de esta Recomendación investiguen la posibilidad de aplicar las ediciones mas recientes de las Recomendaciones y otras referencias citadas a continuación. Se publica periódicamente una lista de las Recomendaciones UIT-T actualmente vigentes.

[1] UIT-X.780 (2001), *Directrices de la red de gestión de las telecomunicaciones para definir objetos gestionados de la arquitectura de negociación de petición de objetos comunes.*

[2] OMG Document formal/1999-10-07, *The Common Object Request Broker: Architecture and Specification*, Revision 2.3.1.

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

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

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

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

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

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

[9] OMG Document formal/2000-08-01, *Interworking between CORBA and TMN systems specification*.

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

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

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

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

[14] UIT-T X.701 (1997), *Tecnología de la información – Interconexión de sistemas abiertos – Visión general de la gestión de sistemas.*

[15] UIT-T X.703 (1997), *Tecnología de la información – Arquitectura de gestión distribuida abierta*.

3 Definiciones y abreviaturas

3.1 Definiciones de UIT-T X.701

Los siguientes términos utilizados en esta Recomendación se definen en UIT-T X.701:

– clase de objetos gestionados;

– gestor;

– agente.

3.2 Definiciones de UIT-T X.703

El siguiente término utilizado en esta Recomendación se define en UIT-T X.703

– notificación.

3.3 Definiciones adicionales

3.3.1 canal de eventos: objeto soporte en un sistema gestionado con una o más interfaces que permiten a un sistema de gestión recibir notificaciones de ese sistema gestionado.

NOTA – Hay dos modelos para la entrega de notificaciones del canal de eventos (véase UIT-T X.703):

- En el modelo difusor (push) de entrega de eventos, el sistema de gestión utiliza una o más instancias de interfaz de canal de eventos para registrar una o más referencias de interfaz de sistema de gestión que puedan ser utilizadas por el canal para invocar operaciones difusoras de eventos.
- En el modelo selector (pull) de entrega de eventos, el sistema de gestión utiliza una o más referencias de interfaz de canal de eventos para invocar operaciones que retornen información de notificación.

3.4 Abreviaturas

En esta Recomendación se utilizan las siguientes siglas.

AMI Invocación de método asíncrono *(asynchronous method invocation)*

API Interfaz de programación de aplicación *(application programming interface)*

ASN.1 Notación de sintaxis abstracta uno *(abstract syntax notation one)*

ATM Modo de transferencia asíncrono *(asynchronous transfer mode)*

AVA Aserción de valor de atributos *(attribute value assertion)*

CMIP Protocolo común de información de gestión *(common management information protocol)*

CORBA Arquitectura de intermediano de petición de objeto común *(common object request broker architecture)*

COS Servicios de objeto común *(common object services)*

DN Nombre distinguido *(distinguished name)*

EMS Sistema de gestión de elementos *(element management system)*

FIFO Primero en llegar, primero en salir *(first in, first out)*

GDMO Directrices para la definición de objetos gestionados *(guidelines for the definition of managed objects)*

GIOP Protocolo de interoperabilidad general *(general interoperability protocol)*

HTML Lenguaje de marcaje hipertexto *(hypertext markup language)*

ID Identificador

IDL Lenguaje de definición de interfaz *(interface definition language)*

IEEE Instituto de Ingenieros Eléctricos y Electrónicos *(Institute of Electrical and Electronics Engineers)*

IETF Grupo de tareas especiales de ingeniería de Internet *(Internet Engineering Task Force)*

IIOP Protocolo de interoperabilidad de Internet *(Internet interoperability protocol)*

IOR Referencia de objeto interoperable *(interoperable object reference)*

JIDM Gestión conjunta entre dominios *(joint inter-domain management)*

MO Objeto gestionado *(managed object)*

MOO Operación de objetos múltiples *(multiple object operation)*

NE Elemento de red *(network element)*

NMS Sistema de gestión de red *(Network Management System)*

OAM&P Operaciones, administración, mantenimiento y provisionamiento *(operations, administration, maintenance, and provisioning)*

OID Identificador de objeto *(object identifier)*

OMG Grupo de gestión de objetos *(object management group)*

ORB Mediador de petición de objetos *(object resource broker)*

OSI Interconexión de sistemas abiertos *(open systems interconnection)*

PDU Unidad de datos de protocolo *(protocol data unit)*

PM Gestión de la calidad de funcionamiento *(performance management)*

POA Adaptador de objeto portátil *(portable object adapter)*

POP Punto de presencia *(point of presence)*

POSIX Interfaz del sistema de operación portable *(portable operating system interface)*

QoS Calidad de servicio *(quality of service)*

RDN Nombre distinguido relativo *(relative distinguished name)*

RGT Red de gestión de las telecomunicaciones

SDH Jerarquía digital síncrona *(synchronous digital hierarchy)*

SONET Red óptica síncrona *(synchronous optical network)*

SSL Capa de zocálo segura *(secure socket layer)*

TII Invocación independiente del tiempo *(time-independent invocation)*

TLS Seguridad de la capa de transporte *(transport layer security)*

TTP Punto de terminación de camino *(trail termination point)*

UID Identificador universal *(universal identifier)*

UIT-T Unión Internacional de Telecomunicaciones – Sector de Normalización de las Telecomunicaciones de la UIT

UML Lenguaje de modelado unificado *(unified modelling language)*

UTC Código horario universal *(universal time code)*

4 Objetivos y requisitos de los servicios de la RGT basados en CORBA

Esta cláusula describe los objetivos fundamentales del marco de servicios y los requisitos que ayudan a los servicios de la RGT basados en CORBA a sustentar estos objetivos. La subcláusula 4.1 introduce los objetivos del marco CORBA. La subcláusula 4.2 proporciona la terminología y los requisitos. Los requisitos de esta cláusula 4 son los requisitos que tiene que satisfacer el marco. Estas basados en las necesidades de gestión de las telecomunicaciones. Las cláusulas 5, 6, 7 y 8 describen un marco que cumple estas necesidades y definen como cumplir los requisitos de esta cláusula utilizando CORBA de una cierta manera. También se indican como requisitos las reglas de las cláusulas 5, 6, 7 y 8 para la utilización de CORBA.

4.1 Objetivos

Esta Recomendación define un marco para indicar como se deben modelar las interfaces soportadas por sistemas de gestión y elementos de red. Se identifican aquí algunos de los objetivos fundamentales:

- Interoperabilidad de aplicación;
- Utilización común de servicios de objetos comunes CORBA;
- Transparencia del modelo de aplicación.

Esta cláusula explica con más detalle estos tres objetivos.

4.1.1 Interoperabilidad de aplicación

Un objetivo fundamental de la arquitectura de la RGT, y en particular de información, consiste en promover un marco normalizado para proporcionar interoperabilidad e intercambio de información entre sistemas a partir de un conjunto diverso de suministradores de sistemas de gestión de red. La interoperabilidad entre sistemas implica muchos aspectos de desarrollo. En su capa más baja, se tiene que situar un mecanismo de comunicación común para soportar una sintaxis común, el establecimiento de conectividad y el intercambio de operación petición/respuesta entre sistemas. Este aspecto de la interoperabilidad es inherente a la especificación CORBA.

Para la RGT existe la necesidad de proporcionar interoperabilidad de aplicaciones. Es decir, se utilizarán sistemas de gestión provenientes de suministradores diversos en una RGT de una única administración para soportar diversas funciones necesarias para sustentar la gestión de sus redes.

Para simplificar la integración de estos diversos sistemas de suministradores, tienen que estar de acuerdo sobre la semántica de la información que se está intercambiando. Esto se logra con la especificación de un modelo de información. Las directrices para la definición de modelos de información basados en CORBA se especifican en UIT-T X.780 y los servicios definidos aquí tiene que soportar dichas directrices.

4.1.2 Utilización común de los servicios de objetos comunes CORBA

Un segundo aspecto de este marco es la definición de la utilización común y la definición del entorno del procesamiento distribuido de la elección. Este aspecto del marco debe indicar las posibles expectativas que los suministradores de sistemas de gestión de red pueden tener entre ellos. En lugar de volver a definir las capacidades de interfaz necesarias para soportar funciones de gestión de red comunes como la denominación de objetos y el filtrado de notificación con cada modelo de información, las directrices de modelado de UIT-T X.780 se basan en un conjunto de servicios de soporte. Estos servicios de soporte permiten que los modelos de información sean más sencillos y también mejoran la interoperabilidad.

Al definir estos servicios, se realizará un esfuerzo especial para utilizar los servicios de objetos comunes CORBA, en particular esta Recomendación tratará la utilización de la ORB CORBA y de los servicios de objetos comunes (COS) CORBA que repercuten en la interoperabilidad del sistema (es decir, los asuntos que implican la utilización de CORBA en un único sistema se encuentran fuera del ámbito de esta Recomendación). Cuando las necesidades de gestión de red no se pueden cumplir mediante los COS de CORBA, se definirán servicios adicionales.

4.1.3 Transparencia del modelo de información

Si se utiliza CORBA en lugares de la arquitectura de la RGT en los que estén bien establecidos modelos de información existentes (por ejemplo, GDMO), el marco deberá soportar la reutilización de dichos modelos sin ningún cambio sustancial.

Esta Recomendación se centra en el conjunto de servicios necesarios para permitir utilizar los modelos existentes como se pretendía originalmente con una eficacia razonable.

4.2 Dependencias del modelado de la información

Como se ha descrito en las cláusulas anteriores, el modelado explícito de recursos que son gestionables a través de una interfaz básico para la interoperabilidad de aplicaciones. Las directrices para definir objetos gestionados CORBA que se detallan en UIT-T X.780 describen las normas para modelar recursos gestionables. También incluyen algunas decisiones que tienen que ser soportadas por el marco de servicios de la RGT basados en CORBA. Esta cláusula resume dichos puntos.

4.2.1 Granularidad de acceso

La *granuralidad* de interfaz CORBA se refiere a la relación entre los recursos que están modelados en una interfaz y los medios mediante los cuales se accede a ellos utilizando CORBA. La Recomendación UIT-T X.780 utiliza un planteamiento de modelado *instancia-gránulo*, que significa que cada recurso de modelado es accesible utilizando una única referencia de objeto CORBA, conocida como una *referencia de objeto interoperable* (IOR). Los objetos que representan recursos gestionables se denominan *objetos gestionados*.

4.2.2 Representación de contenencia y denominación

Contenencia es una representación lógica de cómo los recursos modelados contienen otros recursos modelados. La contenencia ha sido tradicionalmente una relación muy importante en las aplicaciones de gestión de red puesto que es un medio conveniente para identificar el gran número de recursos que tienen que ser gestionados normalmente. Las directrices de UIT-T X.780 requieren que se asigne un único nombre a cada objeto gestionado, basado en parte en el nombre del objeto que lo contiene. Los servicios de la RGT basado en CORBA tienen que proporcionar un medio para almacenar estos

nombres (y por tanto las relaciones de contenencia que representan) así como un medio para encontrar la IOR de un objeto basada en su nombre.

4.2.3 Creación y supresión de objetos

La ORB CORBA no proporciona clientes para crear objetos en sistemas distantes. En su lugar, normalmente sistemas remotos instancian objetos fabricados, y estos objetos fabricados proporcionan operaciones que pueden ser invocadas por clientes para crear objetos en el sistema remoto. Para algunos propósitos de modelado de información, UIT-T X.780 especifica que se debe definir una interfaz IDL fabricada para cada interfaz IDL de objeto gestionado en un modelo de información. De esta forma, la creación de objetos dependerá del modelo y no constituye un buen candidato para el servicio CORBA de la RGT. Sin embargo, en esta Recomendación se define un servicio para sistemas de gestión que buscan una factoría con el fin de solicitar la creación de un objeto en un sistema gestionado.

La supresión de objetos también es un asunto que precisa soporte. A menudo, se suprimen objetos CORBA sencillamente invocando alguna operación de supresión de un objeto, pero esto no es un buen planteamiento para aplicaciones de gestión de red, debido a su dependencia de las relaciones de contenencia. Suprimir un objeto que contiene otros objetos tiene consecuencias más allá del objeto que se está suprimiendo. Asimismo, como se ha descrito en la cláusula anterior, se precisa soporte para almacenar los nombres de instancias de objetos gestionados y estos datos tienen que ser actualizados cuando se suprimen objetos. Los servicios CORBA de la RGT por tanto necesitan proporcionar soporte para suprimir objetos gestionados de una forma ordenada.

4.3 Delimitación y filtrado

La capacidad para realizar peticiones complejas (es decir, operaciones GET), actualizaciones (es decir, SET), y suprimir operaciones en un grupo de entidades con una única petición de operación es un componente valioso de la RGT. Los sistemas de gestión pueden tener que gestionar hasta 10^7 instancias de objetos gestionados. Debido al tamaño de la base de datos de información de gestión, un sistema de gestión no puede realizar eficazmente peticiones específicas para instancias individuales de objetos gestionados (es decir, entidades). En su lugar, el sistema de gestión espera que el sistema gestionado soporte cierto nivel de inteligencia.

La inteligencia en el sistema gestionado permite al sistema de gestión seleccionar un grupo de entidades gestionadas en las que se realizará alguna operación. La selección de la entidad gestionada comprende dos fases: la delimitación y el filtrado. Este proceso de selección de entidades gestionadas está soportado por un servicio definido más adelante en esta Recomendación. Este servicio permite a un sistema de gestión seleccionar un ámbito de objetos sobre los que actuar (el ámbito se define mediante relaciones de contenencia, véase 4.2.2). Una vez determinado el ámbito de entidades, la operación (especificada mediante el ámbito y la petición filtrada) se realiza únicamente en aquellas entidades que cumplen los criterios definidos por un filtro.

La utilización de delimitación y filtrado en este marco soporta:

– Obtención delimitada y filtrada: Devuelve los valores (para una lista de atributos) desde cada una de las entidades que cumplen los criterios de ámbito y de filtro.

– Actualización delimitada y filtrada: Sustituye un valor de atributo o añade/suprime valores a/desde atributos set-valued, en un grupo de entidades que cumplen los criterios de ámbito y de filtro, a los valores especificados en la petición delimitada y filtrada. Se puede utilizar para actualizar uno o muchos atributos en un único objeto o en muchos objetos.

– Supresión delimitada y filtrada: Suprime todas las entidades que cumplen los criterios de ámbito y de filtro.

4.3.1 Delimitación

Delimitación implica la identificación de las entidades a las que debe aplicarse un filtro. Delimitación se aplica basándose en la jerarquía de contenencia definida en 4.2.2. El ámbito se aplica para alguna entidad básica gestionada hasta cierta profundidad en el árbol de contenencia.

La entidad básica para el ámbito se define como la raíz del árbol de contenencia a partir del cual comienza la búsqueda. Una petición delimitada tiene que especificar la entidad básica gestionada del ámbito. La profundidad del nivel de delimitación se puede entonces especificar en una de las cuatro formas siguientes en la petición delimitada:

1) la entidad básica;
2) los subordinados de nivel enésimo de la entidad básica;
3) la entidad básica y todas sus subordinadas hasta el nivel enésimo incluido;
4) la entidad básica y todas sus subordinadas (es decir, el subárbol completo).

4.3.2 Filtrado

Los filtros permiten la especificación de criterios que tienen que cumplir las entidades con el fin de que se pueda realizar una operación de gestión. Junto a la delimitación, el filtrado permite realizar una única operación a través de múltiples objetos gestionados con una única petición de operación.

Un parámetro filtro se utiliza para determinar si se tiene que realizar o no una operación en un objeto gestionado. Un parámetro filtro aplica una prueba que puede ser satisfecha o no por un determinado objeto gestionado. El filtro se expresa en términos de aserciones sobre la presencia o el valor de ciertos atributos en el objeto gestionado y se satisface si y sólo si se evalúa como VERDADERO.

4.3.2.1 Reglas de concordancia de atributos

Se definen las siguientes reglas de concordancia que se pueden utilizar en aserciones de valor de atributo (AVA). Estas reglas son:

– **Igualdad**: Se evalúa como VERDADERO si y sólo si el valor suministrado en la AVA es igual al valor del atributo.

  Para atributos valorados FIJACIÓN, la AVA se evalúa como VERDADERO si y sólo si el conjunto de miembros suministrado en la AVA es igual al conjunto de miembros en el atributo.

– **Mayor o igual**: Se evalúa como VERDADERO si y sólo si el valor suministrado en la AVA es mayor que o igual al valor del atributo.

  Para atributos valorados FIJACIÓN, el valor en la AVA tiene que tener exactamente un miembro. La AVA lo evalúa como VERDADERO si y solo si dicho miembro es superior o igual al último de los miembros en el valor del atributo.

– **Menor o igual**: Se evalúa como VERDADERO si y sólo si el valor suministrado en la AVA es inferior o igual al valor del atributo.

  Para atributos valorados FIJACIÓN, el valor en la AVA tiene que contener exactamente un miembro. La AVA lo evalúa como VERDADERO si y sólo si dicho miembro es inferior o igual al último de los miembros en el valor del atributo.

– **Presente**: Se evalúa como VERDADERO si y sólo si está presente este tipo de atributo en el objeto gestionado.

– **Subcadenas**: Se evalúa como VERDADERO si y sólo si todas las cadenas especificadas en la AVA aparecen en el atributo en el orden definido sin superposiciones y separadas de los extremos del valor del atributo y entre ellas por cero o más elementos de cadena. Además, para que la AVA lo evalúe como VERDADERO:

- el primer elemento en la subcadena inicial, si está presente, deberá concordar con el primer elemento en el valor del atributo;
- las demás subcadena, si están presentes, deben aparecer en el valor del atributo en el orden en el que las subcadenass aparecen en la AVA;
- el último elemento en la última subcadenas, si está presente, debe concordar con el último elemento en el valor del atributo.

Para atributos valorados FIJACIÓN, cada valor en la AVA tiene que contener exactamente un miembro. La AVA lo evalúa como VERDADERO si y sólo si existe por lo menos uno de los miembros del valor de atributo en el que todas las subcadenas suministradas en la AVA aparecen como se ha descrito anteriormente.

(Las tres pruebas de concordancia restantes sólo aplican a atributos valorados FIJACIÓN).

– **subconjunto de**: Se evalúa como VERDADERO si y sólo si todo los miembros acertados está presentes en el atributo.

– **superconjunto de**: Se evalúa como VERDADERO si y sólo si todo los miembros del atributo están presentes en la aserción del valor del atributo.

– **intersección no nula de conjuntos**: Se evalúa como VERDADERO si y sólo si por lo menos uno de los miembros acertados está presente en el atributo.

4.4 Notificaciones

El marco necesita soportar la capacidad de:

- entregar notificaciones;
- subscribir tipos de notificaciones;
- emitir notificaciones a múltiples destinos;
- filtrar notificaciones;
- identificar unívocamente el recurso que emitió la notificación.

El marco tiene también que soportar los requisitos sobre el contenido de las notificaciones, la liberación y los algoritmos de correlación que figuran en UIT-T X.733 y UIT-T Q.821.

5 Visión de conjunto del marco y requisitos de protocolo

La cláusula anterior destacó las funciones de gestión de red que tiene que soportar el marco. Esta cláusula y el resto de la Recomendación incluyen los detalles de cómo proporcionarán estas funciones los servicios de la RGT basados en CORBA. Los aspectos del marco relativos a objetos de modelado se incluyen en UIT-T X.780. El primer lugar, se presenta una breve visión de conjunto del marco y posteriormente se definen algunos requisitos de protocolo básicos.

5.1 Visión de conjunto del marco

Este marco para interfaces de la RGT basadas en CORBA es una colección de capacidades. Una pieza central del marco es un conjunto de servicios de objetos comunes CORBA. Este marco define su función en las interfaces de gestión de red y define convenios para su utilización. El marco define también servicios de soporte que no se han normalizado como servicios de objetos comunes CORBA, pero que se espera que sean normas en interfaces de gestión de red que conforman este marco. En el anexo A se definen posteriormente las interfaces del IDL para estos servicios.

Para soportar los objeto de soporte lógico que representan recursos gestionables, el marco necesita que implementen algunas capacidades básicas comunes, Por tanto, en UIT-T X.780 se definen dos clases básicas para su utilización en el modelado de recursos de gestión de red. Las clases de objetos gestionados (o clases de objetos) en modelos de información tienen que heredar e implementar un

conjunto básico de capacidades a partir de estas clases básicas con el fin de operar en este marco. Finalmente, se definen algunas reglas y convenios para modeladores de información que desarrollen modelos para su utilización en este marco. Estas consisten en directrices de modelado, reglas para convertir modelos GDMO para CMIP en definiciones IDL de CORBA y en idiomas de estilo IDL. Todos estos se describen gráficamente en la figura 1.

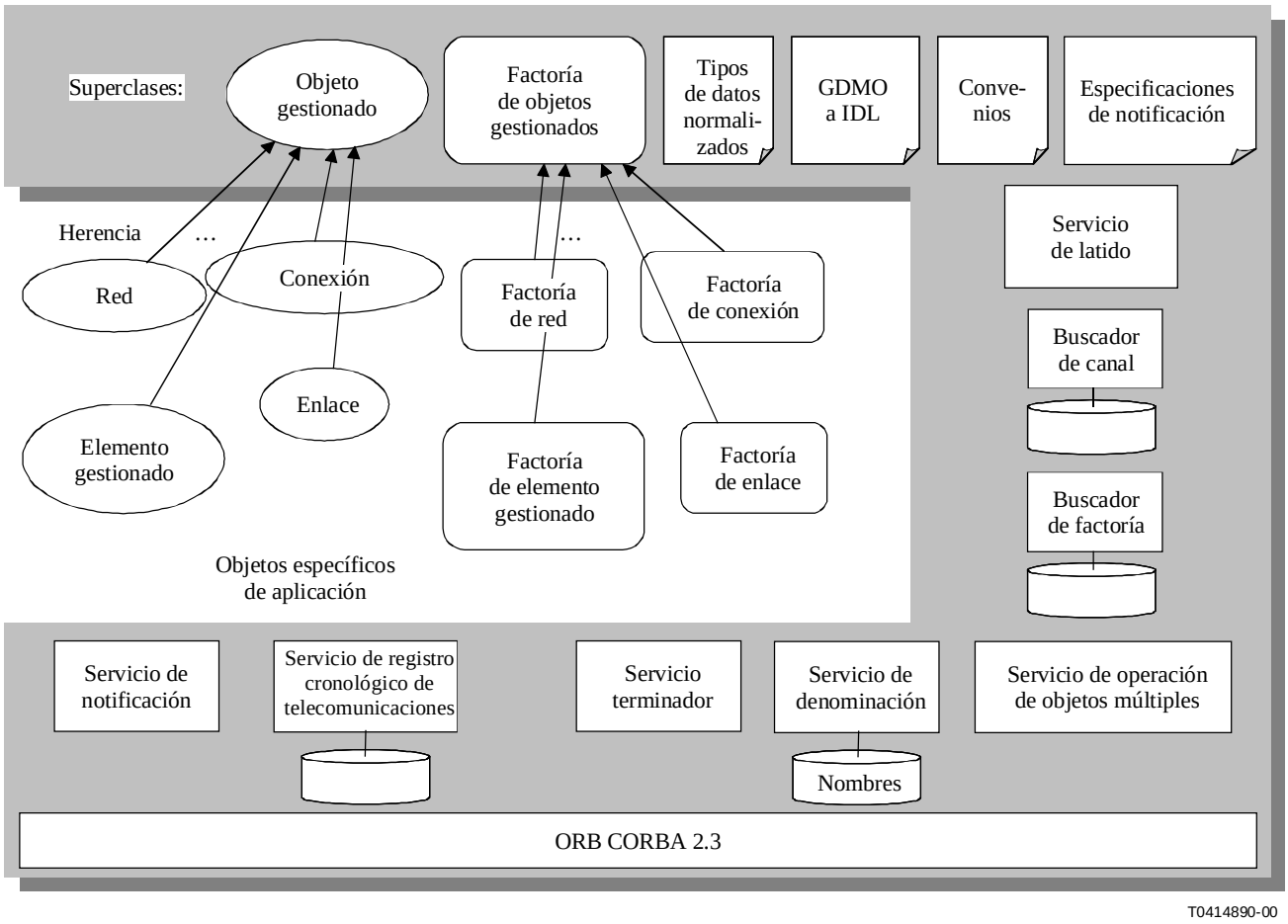

Figura 1/Q.816 – Visión de conjunto del marco

La figura 1 muestra el marco en gris. En el centro están los objetos específicos de aplicación que están soportados por el marco. En la parte inferior se encuentra una casilla que representa la ORB CORBA. En la parte superior se encuentran algunas casillas con nombres que representan los servicios que componen el marco. (Algunas también tienen iconos que describen las bases de datos que tendrían que mantener para realizar sus funciones.) A lo largo de la parte superior de la figura se encuentran iconos que representan dos superclases, una para objetos gestionados y otra para factorías de objetos gestionados. Cada uno de los objetos gestionados y de las factorías de objetos gestionados soportadas por este marco tiene que provenir de estas superclases respectivamente. También se muestran en la figura 1 iconos de páginas con esquinas dobladas que representan convenios de modelado de objetos normalizados.

Los servicios del marco, representados como casillas con esquinas cuadradas, se definen en la presente Recomendación. Las superclases, modificaciones y convenios de modelado de objetos se definen en UIT-T X.780.

5.2 Requisitos de protocolo del marco

Esta cláusula define las versiones de los servicios que son necesarios para soportar este marco. Los servicios CORBA y las especificaciones de protocolo están definidas por el grupo de gestión de objetos (OMG). El cuadro 1 muestra qué versión de la especificación OMG aplicable tiene que ser soportada para cumplir con este marco e indica la cláusula en la que se definen los requisitos detallados para el servicio. Una versión posterior de un servicio que incluya todas las capacidades requeridas de la versión indicada cumple con este marco.

Cuadro 1/Q.816 – Versiones de servicio CORBA

Servicio	Versión	Cláusula
ORB	2.3.1 [2]	5.2
Servicio de denominación	1.0 [3]	6.1
Servicio de notificación	1.0 [4]	6.2
Servicio de registro de telecomunicaciones	1.0 [5]	6.3
Mensajería asíncrona	(determinada por el sistema cliente)	6.4
Seguridad (si es necesario)	"Protocolo IOP seguro" o "interoperabilidad SSL de seguridad CORBA", como se define en [6]	6.5
Servicio de transacción	1.1 [7]	6.6

La elección de la versión 2.3.1 para las capacidades ORB básicas es importante. CORBA 2.3 incluye el soporte para el adaptador de objeto portátil (POA) así como el paso de objetos por valor. El POA es importante para el marco porque permite que implementaciones basadas en este marco alcancen hasta 10 millones de objetos instanciados, una magnitud necesaria para aplicaciones de gestión de red. El marco también utiliza herencia de tipo de valor (que soporta polimorfismo), para mantener flexibilidad y reducir la utilización de los tipos "any" de CORBA, que pueden ser ineficaces y tediosos para los programadores.

Los servicios de denominación, notificación y registro son todas las versiones iniciales disponibles a partir del OMG.

La mensajería asíncrona realmente es sólo una consideración del cliente. Una ORB con capacidades de mensajería síncrona permite a un cliente utilizar interfaces CORBA síncronas (aquellos que normalmente bloquearían al cliente) de forma asíncrona. Esta capacidad es fundamental para clientes con una única conexión que no pueden quedarse bloqueados durante operaciones de gestión de red. La disponibilidad de capacidades de mensajería asíncrona es importante para esta red porque la libera de tener que definir interfaces tanto síncronas como asíncronas. Los clientes no necesitan utilizar una ORB con mensajería asíncrona si tienen múltiples conexiones por lo que pueden admitir un bloqueo durante llamadas CORBA síncronas.

Si se precisa seguridad, este marco permite la utilización de ORB que utilizan SSL 3.0 para seguridad hasta que estén disponibles productos que soporten TLS y el OMG conlleva la seguridad CORBA. Hasta que la especificación de servicio de seguridad OMG CORBA haga referencia a la TLS, la elección de cual está soportada en el producto (si existe) tendrá que ser negociada entre suministradores y usuarios individuales. Así, hasta ahora, la utilización de uno (o ninguno) cumple con este marco.

6 Requisitos de servicios de objetos comunes del marco

La ORB CORBA proporciona capacidades de interacción de objeto a objeto [2]. Otras capacidades se definen como "servicios de objetos comunes" separados. Los servicios de objetos comunes CORBA son servicios de propósito general independientes del dominio que son fundamentales para desarrollar aplicaciones CORBA compuestas por objetos distribuidos. También proporciona los bloques de construcción para la interoperabilidad de aplicación. Los servicios se definen con interfaces de objeto y se pueden combinar de muchas formas diferentes y pueden servir para muchos usos en aplicaciones diferentes. En un dominio específico, los servicios de objetos comunes CORBA se pueden utilizar para construir facilidades de alto nivel y marcos de objetos que pueden interoperar a través de múltiples entornos de plataforma.

Muchos de estos servicios de objetos comunes CORBA ya se han implementado y están disponibles como productos de soporte lógico comerciales. Asimismo, es probable que programadores que trabajen para muchas industrias tengan experiencia con ellos en el próximo futuro. La reutilización de estos servicios de objetos comunes, en lugar de la definición de nuevos servicios estrictamente para la industria de telecomunicaciones, o la reimplantación de la funcionalidad en códigos específicos de aplicación lograrán una adopción más rápida y más económica de CORBA para la gestión de red.

Las cláusulas siguientes especifican requisitos para la utilización de servicios de objetos comunes CORBA con el fin de asegurar la interoperabilidad entre diferentes sistemas de gestión de red y para preservar el contexto de las telecomunicaciones.

6.1 Servicio de denominación

El servicio de denominación OMG es el servicio directorio de CORBA, o "páginas blancas" [3]. Esto permite al cliente construir una asociación nombre a objeto denominada *vinculación de nombre* que otros clientes pueden utilizar posteriormente para encontrar el objeto. (Las referencias de objetos CORBA son binarias y difíciles de utilizar por seres humanos). Una vinculación de nombre siempre está definida en relación con un *contexto de denominación*. Un contexto de denominación es un objeto que contiene un conjunto de vinculaciones de nombre en el que cada nombre es localmente único. Una vinculación de nombre es una estructura de datos que contiene dos cadenas y una referencia de objeto (dirección). La cadena *ID* es un identificador para la vinculación. Una segunda cadena, denominada "clase" (kind) también es parte de la estructura de datos. Juntas, la ID y la clase identifican unívocamente un objeto en relación con un contexto. Se pueden vincular diferentes nombres a un objeto en el mismo contexto o en un contexto diferente al mismo tiempo. El contexto de denominación también se puede vincular a un nombre en otro contexto de denominación. La vinculación de contextos en otros contextos crea un *gráfico de denominación*, un gráfico dirigido con nodos y bordes etiquetados en el que los nodos son contextos. Dado un contexto en un gráfico de denominación, una secuencia de componentes de nombre *(pares ID-clase*) pueden referenciar un objeto. Esta secuencia de estructuras, denominada *nombre compuesto*, define un trayecto en el gráfico de denominación por el que se puede navegar para resolver el nombre y encontrar el objeto.

No existe requisito para que las vinculaciones de nombre CORBA representen una relación de contenencia entre objetos, pero el concepto de contenencia es importante en la gestión de red y necesita ser comunicada a través de las interfaces de gestión de red. El servicio de denominación CORBA es la mejor manera para lograrlo. Los párrafos siguientes definen una serie de requisitos de cómo utilizar el servicio de denominación CORBA para representar las relaciones de contenencia a lo largo de instancias de objetos gestionados.

(R) NAME-1 Todo objeto gestionado tiene que tener un y sólo un nombre (DN). Los componentes del nombre de puede obtener a partir de múltiples servidores federados. Aunque el servicio de denominación OMG soporta múltiples nombres por objeto, este marco restringe al objeto gestionado a que utilice un único nombre. El soporte de múltiples nombres está fuera del ámbito de este marco.

(R) NAME-2 Puesto que una vinculación de nombre simple no puede identificar un objeto ni tampoco objetos contenidos, cada objeto gestionado tiene que tener realmente el correspondiente contexto de denominación. Una vinculación especialmente denominada en cada uno de estos contextos vinculará al valor *ID* "objeto" con una referencia del objeto gestionado real. (El campo *kind* de esta vinculación será nulo.) También se pueden vincular a nombres en este contexto otros contextos de denominación que representan objetos gestionados contenidos.

(R) NAME-3 El campo *ID* de una vinculación en nombre para un contexto de denominación que representa a un objeto gestionando dependerá de la aplicación y puede tener realmente valor semántico en lugar de identificar únicamente un objeto gestionado, para una determinada clase de objetos. Por ejemplo, el valor *ID* de "7" para un objeto de tenedor de equipo que representa un hueco en una estantería puede indicar que este objeto representa el séptimo hueco de la estantería. Los valores semánticos especiales unidos a los ID se documentarán para cada clase de objeto gestionado como parte de la especificación de interfaz de objeto gestionado. Hay que destacar que el campo *ID* es una cadena.

(R) NAME-4 El campo *kind* de una vinculación de nombre para un contexto de denominación que representa a un objeto gestionado se determinará mediante *información de vinculación de nombres de objetos gestionados*. Se trata de información definida como constantes en módulos IDL, específicamente para fines de representación de posibles relaciones de contenencia. Para más detalles, véase UIT-T X.780 sobre la representación de información de vinculación de nombres de objetos gestionados. Sin embargo, un módulo de vinculación de nombre contendrá una cadena constante denominada "clase" no delimitada del objeto gestionado que se utilizará como un valor para el campo kind en vinculaciones de nombres CORBA. El valor de esta cadena será normalmente el nombre de clase del objeto gestionado. Esto añade valor al hacer más sencilla la identificación del tipo de un objeto y al reducir la posibilidad de colisiones entre nombres. Un factor que complica esto es la aparición de nuevas versiones de un objeto, por ejemplo, un *equipmentR1* que amplia un objeto *equipment*. Cuando la nueva clase únicamente amplia las capacidades de una clase existente sin cambiar su función (es decir, sigue representando el mismo recurso gestionado), el campo kind será normalmente el nombre de clase básica original. Esto, sin embargo, depende en última instancia del modelador de objetos que define el módulo IDL de vinculación de nombres. La utilización del valor original permitirá que aplicaciones existentes continúen utilizando la nueva clase como si fuera de la versión antigua.

La figura 2 muestra un ejemplo de vinculaciones de nombre de conformidad con los requisitos anteriores. En esta figura, se representan los contextos de denominación CORBA como carpetas. El contenido de las carpetas son vinculaciones de nombres. Un componente de nombre como una cadena ser representa mediante el formato <ID>.<kind> (En algunos ejemplos de vinculaciones de nombres no se muestra un puntero en el diagrama para reducir la complejidad del diagrama). El gráfico representa un objeto de red, denominado "CentralNet", que contiene un objeto de elemento gestionado denominado "Element9" y una conexión denominada "R5698".

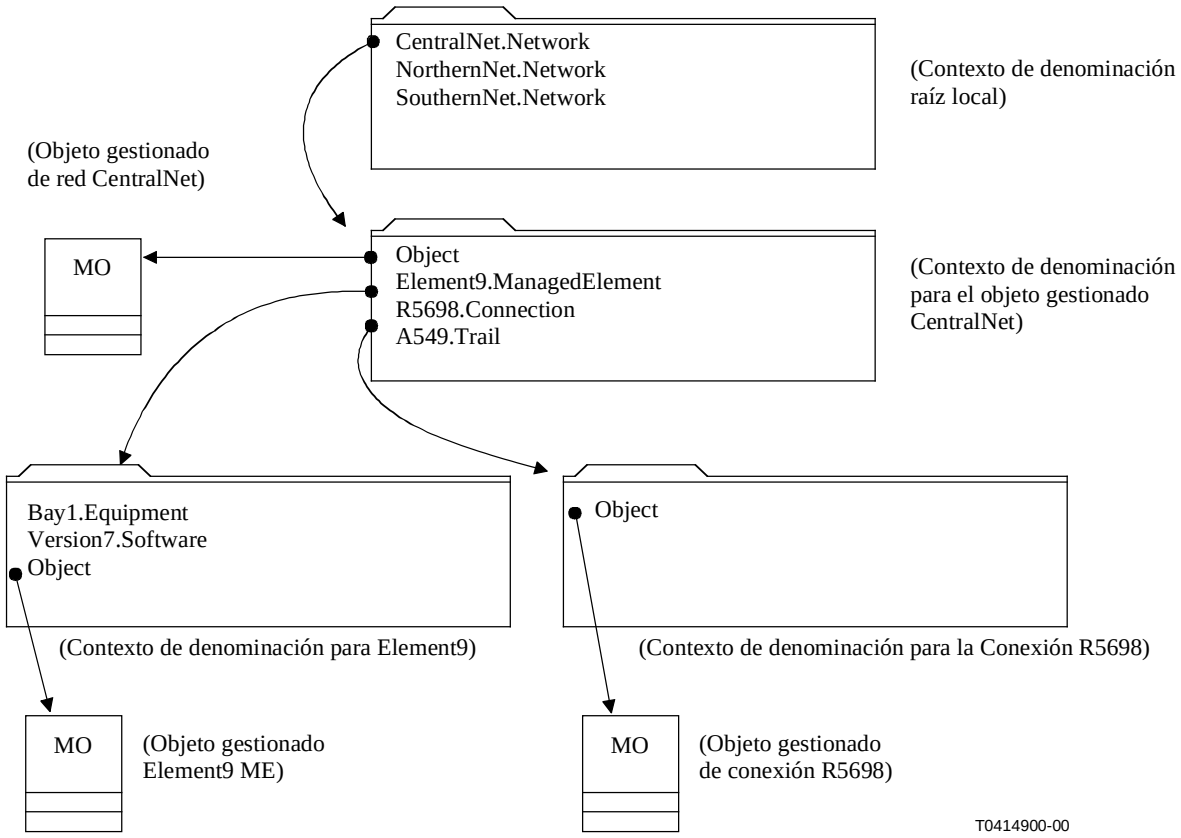

Figura 2/Q.816 – Gráfico de denominación de objetos gestionados

(R) NAME-5 Cada sistema gestionado proporcionará por lo menos un contexto de denominación de raíz local. Hay que destacar en la figura 2 que se hace referencia al contexto de denominación de la parte superior como un contexto de denominación "raíz local". Éste es un contexto de denominación en el que se vincularán nombres para los objetos gestionados más altos en el sistema, así como nombres para ciertos objetos de servicio soporte.

Un sistema gestionado puede tener múltiples contextos de denominación de raíz local. Puesto que los objetos gestionados no pueden tener múltiples nombres, pueden estar vinculados bajo una única raíz local los objetos de servicio soporte, sin embargo, pueden tener nombres vinculados en múltiples contextos de denominación raíz en el mismo sistema. Un factor a considerar cuando se determina cuantos contextos de denominación de raíz local tendrá un sistema gestionado es si existe la posibilidad de que alguno de los objetos gestionados pudiera en algún momento tener que desplazarse a otro sistema. El desplazamiento de un árbol entero de objetos gestionados, incluido el contexto de denominación de raíz local, resultará más sencillo que desplazar un subárbol de objetos.

(R) NAME-6 Un sistema gestionado proporcionará un procedimiento administrativo local para asignar un nombre CORBA a cada contexto de denominación de raíz local en el sistema. Todos los nombres intercambiados a través de la interfaz gestionada incluirán el nombre del contexto de raíz local a menos que se indique lo contrario. Esto incluye parámetros de operación y notificaciones.

Esta característica permite a una administración hacer que los nombres sean globalmente únicos. Puesto que el sistema gestionado tiene que asegurar que todos los nombres son únicos en relación con el contexto de denominación de raíz local, al asignar un nombre globalmente único al contexto de denominación de raíz local una administración puede asegurar que todos los nombres en un sistema gestionado son únicos. El mecanismo utilizado para elegir un nombre globalmente único para el contexto de raíz local se deja a la elección de la administración. El formato del nombre será el mismo que el utilizado por el servicio de denominación CORBA, *CostNaming::Name*. Se permiten

múltiples componentes, pero las administraciones querrán mantener cortos los nombres del contexto de raíz local para reducir la cabecera.

Además de hacer que los nombres sean únicos, la asignación de un nombre a un contexto de denominación de raíz local hará que sea más sencillo para un sistema de gestión definir nombres. Esto se debe a que el sistema de gestión puede vincular los contextos de denominación de raíz local para todos los sistemas que gestiona en su propio servicio de denominación local. El nombre que utiliza para esta vinculación será el mismo nombre asignado al contexto de denominación raíz en el sistema gestionado. En la figura 3 se muestra un ejemplo.

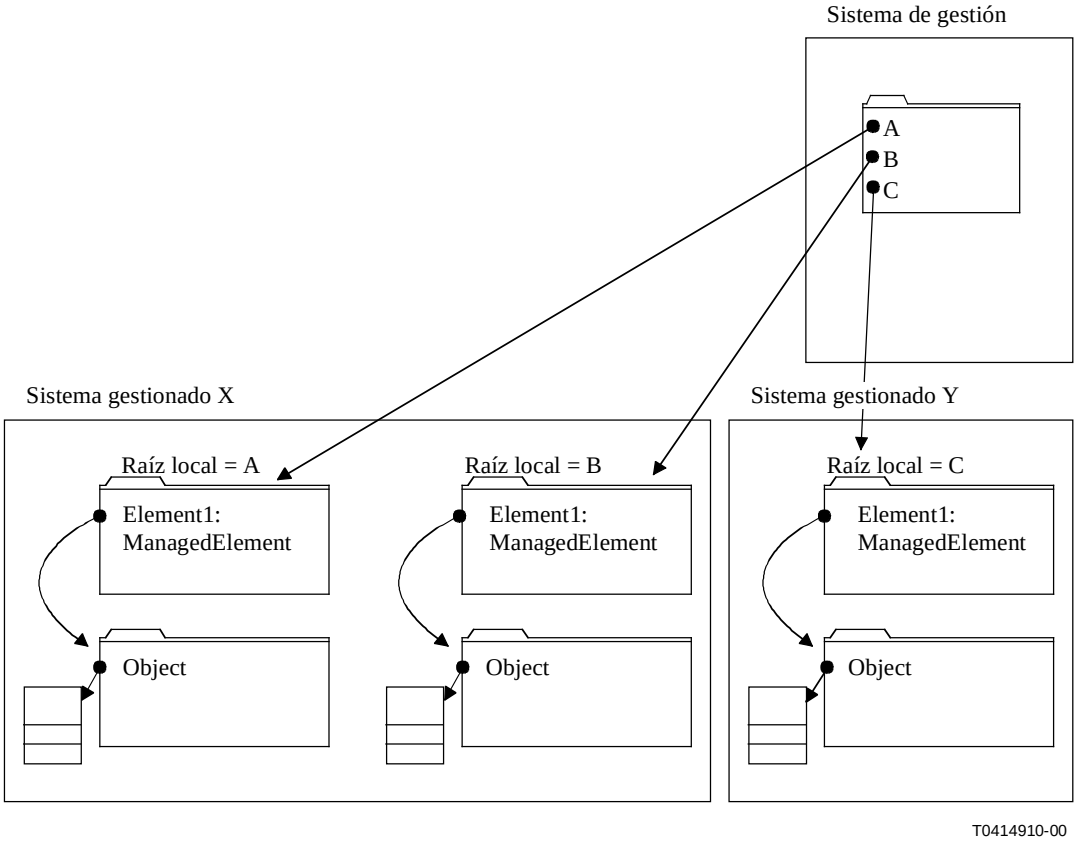

Figura 3/Q.816 – Asignación de nombres a contextos de denominación de raíz

La figura 3 muestra en la parte inferior dos sistemas de gestión de elementos. El sistema "X" tiene dos objetos del tipo *ManagedElement*, y el sistema "Y" tiene 1. Cada objeto *ManagedElement* pertenece a su propio contexto de denominación de raíz local, lo que significa que el sistema X tiene dos raíces locales y el sistema Y una. También hay un sistema de gestión de red y se han vinculado los contextos de raíz local de ambos EMS dentro de un servicio de denominación en este sistema. Esta administración ha elegido asignar los nombres únicos "A" y "B" a los contextos de raíz local en el sistema X y "C" al contexto de raíz local en el sistema Y. Las referencias a los contextos de denominación locales se han vinculado con estos nombres en el sistema de gestión de red.

Supongamos que el sistema Y emite una notificación relativa a su objeto *ManagedElement*. El nombre completo de dicho objeto (contenido en la notificación) será "C/Element1.ManagedElement". Digamos ahora que el NMS desea extraer mas datos del objeto. Para ello, tendrá que definir el nombre en una referencia de objeto CORBA. El NMS puede conseguirlo realizando simplemente una operación de definición utilizando el nombre completo del contexto local al que vinculó los contextos de raíz local EMS. Puesto que el servicio de denominación de NMS está federado con los servicios de denominación EMS, el servicio de

denominación de NMS puede enviar automáticamente la operación de definición al servicio de denominación adecuado en el EMS y devolver la referencia de objeto en la aplicación del NMS.

Está previsto que el nombre del contexto de denominación de raíz local se asignará durante la inicialización de un nuevo sistema. Una vez en funcionamiento, resultará extremadamente difícil si no imposible cambiarlo.

Una vez asignado un nombre, la referencia de objeto interoperable CORBA del contexto de raíz local deberá ser vinculado a un contexto de denominación en el sistema de gestión, puesto que hasta ahora no tiene ni idea de que exista el nuevo sistema. Esto significa que el sistema gestionado también tendrá que proporcionar un medio para acceder a la IOR "encadenada" del contexto de denominación de raíz local. Este valor será entonces transferido al sistema de gestión por algún medio distinto de la interfaz de gestión (correo electrónico, ftp, etc.). El sistema de gestión precisará una forma de aceptar esta IOR encadenada y de vincularla a un nombre en el sistema de gestión. Tan pronto como la IOR del contexto de raíz local esté vinculada a un nombre en el sistema de gestión, el sistema puede empezar a descubrir los objetos en el nuevo sistema (utilizando el servicio de operación de objetos múltiples descrito más adelante) y empezar a gestionarlos.

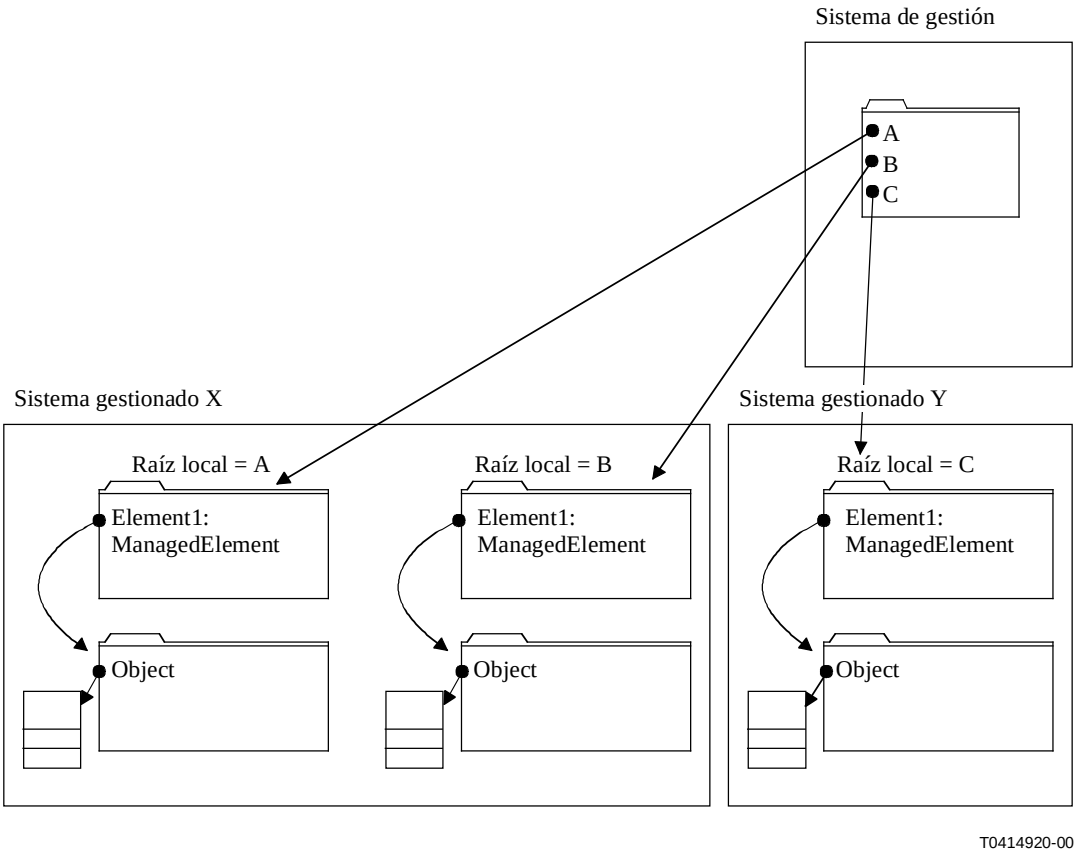

Figura 4/Q.816 – Desplazamiento de un contexto de denominación de raíz local y objetos contenidos

La figura 4 muestra como un contexto de denominación de raíz local y todos lo objetos contenidos por debajo de él se pueden desplazar a otro sistema sin cambiar los nombres de los objetos. El único cambio que pudiera precisarse sería cambiar la referencia de objeto ligada al sistema o sistemas de gestión de red. Asimismo, se debería actualizar cualquier referencia restante para desplazar objetos. El desplazamiento de sólo una parte de un árbol incluida por debajo de un contexto de denominación de raíz local precisaría volver a denominar dichos objetos.

6.1.1 Traducción de nombres de objetos gestionados a cadenas

En este marco, los nombres de objetos gestionados se representan por estructuras de datos. Sin embargo, a veces puede haber necesidad de representar nombres de objetos gestionados como cadenas. La especificación del servicio de denominación interoperable CORBA [Ref] define reglas para traducir nombres del servicio de denominación CORBA (que son utilizados por este marco) en cadenas. Los sistemas de gestión de red pueden, sin embargo, considerar necesario almacenar nombres de objetos gestionados en servidores de protocolo ligero de acceso al directorio (LDAP, lightweight directory access protocol). Esta cláusula define las reglas para traducir nombres de objetos gestionados en cadenas adecuadas para uso con el LDAP.

Un nombre de objeto gestionado es una estructura de datos que contiene secuencias de estructuras de datos con dos cadenas denominadas *kind* e ID. Los nombres de objetos gestionados se convierten en cadenas de nombre distinguido concadenando las cadenas a partir de cada una de las estructuras de datos de la secuencia, así como los signos igual ("=") y las comas (,) en el orden definido a continuación.

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

"<*kind*>$_0$ designa el valor de la cadena *kind* del primer componente de la secuencia, <*kind*>$_1$ designa el valor de la cadena *kind* del segundo componente de la secuencia, etc. Cuando se colocan en la forma de cadena DN LDAP, las cadenas *kind* e ID pueden tener que contener secuencias de escape. Véase detalles en [12].

6.2 Servicio de notificación

El servicio de notificación CORBA soporta el intercambio asíncrono de mensajes de eventos entre clientes mediante la utilización de un paradigma de suscripción y publicación [4]. El servicio de notificación introduce canales de eventos que negocian mensajes de eventos, suministradores de notificación que suministran mensajes de eventos y consumidores de notificación que consumen mensajes de eventos. El servicio de notificación CORBA mantiene toda la semántica especificada para el servicio de eventos CORBA, permitiendo compatibilidad hacia atrás con clientes de servicios de evento. La funcionalidad ampliada que importa al dominio de gestión de red es el evento estructurado y el filtrado de eventos y la QoS (Calidad de servicio). La figura 5 describe la arquitectura general del servicio de notificación.

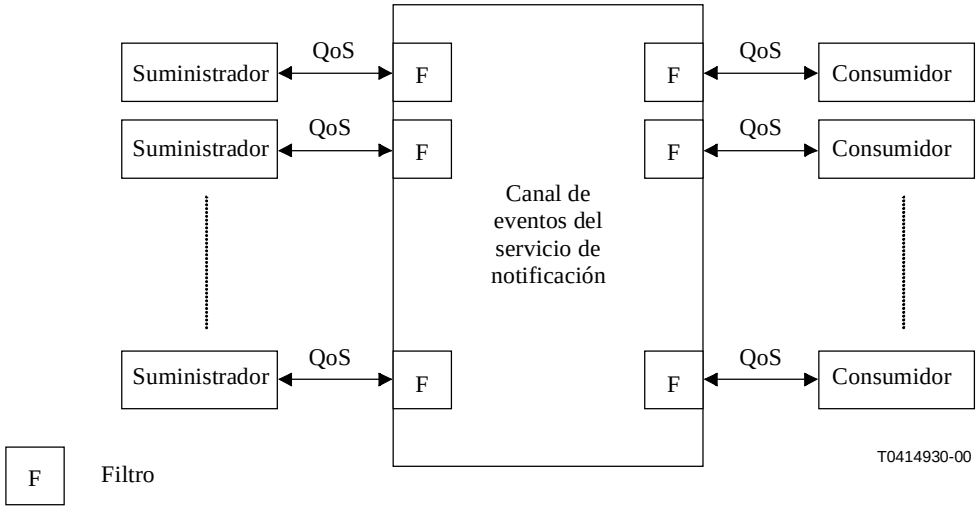

Figura 5/Q.816 – Arquitectura del servicio de notificación

(R) NOTIF-1 El servicio de notificación soportará el modelo de interfaz difusor (push). La interfaz del objeto gestionado con el canal de eventos será un suministrador difusor.

(R) NOTIF-2 El sistema gestionado instanciará el objeto u objetos de canal de eventos que utilice. Un sistema gestionado tiene que instanciar por lo menos un canal y puede instanciar más de uno. (Estos canales pueden ser canales de eventos de notificación OMG [4] o canales de eventos de registro cronológico de telecomunicación OMG [5], o un objeto que soporte la interfaz 3GPP NotificationIRPOperation interface [13].) El marco no soporta la creación o supresión de canales de eventos a través de la interfaz de gestión. Con este fin se pueden proporcionar procedimientos administrativos locales. (Los canales de eventos, sin embargo, soportan la creación y supresión de filtros a través de la interfaz de gestión.)

(R) NOTIF-3 Cada canal de eventos estará registrado con el servicio buscador de canal. El servicio buscador de canal es un servicio soporte definido por este marco en 7.2. Durante el registro, el canal estará asociado con uno o más objetos gestionados que forman cada uno de ellos la base de un ámbito de objetos gestionados que envían sus eventos al canal. Pueden estar asociados múltiples canales con el mismo objeto gestionado básico. Una utilización probable de esto consiste en tener diferentes canales para diferentes tipos de eventos. Por ejemplo, un canal podría manejar eventos de gestión de características de funcionamiento mientras otro maneja alarmas. Cuando el canal está registrado por el servicio buscador de canal también está unido a un conjunto de tipos de eventos que maneja y a un conjunto de tipos de objetos gestionados que le envíen sus eventos. Todas las notificaciones provenientes de todos los objetos gestionados tienen que ir por lo menos a un canal.

Aunque que este planteamiento es bastante flexible y permite disposiciones complejas de canales, puesto que los canales no se pueden crear a través de la interfaz de gestión, la complejidad está controlada por la implementación del sistema gestionado. Podría ser tan simple como que un único canal comprobara todos los objetos gestionados del sistema. (Hay que destacar de nuevo que mientras los canales no se pueden crear a través de la interfaz, canales individuales soportan de hecho la creación y la supresión de filtros a través de la interfaz de gestión. Por tanto, se puede registrar cualquier número de clientes para los eventos que deseen recibir.)

(R) NOTIF-4 El servicio de notificación soportará eventos estructurados.

(O) NOTIF-5 La utilización de eventos tipificados es opcional. Las secuencias de eventos estructurados se definen en [4] y se utilizan para enviar múltiples eventos en un mensaje.

(O) NOTIF-6 La utilización de eventos estructurados es opcional.

NOTA – Si el sistema gestionado soporta eventos tipificados, deberá seguir permitiendo que los sistemas de gestión reciban eventos estructurados si el gestor lo decide. Esto puede efectuarse utilizando un canal de eventos que soporte la traducción de eventos tipificados en eventos estructurados como se define en la especificación del servicio de notificación OMG [4].

La interfaz de mensajes entre suministradores y consumidores está definida en el IDL como si estuvieran utilizando eventos tipificados. Esto se realiza para permitir capturar la notificación en la IDL (que no se puede realizar por eventos estructurados salvo con comentarios) así como para soportar notificaciones tipificadas para aplicaciones que deseen utilizarlas.

Más adelante se proporcionan reglas para crear notificaciones estructuradas basadas en estas operaciones tipificadas.

La definición de servicio de notificación OMG define reglas para canales para convertir automáticamente notificaciones tipificadas en notificaciones estructuradas. Si los objetos gestionados crean originalmente notificaciones tipificadas pero el cliente desea recibir notificaciones estructuradas, el canal seguirá las reglas de traducción OMG. Sin embargo, hay que destacar que esta disposición es probablemente menos eficaz que cuando ambos sistemas utilizan sistemas tipificados. Si los objetos gestionados crean originalmente notificaciones estructuradas, deberán hacerlo de conformidad con las reglas siguientes.

Las notificaciones estructuras creadas originalmente por un sistema gestionado diferirán ligeramente de las notificaciones estructuradas creadas por conversión automática a partir de notificaciones tipificadas. Una razón para ello es hacer que sea posible que un sistema de gestión distinga la diferencia y acepte las notificaciones tipificadas si están soportadas por el sistema gestionado. Otra razón es utilizar más eficazmente las notificaciones estructuradas. Los sistemas gestionados que crean originalmente notificaciones estructuradas pueden excluir parámetros opcionales en esas notificaciones. Debido a que las notificaciones tipificadas se crean a partir de una invocación de método fuertemente tipificada, un canal de notificación comercial que la traslada a una notificación estructurada incluirá cualesquiera valores nulos como parejas nombre-valor en el cuerpo del evento estructurado antes que excluirlo. Hay que destacar que al autorizar a los sistemas gestionados que originalmente crean notificaciones estructuradas la exclusión de parámetros opcionales hace que resulte muy improbable que canales de notificación comerciales sean capaces de soportar la conversión automática de eventos estructurados en eventos tipificados.

En resumen, si los objetos gestionados crean originalmente eventos estructurados, lo harán de conformidad con las reglas siguientes. Debido a que, por eficacia, estas reglas permiten a los sistemas gestionados excluir parámetros opcionales de las notificaciones estructuradas, no se espera que soporten la conversión automática de estas notificaciones estructuradas en notificaciones tipificadas por canales de notificación comerciales. Por tanto, el sistema de gestión tiene que aceptar eventos estructurados. Si el sistema gestionado crea originalmente eventos tipificados, el sistema de gestión puede apoyarse en el canal de notificación para convertirlos automáticamente en eventos estructurados basándose en las reglas del servicio de notificación OMG. Sin embargo, las notificaciones estructuradas dependen del uso extensivo de tipos de datos CORBA "any", lo que puede resultar poco efectivo. Por lo tanto, como en este caso, el sistema de gestión preferirá probablemente aceptar notificaciones tipificadas.

(R) NOTIF-7 Los suministradores y consumidores de eventos estructurados seguirán estas reglas para construir y recibir los eventos estructurados. (Véase la figura 6 que muestra la estructura de notificación y como se debe establecer la correspondencia en ella de los elementos provenientes del IDL definición de notificación):

- La cadena `domain_type` en el encabezamiento fijo del evento estructurado se fijará "telecomunicaciones".
- La cadena `type_name` en el encabezamiento fijo del evento estructurado se fijará al nombre delimitado de la operación que define la notificación en el IDL, por ejemplo, `"itut_x780::Notifications::attributeValueChange"`.
- La cadena `event_name` en el encabezamiento fijo del evento estructurado no es utilizada por este marco.
- Se pueden incluir según proceda campos de encabezamiento opcionales para soportar características como calidad de servicio.
- Cada parámetro en la operación se situará en un par nombre-valor en la parte filtrable del cuerpo estructurado. La cadena `fd_name` de este par se fijará al nombre de parámetro y el tipo situado en el `fd_value` asociado será del tipo especificado para el parámetro. Utilizando como ejemplo la notificación `equipmentAlarm` del IDL que se presenta mas adelante en esta Recomendación, la primera cadena `fd_name` se fijará a "`eventTime`" y el primer `fd_value` contendría un tipo de datos `externalTimeType`. Aunque todos los parámetros de notificación están en el cuerpo filtrable de la estructura de notificación, dependiendo del tipo de datos del parámetro puede ser difícil o incluso imposible un filtro útil que utilice dicho parámetro. Las "reglas de concordancia" del filtro se basan en las capacidades del canal.
- Facultativamente se podrán excluir de la estructura de notificación los parámetros marcados como "opcional". Si se utilizan notificaciones tipificadas, estos parámetros está incluidos, pero normalmente tendrán un valor nulo especial si no están soportados. Para tipos para los cuales no existe un valor nulo especial (como números enteros) se define normalmente un

tipo especial que consiste en una unión entre el tipo básico (como un número entero) y el tipo nulo. Estos tipos de unión pueden ser excluidos de las notificaciones estructuradas cuando tiene un valor nulo, pero si está incluidas, se tiene que utilizar el tipo de unión. Esto permite utilizar los mismos filtros tanto para notificaciones estructuradas como tipificadas.

- El resto del cuerpo del evento estructurado (la parte no filtrable) será nula.
- En operaciones de notificación se evitarán parámetros denominados "operación" para soportar potencialmente la utilización de notificaciones tipificadas (cuando se convierten notificaciones tipificadas en notificaciones estructuradas, el canal de eventos sitúa automáticamente los parámetros de una operación en una estructura de notificación. Desgraciadamente, las reglas desarrolladas para hacer esto establecen que el nombre de la operación utilizada para emitir la operación no está en el encabezamiento del evento, sino en el cuerpo de la estructura como el primer par nombre-valor. La cadena `fd_name` se fija a "operación" y el `fd_value` se fija a una cadena que contiene el nombre de la operación. La utilización de un parámetro denominado "operación" daría entonces como resultado un segundo par nombre-valor con el nombre "operación" y se podrían confundir los dos).

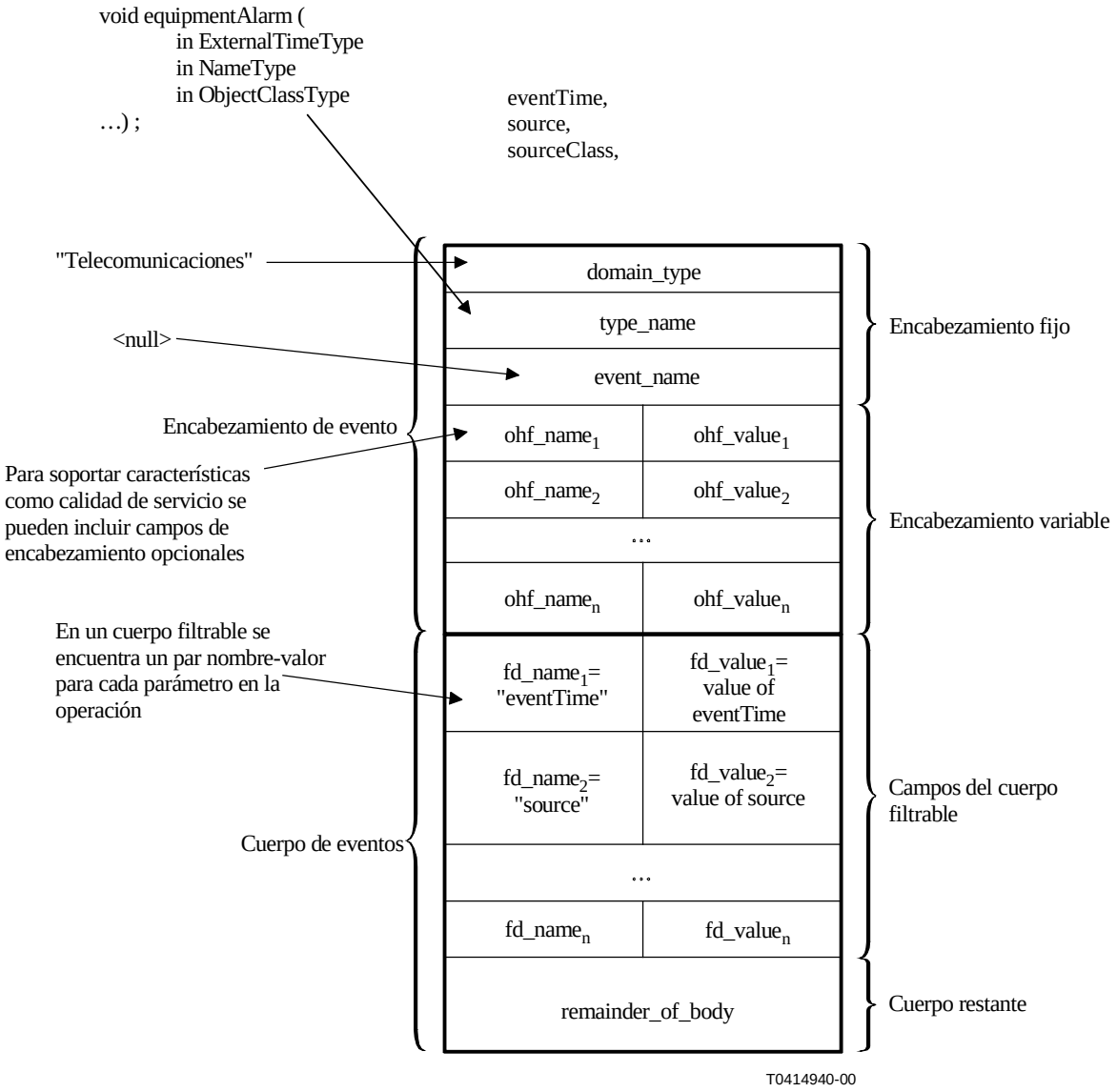

Figura 6/Q.816 – Notificaciones de correspondencia a eventos estructurados

(R) NOTIF-8 La especificación de servicio de notificación soporta expresiones de filtro que se utilizan para determinar si se debe enviar el evento. También soporta expresiones de filtro que "hacen corresponder" valores en la notificación con parámetros utilizados para influir en la operación del canal de eventos, tales como la QoS utilizada en la entrega del evento. Por ejemplo, se podría utilizar un filtro de correspondencia para hacer corresponder un campo "severity=mayor" a partir de un evento (que no significaba nada para un canal de eventos) con un parámetro QoS "priority=1" (que sí significa algo para el canal). El servicio de notificación soportará el filtrado de eventos con objetos de filtro que soporten limitaciones expresadas en la gramática de constricciones por defecto especificada por el OMG. El servicio de notificación soportará también filtros de correspondencia.

(R) NOTIF-9 La QoS de fiabilidad del servicio de notificación soportará EventReliability = Persistent y ConnectionReliability = Persistent.

Se garantiza que cada evento se entrega a todos los consumidores registrados para recibirlo en el instante en que se entregó el evento al canal, dentro de los límites de vencimiento. Si por cualquier razón se pierde la conexión entre el canal y un consumidor, el canal almacenará permanentemente cualquier evento destinado a dicho consumidor hasta que cada evento caduque debido a los límites de vencimiento o el consumidor esté de nuevo disponible y el canal por tanto sea capaz de entregar los eventos a todos los consumidores registrados. Además, desde el principio a partir de un fallo el canal de notificación restablecerá automáticamente las conexiones a todos los clientes que estaban conectados con él en el instante en que surgió el fallo [4].

(R) NOTIF-10 La QoS de política de orden del servicio de notificación permitirá que se entreguen los eventos en el orden de llegada, es decir FIFO. El servicio de notificación también puede opcionalmente soportar una QoS de prioridades de orden según la cual los eventos se podrán almacenar en memoria con un orden de prioridad, de forma que los eventos con mayor prioridad se entreguen antes que los eventos con menor prioridad.

(R) NOTIF-11 La implementación del servicio de notificación utilizada será compatible con las declaraciones de conformidad establecidas en la especificación del servicio de notificación OMG salvo para el modelo de interfaz selector (pull).

6.3 Servicio registro cronológico – Telecom

El servicio registro cronológico telecom de CORBA [5] es un servicio de registro cronológico basado en CORBA que soporta totalmente UIT-T X.735. El registro cronológico se implementa como un canal de eventos del servicio de evento o de servicio de notificación. El servicio de registro cronológico soporta la funcionalidad siguiente:

- Escribir en el registro: Los eventos suministrados al fichero se almacenan permanentemente como registros cronológicos.
- Enviar desde el fichero del registro cronológico: Los eventos suministrados al fichero también se envían a otros ficheros o a cualquier otra aplicación que desee recibirlos.
- Eventos generados en el fichero de registro cronológico: El propio fichero generará eventos.

El servicio de registro cronológico proporciona también funciones de control y gestión de ficheros de registro cronológico, manipulación de registros, gestión de ciclos de vida cronológicos. La figura 7 muestra una representación gráfica del servicio de ficheros de registro cronológico.

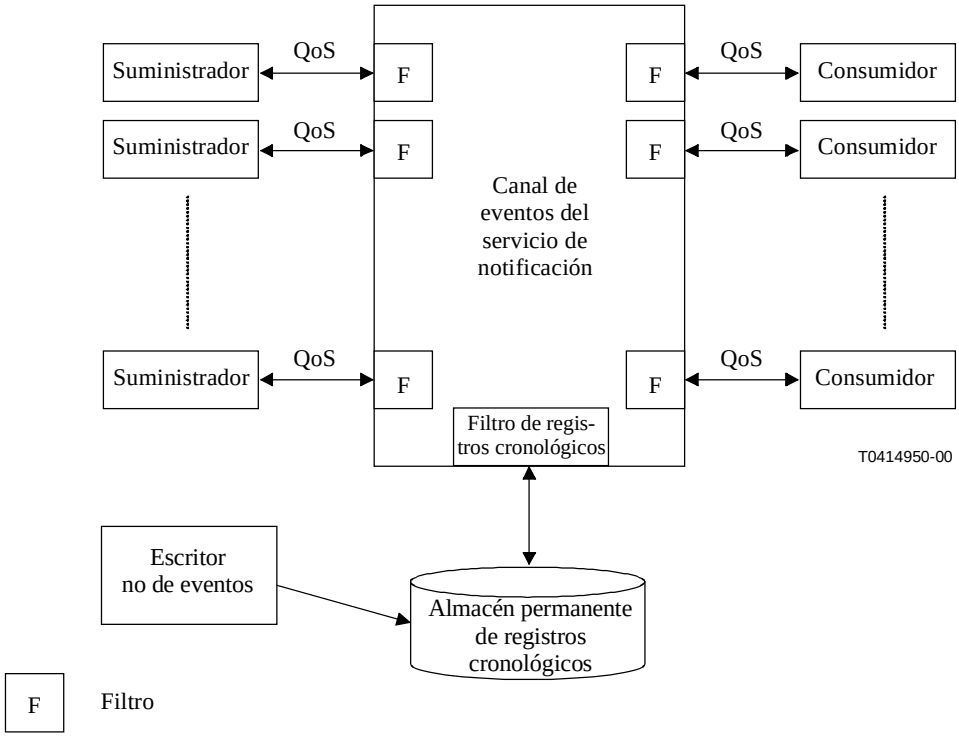

Figura 7/Q.816 – Servicio de fichero de registro cronológico de telecomunicaciones

Al manipular el filtro cronológico, un sistema de gestión puede controlar qué eventos están registrados y cuales no, exactamente de la misma manera en que es capaz de controlar qué eventos se envían y cuales no. La única excepción es el "escritor no de eventos", que es una aplicación que escribe datos directamente en el fichero de registro cronológico.

Nótese que la definición del fichero de registro cronológico de telecomunicaciones del OMG es anterior a este marco. Las notificaciones provenientes del registro telecom son estructuralmente diferentes de las otras notificaciones en el marco, incluso aunque algunos de los nombres de las notificaciones y de los parámetros sean los mismos desde el punto de vista semántico.

(R) LOG-1 El servicio de registro cronológico soportará todos los requisitos del servicio de notificación. Los canales de eventos de registro cronológico tienen que estar registrados en el servicio buscador de canal.

(R) LOG-2 El registro de registro cronológico soportado por el servicio será el `struc LogRecord` normal. Soportar `struc TypedLog Record` es facultativo.

(R) LOG-3 La implantación del servicio de registro cronológico cumplirá la declaración de conformidad en la especificación del servicio de registro cronológico telecom del OMG con la excepción del modelo de interfaz pull.

6.4 Servicio de mensajes

El servicio de mensajes CORBA considera tres campos: invocación de método asíncrono (AMI), invocación independiente del tiempo (TII) y calidad de servicio (QoS) de los mensajes [8]. De estos tres campos, la AMI tiene un papel importante en el dominio de gestión de red porque permite a los clientes hacer peticiones sin bloqueo en un objeto CORBA.

Hay que destacar que CORBA está diseñado para permitir una aplicación para invocar un método en un objeto como si el objeto fuera local para la aplicación, independientemente de la ubicación real del objeto. Normalmente, cuando un método se invoca sobre un objeto en una aplicación no

distribuida, el control pasa a dicho objeto y la rutina llamante se bloquea hasta que el método se completa y se devuelve el control. Estas semánticas se mantienen en CORBA. En una semántica distribuida, sin embargo, la latencia de la red puede llevar a un funcionamiento limitado. Se pueden considerar cinco soluciones:

1) Las aplicaciones pueden sencillamente permitir los retardos.

2) Las interfaces RGT IDL pueden en su lugar definirse como asíncronas. Es decir, las operaciones de gestión siempre se definen para no devolver resultados. Por tanto, se pueden realizar invocaciones sin bloqueo. (Esto está soportado por CORBA.) Los resultados se devuelven más tarde cuando el sistema gestionado realiza una "comunicación por intermediario" en el sistema de gestión.

3) Las interfaces RGT IDL tienen dos conjuntos de operaciones, uno que es asíncrono y el otro síncrono.

4) Las interfaces RGT IDL se definen para que sean síncronas y las aplicaciones de gestor mejoran las características al tener múltiples conexiones y ser capaces de bloquearse con peticiones múltiples y continuar procesando otro trabajo.

5) Las interfaces RGT IDL se definen para que sean síncronas y las aplicaciones de gestor mejoran las características utilizando el servicio de invocación de método asíncrono y una ORB que lo soporte.

Este marco elige una combinación de las soluciones 4 y 5. Las interfaces RGT IDL se definen para que sean síncronas. Las aplicaciones de gestor con múltiples conexiones pueden utilizar estas interfaces directamente y tener buenas características de funcionamiento. Las aplicaciones de gestor que no tienen múltiples conexiones utilizarán el servicio AMI para mejorar sus características. Puesto que los gestores con múltiples conexiones no necesitan servicio AMI, su utilización es facultativa.

La AMI es tratada sólo como un asunto de correspondencia de lenguaje en el extremo del cliente. En la mayoría de los casos no es preciso que cambien las implementaciones del extremo servidor. En algunas situaciones, tales como con un servidor transaccional, la asincronía de un cliente importa y se requieren cambios en el extremo servidor si se pretende manejar peticiones asíncronas transaccionales. Las peticiones asíncronas transaccionales, sin embargo, no se consideran en esta Recomendación. La figura 8 muestra el concepto básico del modelo AMI del OMG.

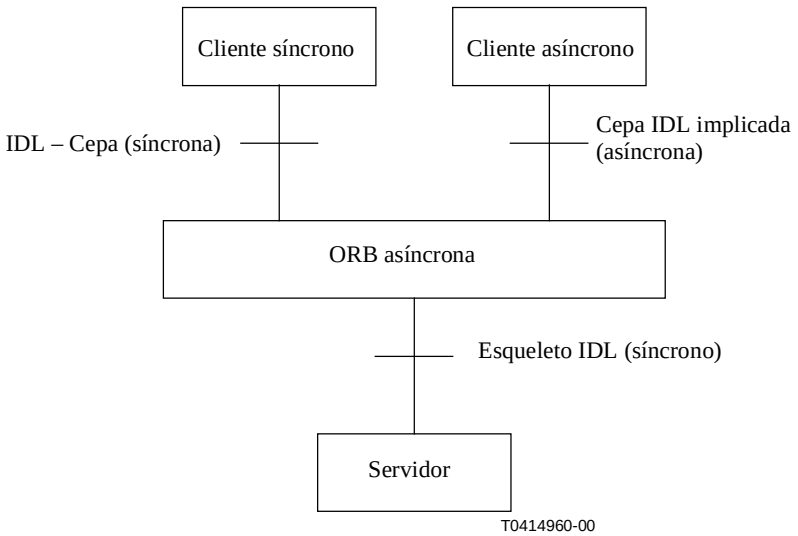

Figura 8/Q.816 – ORB asíncrona

La especificación AMI proporciona dos modelos para peticiones asíncronas: comunicación por intermediario y votación. En el modelo de comunicación por intermediario el cliente transfiere una referencia de objeto para un objeto `ReplyHandler` como parámetro cuando invoca una operación bidireccional asíncrona en un servidor. Cuando el servidor responde, la ORB cliente recibe la respuesta y la envía al método adecuado en el servidor `ReplyHandler` de forma que el cliente pueda manejar la respuesta. En otras palabras, la ORB transforma la respuesta en una petición en el `ReplyHandler` del cliente. `ReplyHandler` es un objeto CORBA normal que está implementado por el programador de la misma forma que cualquier implementación de objeto. En el modelo de votación el cliente establece la petición que pasa por todos los parámetros necesarios para la invocación y le devuelven un objeto `Poller` que puede ser solicitado para obtener los resultados de la invocación. Este `Poller` es una instancia de un `valueType`, que es un nuevo tipo de IDL introducido por la nueva especificación *Objects-by-Value*. Un `valueType` tiene tanto miembros de datos como métodos que, cuando se les invoca, son únicamente llamadas de función local y no invocaciones de operación CORBA distribuidas.

El valor de la capacidad de invocación del método asíncrono en aplicaciones de gestión de red consiste en que permite a los sistemas de gestión que desean utilizar llamadas del método asíncrono interoperar con sistemas gestionados que utilizan las mismas definiciones de interfaz que las utilizadas por clientes síncronos. No se precisan cambios en la definición de interfaz o en la implementación del sistema gestionado. Se proponen los requisitos siguientes para las implementaciones que desean opcionalmente soportar invocaciones del método asíncrono (pero no transacciones con capacidades "ACID").

(O) AMI-1 La implementación CORBA AMI-aware soportará por lo menos el modelo de programación de llamada con intermediario.

(O) AMI-2 Para cada operación en una interfaz IDL, la implementación CORBA AMI-aware generará las firmas correspondientes del método de llamada con intermediario síncrono. Estas firmas se escriben en la IDL implicada que se utiliza para generar firmas de operación específicas de lenguaje.

(O) AMI-3 La ORB CORBA AMI-aware pasará una instancia de valor `ExceptionHolder` específica que contiene las excepciones ordenadas como su estado del `ReplyHandler` cuando el objeto CORBA devuelve respuestas de excepción. El compilador IDL AMI-aware generará un `ExceptionHolder` específico para cada interfaz IDL.

(O) AMI-4 El compilador IDL AMI-aware generará un negociador de respuesta específico para cada interfaz IDL. El cliente implementará y registrará un negociador de respuesta con cada petición asíncrona y recibirá una llamada con intermediario cuando se devuelva una respuesta a dicha petición. El negociador de respuesta se obtiene del `Messaging::ReplyHandler` genérico.

6.5 Servicio de seguridad

El servicio de seguridad CORBA comprende la funcionalidad de seguridad de autenticación de participantes (usuarios humanos y objetos), de autorización de acceso a objetos por los participantes, de auditorías de seguridad, seguridad de comunicación, no repudio y administración [6]. Sin embargo, todo esto puede ser superado para muchas aplicaciones. En su lugar, las aplicaciones podrían necesitar únicamente la seguridad de comunicación y la funcionalidad de autenticación en el sistema basadas en tecnología de seguridad de capa de transporte (TLS) y su precursor (SSL) por razones de disponibilidad y de sencillez. Finalmente, algunas aplicaciones podrían no necesitar seguridad. Por tanto, los requisitos opcionales siguientes reflejan tres posibles opciones:

1) Ninguna seguridad.
2) Las ORB utilizan SSL para proporcionar seguridad de comunicaciones y autenticación de sistema que es fundamentalmente una seguridad de "sesión".

3) Las ORB utilizan el servicio de seguridad CORBA para proporcionar seguridad de comunicaciones, autenticación, no repudio, listas de control de acceso para grupos o individuos que acceden a objetos y operaciones individuales, etc.

El nivel real de servicio a proporcionar en una interfaz se deja como un asunto a negociar entre las partes que suministran sistemas gestionados y de gestión.

(O) SEC-1 La interfaz CORBA puede opcionalmente soportar el "protocolo IOP seguro" o interoperabilidad SSL de seguridad CORBA según se define en la especificación del servicio de seguridad CORBA [6].

(O) SEC-2 El servicio de seguridad CORBA se puede utilizar para soportar su amplia gama de capacidades.

(O) SEC-3 El soporte del intercambio de certificados de autenticación será una opción de la administración.

6.6 Servicio de transacción

En un entorno de cálculo distribuido como es CORBA, es posible que actualizaciones provenientes de algunos clientes se pudieran superponer con actualizaciones concurrentes (o casi concurrentes) provenientes de otros clientes, a menos que se proporcionen salvaguardas adecuadas. Aunque el servicio de notificación y el servicio de registro cronológico telecom proporcionan una base para que el cliente sea consciente de que su actualización se ha sobreescrito, no proporcionan un mecanismo de fijación que evite la aparición de este tipo de sobreescritura. El servicio de transacción OMG [7] proporciona un mecanismo adecuado para el bloqueo para evitar la sobreescritura de una actualización del cliente por una actualización proveniente de un cliente diferente. Esta solución está diseñada para alta fiabilidad. Sin embargo, puede que no sea necesario el servicio de transacción OMG en todas las aplicaciones y puede que no esté justificada la cabecera adicional que implica. Si se tiene que soportar la consistencia de los datos después de actualizaciones concurrentes, debe considerarse el servicio de transacción desde un OMG.

(O) TRANS-1 La interfaz CORBA puede soportar opcionalmente el servicio de transacción OMG para garantizar la consistencia de los datos.

7 Servicios soporte del marco

Esta cláusula define servicios soporte comunes incluidos en el marco que no son servicios de objetos comunes CORBA de OMG normalizados. Muchas aplicaciones de gestión de red realizan funciones que no se requieren normalmente en aplicaciones empresariales de propósito general, por lo que no es razonable esperar que el marco CORBA normalizado suministre todos los servicios necesarios para la gestión de red. Esta cláusula define servicios que las aplicaciones de gestión de red utilizarán ampliamente pero que no es probable que sean utilizadas por muchos otros tipos de aplicaciones y que por tanto es poco probable que lleguen a ser servicios de objetos comunes CORBA. Estos servicios también proporcionan la funcionalidad requerida para permitir la reutilización de modelos de información existentes sin grandes cambios en la semántica.

Las ventajas de situar esta funcionalidad en los servicios soporte comunes es que descarga las implementaciones de objetos gestionados de la provisión de estos servicios y permite a los servicios evolucionar para proporcionar una mayor funcionalidad sin modificar las interfaces o implementaciones de objetos gestionados. En el anexo A se puede encontrar un IDL que describe las interfaces para estos servicios.

7.1 Servicio buscador de factorías

El servicio buscador de factorías permite a los clientes encontrar factorías. Un cliente encuentra una factoría solicitando este servicio con el nombre de clase de la fábrica. El servicio responde con una

referencia a una factoría de este tipo. Hay que destacar que el nombre de clase suministrado por el cliente es la clase de factoría, no la clase del objeto que se va a crear.

El servicio buscador de factorías se encuentra buscándolo en el servicio de denominación.

Antes de poder encontrar factorías en el servicio, el servicio tiene que saber algo de ellas. Por lo tanto, el servicio buscador de factoría también proporciona un medio para que las factorías se registren o se desregistren ellas mismas en el servicio. Aunque no es necesario exponer esta capacidad a través de la interfaz de gestión, estas operaciones se definen para permitir la implementación del servicio buscador de factorías como un componente separado de los objetos que comprenden un modelo de información específico. Por lo tanto, una vez implementado, no será necesario modificar la realización del servicio buscador de factorías cuando se definan nuevos modelos de información con nuevas factorías. El servicio buscador de factorías podría incluso adquirirse a partir de terceros.

Las operaciones utilizadas para registrar y desregistrar canales se encuentran en una interfaz separada que está calificada a partir de la interfaz de buscador de factorías. Sólo la interfaz de buscador de factorías necesita ser implementada para que esté conforme con este marco. La interfaz subclasificada se proporciona para implementaciones que deseen utilizarla para conseguir un servicio buscador de factorías como componente separado.

Éste es el IDL que define la interfaz del servicio buscador de factorías (sin comentarios):

```
interface FactoryFinder {

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

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

}; // end of FactoryFinder interface


interface FactoryFinderComponent : FactoryFinder {

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

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

}; // end of FactoryFinderComponent interface
```

La operación *find* en la interfaz *FactoryFinder* la utiliza el cliente para encontrar una factoría de un tipo determinado. La operación *list* devuelve una lista de todas las factorías registradas con el buscador de factorías. La operación *register* en la interfaz *FactoryFinderComponent* la utiliza una factoría para registrarse en un servicio y la operación *unregister* la utiliza una factoría para suprimir su registro. Los sistemas de gestión no deberían utilizar estas dos ultimas operaciones

(R) REGISTER_FINDER-1 Un sistema gestionado instanciará por lo menos un objeto de servicio de buscador de factorías. Asimismo, cada contexto de denominación de raíz local en un sistema tendría por lo menos una vinculación de nombre para un objeto servidor de buscador de factorías. El valor de la cadena ID en esta vinculación identificará simplemente al servidor, quizás con un valor similar a "FactoryFinder1". La cadena *kind* en la vinculación identificará la clase de objeto ("itut_q816::FactoryFinder").

(R) FACTORY_FINDER-2 El objeto u objetos de servidor de buscador de factorías soportará la interfaz de buscador de factorías descrita anteriormente y definida en el IDL CORBA del anexo A.

El objeto u objetos de servidor de buscador de factorías pueden soportar las interfaces de componentes de buscador de factorías definidas anteriormente. Se soportará la funcionalidad descrita anteriormente.

7.2 Servicio buscador de canal

Se prevé que un gran sistema de gestión de red podría tener múltiples canales de evento. Estos canales podrían manejar distintos tipos de eventos o podrían manejar eventos provenientes de diferentes conjuntos de objetos. Para asegurar que un cliente no pierde ninguno de los eventos en los que está interesado, se precisa algún medio para identificar los canales presentes en un sistema gestionado y los eventos que está manejando. El servicio buscador de canal realiza está función. Un cliente puede invocar una operación en este servicio para enumerar todos los canales de eventos presentes en un sistema gestionado, junto con los eventos que se están manejando. Una vez que el cliente conoce los canales, puede interactuar con ellos para disponer la recepción de notificaciones.

Un cliente encuentra un objeto del buscador de canal buscándolo en el servicio de denominación.

Antes de que el servicio pueda enumerar los canales, este debe tener conocimiento de ellos. Por lo tanto, el servicio buscador de canal también proporciona un medio para que se registren y se desregistren canales en el servicio. Aunque no es necesario exponer esta capacidad a través de la interfaz de gestión, se definen estas operaciones para permitir la implementación del servicio buscador de canal como un componente separado de los objetos que comprenden un modelo de información específico. Por lo tanto, una vez implementado, no será necesario modificar la realización del servicio buscador de canal cuando se definan nuevos modelos de información. El servicio buscador de canal podría incluso adquirirse a partir de terceros.

Las operaciones utilizadas para registrar y desregistrar canales se encuentran en una interfaz separada que está subclasificada a partir de la interfaz de buscador de canal. Sólo es preciso implementar la interfaz de buscador de canal para estar conforme con este marco. La interfaz subclasificada se proporciona para implementaciones que deseen utilizar el buscador de canal como un componente separado.

7.2.1 Interfaz de buscador de canal

Esta es el IDL que define la interfaz al servicio buscador de canal (sin comentarios):

```
interface ChannelFinder {

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

}; // end of ChannelFinder interface


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

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

}; // end of ChannelFinderComponent interface
```

La operación *list* en la interfaz *ChannelFinder* la puede utilizar un sistema de gestión para descubrir todos los canales presentes en un sistema gestionado. La información devuelta es la misma información incluida cuando un canal se registra, como se trata más adelante. Las dos operaciones de la interfaz *ChannelFinderComponent* las utiliza el sistema gestionado para registrar y desregistrar sus canales del servicio. El sistema de gestión no pretende utilizar estas operaciones.

7.2.1.1 Datos de buscador de canal

Cuando se registra un canal se asocian siete partes de los datos con la referencia al canal. En primer lugar, se especifica un identificador de cadena para el canal. En segundo lugar, se identifica el tipo de canal (este marco tiene dos tipos de canales, canales de notificación y canales de registro cronológico).

Después de eso, el conjunto de objetos cubiertos por el canal se especifica mediante un conjunto de nombres de objetos gestionados y "ámbitos" de objetos contenidos debajo de ellos. Véase en 7.4.1.1 más sobre los ámbitos. Hay un ámbito para cada objeto gestionado básicos del conjunto. Estos objetos básicos y los objetos que se hallan dentro de sus ámbitos asociados son los conjuntos de objetos que envían eventos a este canal. Si múltiples canales tienen ámbitos que se superponen, el canal con el objeto básico que es el objeto antecesor más próximo al objeto fuente maneja los eventos de ese objeto, con arreglo a algunas restricciones que se tratan más adelante. Si múltiples canales tienen el mismo objeto básico, y es el antecesor más próximo al objeto fuente, todos esos canales procesan los eventos de ese objeto fuente. Un conjunto vacío tiene el significado especial de que todos los objetos gestionados vinculados directamente al contexto de denominación de raíz local que también contiene este buscador de canal son objetos básicos y que el ámbito de los objetos que envían eventos al canal son los subárboles completos contenidos por estos objetos.

Seguidamente se identifican los tipos de eventos enviados a este canal. Esto se realiza mediante dos conjuntos de cadenas. El conjunto *eventTypes* incluye explícitamente el nombre de cada tipo enviado a este canal. Por ejemplo, un tipo de evento es "itut_x780:Notifications::equipmentAlarm". Un conjunto vacío tiene el significado especial de que todos los tipos de eventos se envían a este canal. El conjunto *excludedEventTypes* enumera los tipos de eventos que no se envían a este canal. Si la cadena *eventTypes* es un conjunto nulo, entonces todos los eventos salvo aquellos enumerados en el conjunto *excludedEventTypes* se envían al canal. Si la cadena *eventTypes* no es un conjunto nulo, el parámetro *excludedEventTypes* estará vacío y se ignorará.

Finalmente, se identifican los tipos de objetos que envían objetos a este canal. Esto también se realiza mediante dos conjuntos de cadenas. El conjunto *sourceClasses* incluye explícitamente el nombre de cada clase de objeto gestionado que envía eventos a este canal. Un ejemplo de nombre de clase de objeto es "itut_m3120::Equipment". Un conjunto vacío tiene el significado especial de que todos los tipos de objetos gestionados envían eventos a este canal. El conjunto e*xcludedSourceClasses* enumera los nombres de clases de objetos gestionados que no envían eventos a este canal. Si la cadena s*ourceClasses* es un conjunto nulo, entonces todas las clases de objetos gestionados salvo aquellas enumeradas en el conjunto *excludedSourceClasses* envían eventos al canal. Si la cadena s*ourceClasses* es un conjunto no nulo, el parámetro *excludedSourceClasses* estará vacío y será ignorado.

7.2.1.2 Cobertura de canal

La combinación de los objetos básicos, lista de tipo de eventos y clases de fuente identifica los eventos manejados por un canal. Esto puede llevar a algunas combinaciones que se prestan a confusión. Aunque no se espera que se implementen todas ellas en sistemas, se explican a continuación por si acaso.

1) Se puede registrar un canal con múltiples objetos básicos y ámbitos.

2) Se pueden registrar múltiples canales con el mismo objeto básico. Si las listas de eventos y los tipos de objeto de fuente asociados con estos canales también se superponen, los eventos

de los objetos de los ámbitos que se superponen deberán estar disponibles a todos los canales que enumeran el evento concordante y los tipos de objeto fuente.

3) Un intento de volver a registrar un canal dará como resultado una actualización del registro de dicho canal. La nueva información se asociará con el canal y se descartará la información anterior. Se enviará una notificación que informe del cambio.

4) Un objeto gestionado siempre envía un objeto al canal con el objeto básico asociado que:

- tiene un objeto básico y un ámbito que incluye el objeto;
- tiene el objeto básico que es el superior más próximo al objeto gestionado (si otros canales también tienen objetos básicos y ámbitos que incluyen el objeto);
- maneja ese tipo de evento; y
- maneja esa clase de objeto.

Si se unen múltiples canales, el objeto gestionado envía el evento a todos los canales que se unen.

La figura 9 muestra gráficamente algunos ejemplos para mayor claridad.

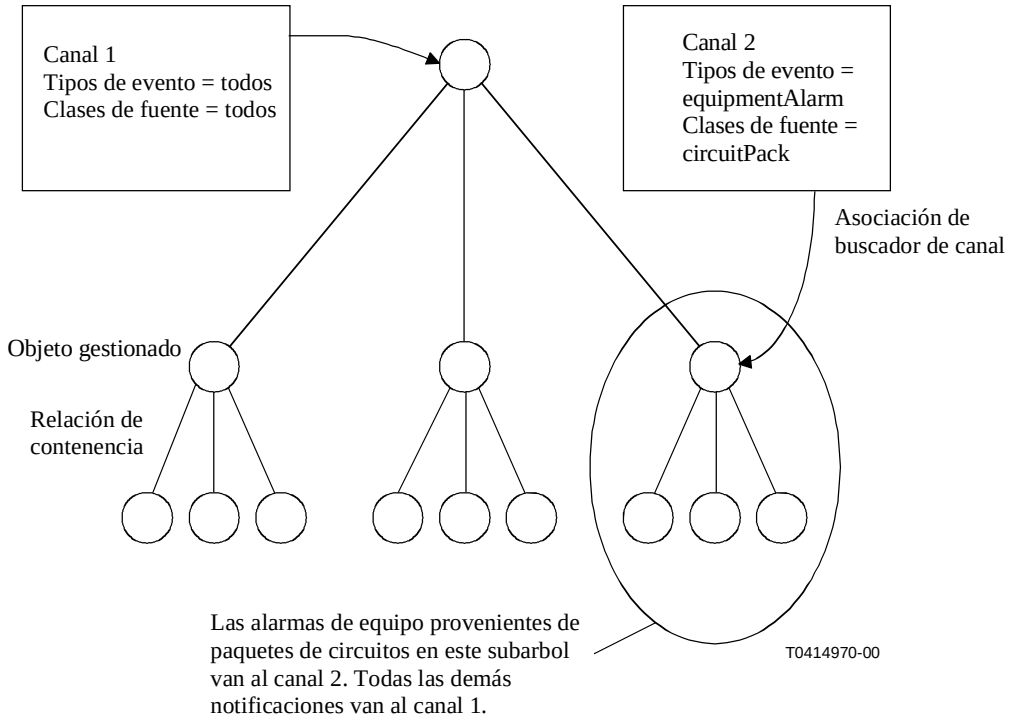

Figura 9/Q.816 – Ejemplo de canal de eventos

7.2.1.3 Notificación de buscador de canal

Se ha definido una notificación especial que tiene que ser enviada siempre que se añada, se suprima o se modifique un registro de canal. Esta modificación se tiene que enviar a todos los canales registrados inmediatamente antes de que se produzca el cambio. Como esta notificación es emitida por el buscador de canal en lugar de un objeto gestionado, y como es enviada a todos los canales, el buscador de canal no deberá enumerar la notificación de cambio de canal junto con las demás notificaciones manejadas por un canal. El IDL que describe esta notificación es:

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

El parámetro de modificación de canal indica si un canal se ha añadido, se ha suprimido o se ha modificado (en términos de los tipos de evento que maneja). El parámetro de información de canal proporciona una cadena que describe el tipo e canal, una referencia al canal, el o los objetos básicos con los que está asociado y las clases de objeto fuente y los tipos de evento que maneja.

Hay que destacar que mientras este planteamiento para soportar múltiples canales es bastante flexible y por tanto complicado, el grado de complejidad está controlado en la realización el sistema gestionado. No está soportada la creación y el registro de canales de eventos a través de la interfaz de gestión. Un sistema gestionado complejo podría soportar un procedimiento administrativo local para añadir, modificar o suprimir canales para adaptar las características de funcionamiento, o podría únicamente actualizar canales mediante entregas de soporte lógico. Un sistema simple tendrá probablemente uno o quizás dos canales (uno para eventos de alta prioridad como alarmas de equipos y otro para cualquier otra cosa) asociados con el objeto gestionado raíz. Asimismo hay que destacar que impedir a un sistema de gestión que cree canales de eventos no le impide crear filtros y "suministradores próximos" en canales existentes. Esto da al cliente una capacidad igual a la de crear discriminadores que envían eventos en los sistemas de gestión de red OSI.

7.2.2 Requisitos del buscador de canal

(R) CHANNEL_FINDER-1 Un sistema gestionado instanciará por lo menos un objeto de servicio de buscador de canal. Asimismo, cada contexto de denominación de raíz local en un sistema tendrá por lo menos una vinculación de nombre para un objeto servidor de buscador de canal. El valor de la cadena ID en esta vinculación simplemente identificará al servidor, quizás con un valor similar a "ChannelFinder1". La cadena *kind* en la vinculación identificará la clase de objeto ("itut_q816::ChannelFinder").

(R) CHANNEL_FINDER-2 El objeto u objetos de servidor del buscador de canal soportarán la interfaz de buscador de canal descrita anteriormente y definida en el IDL CORBA del anexo A. El objeto u objetos de servidor de buscador de canal pueden soportar las interfaces de componentes de buscador de canal definidas anteriormente. Se soportará la funcionalidad descrita anteriormente.

(R) CHANNEL-FINDER-3 Siempre que se realice un cambio a los registros de canal, el buscador de canal enviará una notificación de cambio de canal en todos los canales registrados inmediatamente antes del cambio.

(R) CHANNEL-FINDER-4 La red de canales de eventos informada por un sistema gestionado manejará todas las alarmas provenientes de todos los objetos gestionados en el sistema. Un sistema que enumera un conjunto de canales y que no cubre todos los eventos provenientes de todos los objetos gestionados en el sistema no cumple con este marco.

7.3 Servicio terminador

El objeto del servicio terminador es proporcionar un lugar en el marco para realizar funcionalidad común que en otro caso tendría que realizarse en los objetos gestionados. El servicio terminador lo utilizan los sistemas de gestión para suprimir objetos gestionados. Hace esto de conformidad con la política de supresión del objeto gestionado. (UIT-T X.780 requiere que cada objeto gestionado tenga un atributo *deletePolicy* con uno de tres valores: *notDeletable*, *deleteOnlyIfNoContainedObjects*, y *deleteContainedObjects*.) Si la política de supresión del objeto gestionado es *notDeletable*, el servicio terminador no suprime el objeto y en su lugar establece una excepción. Si la política de supresión es *deleteOnlyIfNoContainedObjects* y el objeto no contiene ningún objeto, entonces el servicio terminador suprime el objeto. En otro caso, establece una excepción. Finalmente, si la política de supresión del objeto es *deletedContainedObjects*, el servicio terminador suprimirá el objeto así como todos los objetos contenidos, siguiendo algunas reglas que se definen a continuación.

La Recomendación UIT-T X.780 también define una operación destroy que deben soportar todos los objetos gestionados establecido para su utilización por el servicio terminador para realmente suprimir un objeto gestionado y liberar su recurso. Además de las siguientes políticas de supresión y de la supresión real de objetos gestionados, el servicio terminador también es un buen lugar para implementar códigos que mantenga la integridad del árbol de denominación al suprimir vinculaciones de nombre para objetos gestionados que se están suprimiendo. Las realizaciones pueden elegir implementar esta función en otro lugar, pero una de las funciones del marco consiste en permitir realizaciones de objetos destinados que se centran en representar recursos de red. Se considera que un servicio como éste ayudará a hacer que la implementación de objetos gestionados sea más sencilla.

El IDL que describe el servicio terminador proporciona dos métodos para suprimir el objeto gestionado. Uno identifica el objeto gestionado por su nombre, el otro por referencia. Esta es el IDL que define la interfaz del servicio de supresión:

```
interface TerminatorService {

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

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

        }; // end of TerminatorService interface
```

(R) TERM-1 Un sistema gestionado instanciará por lo menos un objeto de servicio terminador. Asimismo, cada contexto de denominación de raíz local en un sistema tendrá por lo menos una vinculación de nombre para un objeto servidor de terminador. El valor de la cadena ID en esta vinculación identificará sencillamente al servidor, quizás con un valor similar a "Terminator1". La cadena *kind* en la vinculación identificará la clase de objeto ("itut_q816::TerminatorService").

(R) TERM-2 La interfaz soportada por el objeto u objetos del servicio de terminador será la interfaz del terminador descrita anteriormente y definida en el IDL CORBA del anexo A. Se tiene que soportar la funcionalidad descrita anteriormente.

(R) TERM.-3 El servicio terminador suprimirá objetos de conformidad con el atributo de política de supresión de objetos se establece cuando se crea y no se puede cambiar. Hay que destacar que el servicio terminador no es un servicio delimitado. El servicio terminador puede realmente suprimir múltiples objetos al responder a una única petición, pero su cometido es suprimir el único objeto solicitado. El servicio terminador implementará las reglas siguientes para suprimir un objeto:

1) Ningún objeto debe quedar "huérfano", es decir, no se puede suprimir un objeto sin suprimir a sus subordinados.

2) Si el objeto tiene una política de supresión *notDeletable*, el objeto no se suprimirá, ni tampoco ninguno de sus subordinados si los tiene. La excepción *DeleteError* se establecerá con el identificador error fijado al valor *notDeletable*.

3) Si el objeto tiene una política de supresión *deleteOnlyIfNoContainedObjects* y no tiene ningún subordinado, el objeto se suprimirá. Si el objeto tiene subordinados, independientemente de sus políticas de supresión, no se suprimirá ni tampoco lo serán sus subordinados. La excepción *DeleteError* se establecerá con el identificador de error fijado al valor *containsObjects*.

4) Si el objeto tiene una política de supresión *deleteContainedObjects*, y no tiene subordinados, el objeto se suprimirá. Si el objeto tiene subordinados, el servicio terminador comprobará las políticas de supresión de todos los subordinados. Si algunas son *notDeletable*, no se suprimen objetos. Si algunas son *deleteOnlyIfNoContained*, y contienen subordinados, no se suprimen objetos. En otro caso, se suprimen el objeto y sus subordinados.

5) El servicio terminador suprimirá los servicios contenidos de abajo a arriba. Si cualquier objeto contenido establece una excepción durante la supresión, el servicio terminador no eliminará ese nombre de objeto y no suprimirá ninguno de sus superiores. El servicio terminador, sin embargo, continuará tratando de suprimir otros objetos contenidos. Cuando se suprimen todos los objetos que pueden ser suprimidos, el servicio terminador establecerá una excepción *DeleteError* con el identificador de error fijado al valor *undeletableContainedObject*. Este método optimista de suprimir objetos contenidos es necesario para hacer los resultados deterministas. El cliente puede identificar los objetos no suprimibles, porque estarán en las hojas del árbol que quedan por debajo del objeto fijado como objetivo.

6) Si el objeto básico formula una excepción *DeleteError*, el servicio terminador devolverá la misma excepción (con los datos incluidos). No se suprime el objeto y el nombre del objeto tampoco se suprime del árbol de denominación.

7.4 Servicio de operación multiobjeto

Con la posibilidad de gestionar millones de entidades, el marco necesita soportar operaciones en múltiples objetos con una única invocación de método o quizás con un pequeño número de invocaciones. El servicio de operación multiobjeto (MOO) proporciona esta capacidad.

Se espera que cada plataforma de gestión de red que soporte una interfaz CORBA proporcionará por lo menos una instancia del servicio MOO. (Por razones de características de funcionamiento, se recomienda que el servicio MOO, el servicio de denominación y los objetos gestionados se sitúen en la misma plataforma de cálculo.) Los gestores interactuarán con el servicio utilizando un número limitado de interacciones que requieran una anchura de banda relativamente baja. El servicio a su vez interactuará con objetos gestionados utilizando sus interfaces CORBA publicadas o algún medio propietario. Se espera que este gran número de interacciones requiera una anchura de banda mayor, de ahí la necesidad de situar el servicio en la misma ubicación que los objetos gestionados.

Hay que destacar que el servicio MOO es un ejemplo de la granularidad de acceso específica de aplicación que se ha tratado en la cláusula 4.

7.4.1 Interfaz del servicio MOO

La interfaz del servicio MOO, definida en el anexo A, es de tipo débil. Proporciona un conjunto de capacidades genéricas que pueden ser invocadas en conjuntos de cualquier tipo de objeto gestionado, incluso de objetos de tipos diferentes. Las operaciones que se soportan son:

- Scoped get: Devuelve los valores desde cada uno de los objetos para una lista de atributos.
- Scoped updated: Se utiliza para sustituir un valor de atributo o para añadir y/o suprimir valores en/de atributos set-valued. Se puede utilizar para actualizar uno o múltiples atributos en un único objeto o en múltiples objetos.
- Scoped delete: Suprime múltiples objetos.

La operación scoped get se define en la interfaz *BasicMooService*. Las operaciones scoped updated y delete se definen en la interfaz *AdvancedMooService*, que hereda la operación scoped get de la interfaz de servicio básico. Esto se hizo para permitir alguna flexibilidad en la implementación del servicio de operación multiobjeto. Un servicio básico necesita únicamente implementar la operación scoped get.

7.4.1.1 Parámetros comunes en las operaciones del servicio MOO

Cada una de las operaciones delimitadas necesita cuatro parámetros para definir el conjunto de objetos en los que la operación se realizará:

- Nombre de objeto básico: Nombre del objeto en la raíz del árbol de objetos en el que la operación probablemente se realizará.

- Ámbito: Una unión discriminada que identifica los objetos contenidos en el objeto básico en el que la operación probablemente se realizará. La unión tiene cuatro casos. Dos de los casos incluyen un entero que especifica un nivel de objetos contenido por debajo del objeto básico. Las cuatro opciones son:
  - Objeto básico solamente. Si el ámbito es *baseObjectOnly*, sólo se incluye en el ámbito el objeto (básico) objetivo denominado.
  - Subárbol completo. Si el ámbito es *wholeSubtree*, el ámbito son todos los objetos contenidos por debajo del objeto básico, junto con el objeto básico.
  - Nivel individual. Si el ámbito es *individualLevel*, el ámbito incluirá también un nivel de valor entero. Todos los objetos contenidos a un nivel por debajo del objeto básico igual a este valor están en el ámbito. Los objetos directamente contenidos por el objeto básico son de nivel uno. Si *level* es igual a cero, el ámbito es el objeto básico.
  - Básico a nivel. Si el ámbito es *baseToLevel*, el ámbito incluirá también un *level* de valor entero. El ámbito serán todos los objetos hasta el nivel dado, incluido el objeto básico y el objeto al nivel dado. Si *level* es igual a cero, el ámbito es únicamente el objeto básico.

  Debido a que este marco utiliza algunos convenios de denominación únicos, el servicio tiene que trabajar para determinar la profundidad real de los ámbitos basados en contenencia. El objetivo consiste en hacerlo tan simple como sea posible para el cliente. En primer lugar, el nombre del objeto básico será el nombre del componente completo, incluido el componente final con un valor ID "objeto". El servicio tendrá que "volver" al contexto de denominación que contiene esta vinculación e iniciar la cuenta desde ahí. Asimismo, el servicio seguirá automáticamente las vinculaciones "Objeto" en los contextos de denominación de objetos gestionados en el ámbito. Este último salto no contará en la profundidad.
- Filtro: Expresión escrita en un lenguaje de restricción que se utiliza para evaluar los atributos de un objeto. La operación se aplica a aquellos objetos en el ámbito para los que la expresión filtro se evalúa como *verdadero*.
- Lenguaje: Cadena que indica el lenguaje en el que la expresión filtro está escrita.

Los nombres de objeto son los mismos que el tipo *nombre* definido por el servicio de denominación CORBA. El ámbito es una unión con valores como los que se han descrito anteriormente. Finalmente, los parámetros filtro y lenguaje son cadenas.

Cada una de las operaciones puede formular alguna de estas excepciones:

- Una excepción *InvalidParameter* si uno de los parámetros tiene un valor no válido. Se devuelve el nombre del parámetro no válido. Estas son algunas condiciones bajo las cuales se formulará esta excepción:
  - El nombre del objeto básico no es válido.
  - Se suministra un valor de lenguaje de filtro no reconocido.
- Un excepción *InvalidFilter* si la sintaxis del filtro no es correcta. Esta excepción devuelve el texto cercano al error de sintaxis para la resolución del problema.
- Una excepción *FilterComplexityLimit* si la sintaxis del filtro es correcta, pero el filtro es demasiado complejo para ser procesado por el sistema gestionado.
- Una excepción *ApplicationError* para referirse a otros problemas en el servidor (como es una falta de recursos) que le hacen imposible llevar a cabo la operación solicitada.

Hay que destacar que si una expresión no se puede evaluar para un determinado objeto debido a que los tipos de sus atributos no concuerdan con la expresión, el filtro no es inválido. Es posible que el objeto sencillamente no pueda pasar por el filtro.

Los demás parámetros para las operaciones así como los tipos de devolución son específicos de la operación. Por ejemplo, la operación scoped get toma una lista de nombres de atributo y devuelve

una secuencia de resultados, una a partir de cada objeto, con el nombre del objeto asociado con los resultados de ese objeto.

7.4.1.2 Iteradores del servicio MOO

Puesto que cada una de las operaciones podría devolver grandes cantidades de datos, se utiliza para devolver los resultados el patrón de diseño de iterador. Un iterador es un objeto que está creado para contener los resultados de una operación de forma que puedan ser devueltos al cliente a una velocidad determinada por el cliente. El cliente recibe una referencia del iterador como parte de la información devuelta por el método. El cliente puede entonces invocar operaciones en el iterador para recibir grupos de resultados de tamaños definidos por el cliente. El iterador mantiene el control de hasta donde ha progresado el cliente en los resultados.

Cada vez que un cliente invoca una operación *get* en un iterador, suministra el máximo número de respuestas que desea recibir. El iterador no devolverá más que el tamaño de grupo solicitado, en una estructura de datos *sequence*. El iterador también devuelve una indicación booleana si hay más resultados que extraer. El iterador puede devolver un tamaño de grupo menor que el solicitado, compensando la eficacia de devolver resultados en un grupo grande con la posible necesidad de bloquear hasta que haya disponibles más resultados. El iterador no devolverá una secuencia vacía a menos que no haya más resultados que devolver, ya que al hacerlo forzaría al cliente a interrogar al iterador.

El sistema gestionado controla el ciclo de vida del operador. Sin embargo, se provee una operación destruir si el operador desea cesar de extraer resultados antes de llegar a la última iteración. Al devolver el último resultado, el iterador se destruirá así mismo. El iterador puede también ser destruido por el sistema gestionado si no es utilizado durante un periodo de tiempo razonablemente largo.

Los iteradores se utilizan para organizar la devolución de información sólo desde la operaciones y no deberían controlar cuando se invocan realmente las operaciones en objetos individuales. Una operación delimitada será invocada en los objetos y los resultados se pondrán en fila tan pronto como sea posible. Puede ser más eficaz suprimir la invocación de la operación en los objetos gestionados individuales hasta que se soliciten los resultados a través del iterador, pero esto puede dar lugar a resultados incorrectos o a condiciones competitivas.

Las cláusulas siguientes ofrecen detalles adicionales sobre cada una de las operaciones delimitadas.

7.4.1.3 Scope Get

La firma IDL para la operación scoped get en el servicio MOO básico es:

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

Como se ha descrito anteriormente, los primeros cuatro parámetros se utilizan para seleccionar un conjunto de objetos en los que realizar la operación get. Para cada uno de ellos el servicio intentará devolver un valor para cada atributo denominado en el parámetro "atributos", que es simplemente una lista de cadenas. Sin embargo, la presentación de una lista de atributos nulos tiene el significado especial de que se devolverán todos los valores de atributo para los objetos que pasen el filtro. Los tipos implicados en el valor de retorno son:

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

typedef sequence <AttributeValueType> AttributeSetType; // framework

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

typedef sequence <GetResultsType> GetResultsSetType;
```

Los primeros dos tipos forman una lista de pares *nombre-valor*. El tipo retorno es una secuencia de estructuras, una para cada objeto que pasó el filtro. En dicha estructura se encuentran un nombre de objeto, una bandera que será verdadera si el objeto no pudiera ser evaluado para ver si pasó el filtro, la lista de valores de atributo de ese objeto y los nombres de cualesquiera atributos que no pudieran extraerse de ese objeto. Los objetos que no pudieran ser filtrados se marcan con banderas como caso especial, ya que pueden tratarse de objetos en los que el cliente ni siquiera está interesado. Si el objeto no pudiera ser filtrado, el cliente sabrá que el servidor MOO no pudo extraer atributos de ese objeto, por lo que los otros dos miembros de datos estarán vacíos. Si un objeto pasa el filtro pero no se pudiera retirar un valor de atributo, ya sea porque el objeto no tuviera un atributo concordante o que se hubiera formulado alguna excepción sobre el acceso, el nombre de dicho atributo debe ponerse en la lista de atributos fallidos para ese objeto.

El parámetro *howMany* indica al servicio cuántos resultados de los objetos se incluirán en el primer conjunto de respuestas. (Se permite cero, lo que obliga a devolver todos los resultados a través del iterador). El parámetro de salida *resultsIterator* es una referencia a un objeto iterador que se puede utilizar para extraer resultados adicionales por grupos. Si se devolvieran todos los resultados mediante la operación *scopedGet*, esta referencia será nula. El cliente tiene que destruir este objeto cuando termina con él y puede hacerlo antes de que se hayan recuperado todos los resultados. La funcionalidad de la operación CMIP Cancel Get se provee en este marco destruyendo el iterador de resultados.

7.4.1.4 Scoped Update

La firma IDL para la operación scoped update en el servicio MOO avanzado es:

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

De nuevo, los primeros cuatro parámetros se utilizan para seleccionar el conjunto de objetos sobre los que se realiza la actualización. La lista de modificaciones es una lista de estructuras, cada una con el nombre de un atributo, un valor para dicho atributo y un valor enumerado que indica si el valor debe sustituir el valor actual del atributo, si se debe añadir al valor actual del atributo o si se debe suprimir de él. Las opciones añadir y suprimir son válidas si el tipo de atributo es una secuencia CORBA y si la interfaz tiene operaciones añadir y suprimir para el atributo. Los valores en las

estructuras de la lista de modificaciones pasan como tipos *any* de CORBA. Si el tipo de atributo es una secuencia CORBA, en el campo *any* se pondrá una secuencia del tipo adecuado, incluso si sólo contiene un valor único. La IDL que describe la lista de modificaciones es:

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

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

typedef sequence <ModificationType> ModificationSeqType;
```

La bandera *failuresOnly* se utiliza para indicar si el cliente desea que el servicio devuelva resultados para todos los objetos que cumplen el ámbito y el filtro, o únicamente para aquellos objetos para los cuales no se podría realizar por lo menos una de las modificaciones, incluso aunque se satisfaga el ámbito y el filtro.

El valor retorno es una lista de estructuras que contiene cada una un nombre de objeto, junto con un valor booleano que indica si el objeto no pudo ser evaluado para ver si pasó el filtro y una lista de todos los atributos que no pudieron ser modificados. Si el servicio no puede interactuar con el objeto para determinar si pasa el filtro, los resultados para ese objeto tendrán el *notFilterable* puesto a verdadero y el miembro de datos *failedAttributes* estará vacío. (Esto se banderiza como un caso especial porque el objeto puede ser uno en el que el cliente ni siquiera está interesado.) El servicio intentará realizar todas las modificaciones de la lista, por orden, y continuará intentando el resto incluso si falla una modificación. Si cualquier operación falla en un atributo, se añade el nombre del atributo a la lista de fallos. Si la bandera *notFilterable* es falsa, y el miembro de datos *failedAttributes* está vacío, el cliente sabrá que se efectuaron en ese objeto todas las actualizaciones. Los nuevos tipos implicados en el valor retorno son:

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

typedef sequence <UpdateResultsType> UpdateResultsSetType;
```

El parámetro *howMany* indica al servicio cuántos resultados de los objetos se incluirán en el primer conjunto de respuestas. (Se permite cero, lo que obliga a devolver todos los resultados a través del iterador). El parámetro de salida *resultsIterator* es una referencia a un objeto de iterador que se puede utilizar para extraer resultados adicionales por grupos. Si la operación *scopedUpdate* devuelve todos los resultados, esta referencia será nula. El cliente tiene que destruir este objeto cuando termina con él y puede hacerlo antes de que se hayan recuperado todos los resultados.

7.4.1.5 Scoped Delete

La firma IDL para la operación scoped delete en el servicio MOO avanzado es:

```
DeleteResultsSetType scopedDelete (
        in NameType baseName,
        in ScopeType scope,
        in FilterType filter,
        in LanguageType language,
        in boolean failuresOnly,
        in unsigned short howMany,
        out DeleteResultsIterator resultsIterator)
        raises (InvalidParameter,
                InvalidFilter,
                FilterComplexityLimit,
                Itut_x780::ApplicationError);
```

Antes que acceder a valores de atributo, esta operación sencillamente intenta suprimir cada objeto en el ámbito que pasa el filtro.

La bandera *failuresOnly* se utiliza para indicar si el cliente desea el servicio para devolver resultados para todos los objetos que satisfagan el ámbito y el filtro, o simplemente los objetos que no pudieron ser suprimidos. Como suele enviarse la notificación de supresión de objetos, los clientes pueden a menudo desear recibir resultados únicamente para aquellos objetos que no pudieron ser suprimidos.

El valor retorno enumera el nombre de cada objeto junto con dos banderas que podrían indicar que el objeto no pudo ser suprimido. La bandera *notFilterable* será verdadero si el servicio MOO pudiera interactuar con el objeto incluso para determinar si pasó el filtro. La bandera *notDeletable* será verdadero si el objeto pasó el filtro, pero no pudo ser suprimido, ya sea debido a la política de supresión, o porque establecía una excepción. Un objeto que no puede ser evaluado por comparación con el filtro se banderiza como un caso especial para permitir al cliente saber si puede ser un objeto del que ni siquiera pretendía la supresión.

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

typedef sequence <DeleteResultsType> DeleteResultsSetType;
```

El parámetro *howMany* indica al servicio cuántos resultados de objeto se incluirán en el primer conjunto de respuestas. (Se permite cero, lo que obliga a devolver todos los resultados a través del iterador). El parámetro de salida *resultsIterator* es una referencia a un objeto de iterador que se puede utilizar para extraer resultados adicionales por grupos. Si la operación *scopedDelete* devuelve todos los resultados, esta referencia será nula. El cliente tiene que destruir este objeto cuando termine con él y puede hacerlo antes de que se hayan recuperado todos los resultados.

Debido a que muchos objetos no se pueden suprimir si contienen otros objetos, para ámbitos basados en relaciones de continencia, el servicio tiene que empezar suprimiendo los objetos "hoja" que se encuentran dentro del ámbito y proceder hacia el objeto "raíz". Cuando suprime objetos, el servicio MOO tiene que seguir las reglas de supresión de un objeto basadas en la política de supresión de objetos que se describe en 7.3. Aunque se están aplicando reglas a cada uno de los objetos en el ámbito, a partir de la parte superior, sin embargo, el efecto será diferente que intentar sencillamente suprimir el objeto en la raíz del subárbol. Asimismo, el servicio MOO es optimista. Por lo tanto, es posible que algunos de los objetos en un subárbol delimitado sean suprimidos mientras que otros no. Estas son las reglas que tienen que aplicarse a operaciones scoped delete:

1) Ningún objeto puede quedarse "huérfano". Es decir, no se puede suprimir un objeto sin suprimir todos sus objetos contenidos (hijo).

2) Realizar una operación scoped delete en todo un subárbol da como resultado la supresión de todos los objetos en ese subárbol, a menos que un objeto tenga una política de supresión *notDeletable*, que el objeto estableció una excepción en la operación destruir, o que es un objeto que tiene un subordinado que no se puede suprimir.

3) La realización de una operación scoped delete en parte de un subárbol requiere la evaluación de cada uno de los objetos de la capa delimitada más baja que utilice las reglas de supresión de 7.3. Si se puede suprimir un objeto en la capa más baja del ámbito de conformidad con estas reglas, se suprimen el objeto y cualquier subordinado. Si no se puede suprimir un objeto de la capa más baja, no se suprime, ni tampoco ninguno de sus objetos superiores. Sin embargo, se pueden suprimir otros objetos en el ámbito si las reglas de supresión lo permiten. El servicio entonces procede en la capa siguiente y así sucesivamente.

7.4.2 Lenguaje de filtro por defecto

Esta cláusula describe el lenguaje de restricción de filtrado por defecto que tiene que ser soportado por todas las implementaciones conformes del servicio MOO. Las implementaciones conformadoras pueden soportar otras gramáticas de restricción además de la gramática descrita aquí. Se provee una operación en la interfaz del servicio MOO básico para permitir a un cliente extraer los lenguajes soportados por un servicio. La gramática utilizada en una petición se indica mediante un parámetro con valor de cadena denominado "lenguaje" en cada operación. Un valor "MOO 1.0" (con un espacio entre "MOO" y "1.0") indicará la gramática descrita aquí. (Se provee en el módulo IDL una constante denominada *defaultLanguage* con este valor.)

La gramática por defecto soportada por cada implementación conformadora será la gramática de restricción por defecto definida por la versión 1.0 del servicio de notificación [4] con los cambios que se describen en las siguientes subcláusulas. Hay que destacar que al considerar este planteamiento, el marco fija el soporte para reglas de comparación (o "reglas de concordancia") con la gramática de filtro. No se pueden añadir nuevas reglas (por ejemplo, una concordancia de cadena insensible al caso) con la adición de un nuevo tipo de datos o de atributo. En cambio, se deberá actualizar la gramática si se precisan nuevas capacidades.

7.4.2.1 Aplicación del lenguaje de restricción a atributos de objeto

La gramática de restricción del servicio de notificación por defecto introdujo un testigo especial "$" para indicar las variables actuales de evento y de paso del tiempo. Para operaciones de objetos múltiples, el testigo "$" indicará el objeto "actual" así como variables del paso del tiempo. Es decir, uno puede pensar del servicio MOO que selecciona un conjunto de objetos basándose en el nombre básico suministrado y en el parámetro ámbito y que, posteriormente, aplica la expresión filtro individualmente a cada uno de los objetos de ese conjunto. El objeto "actual" es el objeto contra el que se está evaluando la expresión. Los ejemplos siguientes ilustran la expresión "$":

$.administrativeState	Atributo de estado administrativo del objeto actual.
$curtime	Una variable "incorporada" denominada "curtime".

Los identificadores que vienen después de "$." (signo de dólar seguido de un punto) son nombres de los atributos del objeto actual que se encuentran en el objeto valor definido para devolver los atributos de un objeto. (Para más detalles véase UIT-T X.780 sobre atributos de objetos gestionados.) Es decir, "$.administrativeState" se refiere al miembro denominado "administrativeState" en el tipo de valor devuelto por una llamada a la operación *getAttributes*() en el objeto actual. (Se supone que muchas implementaciones del servicio MOO utilizarán la operación *getAttributes*() para extraer los atributos de un objeto antes de evaluar el filtro).

El lenguaje de restricción del servicio de notificación, en el que se construye el lenguaje de restricción del servicio MOO tiene un operador "punto" (".") que se puede utilizar para acceder a miembros individuales dentro de una estructura de datos y puede albergar estructuras de datos. Por lo tanto, se puede utilizar un identificador como éste para acceder a un valor dentro de un atributo que tiene un valor de estructura de datos:

$.systemTimingSource.primaryTimingSource

Una comparación con un nombre de atributo que no esté presente en un objeto siempre se evalúa como falsa. Por ejemplo, en la expresión "(A == 0) || (A != 0)", si no está presente un atributo denominado "A" en el objeto, ambas comparaciones la evaluarán como falsa y la expresión se evaluará como falsa. El lenguaje del servicio de notificación por defecto no soporta una operación "existe" que se pueda utilizar para comprobar la existencia de un atributo antes de incluirlo en una comparación. Asimismo, una comparación siempre se evalúa como falsa si los tipos de los operandos no concuerdan. En el ejemplo anterior, si "A" es una cadena la expresión será falsa.

Cabe destacar que la gramática de restricción del servicio de notificación por defecto define un conjunto de variables runtime (que se pueden considerar mejor como variables "incluidas" o "predefinidas") pero no admite variables definidas por el usuario en expresiones de filtro. De hecho, no hay operador de asignación que pueda soportar la utilización de variables definidas por el usuario. Actualmente no existen variables incluidas definidas por el servicio de delimitación y filtrado y no se soportan variables definidas por el usuario.

NOTA – Puesto que el servicio de notificación evalúa objetos basándose en los nombres de sus atributos, se tiene que tener cuidado al definir los nombres de atributo (los nombres de los miembros del objeto del valor del atributo definido para una interfaz). Un atributo del tipo *administrativeStateType* denominado "adminState" con un valor "unlocked" fracasará en un filtro "administrativeState == unlocked" puesto que el nombre no concuerda.

7.4.2.2 Soporte para expresiones normales

El lenguaje de restricción del servicio de notificación por defecto define un operador de subcadena que funciona de esta forma: "A ~ B" es verdadero si A es una subcadena de B. El lenguaje de restricción del servicio MOO amplía el lenguaje definiendo un operador de concordancia de expresiones normales. El carácter "#" es designado para este operador. Utilizando éste, "A # B" se evalúa como verdadero si la plantilla de expresión normal definido en A concuerda con B. Para este marco, las expresiones normales son expresiones normales "modernas" según se define en 2.8 de POSIX 1003.2 [11].

Una expresión normal es una plantilla que describe un conjunto de cadenas. La inclusión de caracteres especiales conocidos como "metacaracteres" permite a una cadena describir un conjunto de cadenas. La página manual para el comando "grep" de la mayoría de los sistemas que cumplen POSIX dan una descripción completa de expresiones normales y de su uso.

La concordancia de expresiones normales se añade al lenguaje de restricción para satisfacer el requisito de concordancia de subcadenas al principio, en medio o al final de una cadena. Las expresiones normales POSIX soportan esta capacidad utilizando metacaracteres que representan el principio o el final de una cadena ("^" y "$"). La concordancia en medio de una cadena se realiza excluyendo estos caracteres de la expresión normal. Ciertamente, este requisito se podría haber cumplido añadiendo únicamente un par de caracteres a la función de concordancia de la cadena. Sin embargo, se consideró que, puesto que la concordancia de expresiones normales está soportada como una utilidad de sistemas que cumplen POSIX, tendría sentido proseguir y utilizar esta capacidad que añade una buena concordancia de plantilla al lenguaje, antes que requerir desarrolladores para implementar una capacidad especial que ofreciera mucha menos funcionalidad.

Como tanto el lenguaje del filtro de notificación para cadenas y el lenguaje de expresión normal utilizan "\" para el carácter de escape, "\\" debe utilizarse en una cadena de filtros (que será cambiada por el analizador de cadenas de filtros a "\"), siempre que el carácter "\" deba incluirse en la expresión normal. Considérese este ejemplo. El "." en una expresión normal puede concordar con cualquier carácter único. Para asegurarse de que el canal concordado es realmente un carácter ".", podría utilizarse la siguiente operación en un filtro: "'itu\\.int' # $.domain_name.

7.4.2.3 Soporte de los atributos de prueba Set-valued y Sequence-valued

Las aplicaciones de gestión de red tienen que apoyarse fuertemente en los atributos de los objetos gestionados y, a menudo, estos atributos son realmente conjuntos o secuencias de valores. (Los conjuntos y las secuencias son diferentes. Los conjuntos no deben ir duplicados y no importa el orden de los elementos). Para soportar la utilización de atributos Set-valued en expresiones de filtro, el lenguaje de restricción del servicio de notificación por defecto necesita ser ampliado. Se requieren dos grupos de ampliaciones para soportar la utilización de conjuntos y secuencias. El primero permite incluir en la expresión de filtro conjuntos y secuencias de valores literales. El segundo define operadores para conjuntos y secuencias.

7.4.2.3.1 Conjuntos y secuencias de valores literales

Se incluyen conjuntos y secuencias de valores literales en expresiones de filtro poniendo entre corchetes y separando por comas una lista de valores literales. Por ejemplo:

{1, 16, 21}	Conjunto o secuencia de números enteros
{5.2, 6.8, 7.01}	Conjunto o secuencia de números de punto flotante
{"manzana", "naranja"}	Conjunto o secuencia de cadenas
{Crítico, Mayor, Menor}	Conjunto o secuencia de valores enumerados
{}	Conjunto o secuencia nulo.

Los valores literales tienen que ser de los tipos "simples" definidos para el lenguaje de restricción del servicio de notificaciones (booleano, corto, corto sin firmar, largo, largo sin firmar, flotante, doble, char, wchar, string, wstring), o valores enumerados. Todos los valores en un conjunto o secuencia tienen que ser del mismo tipo.

Evidentemente, en este lenguaje de restricción se definen conjuntos y secuencias literales de la misma forma. Realmente, esto concuerda con el caso de los tipos de atributos de interfaz CORBA. Al contrario que otros lenguajes de sintaxis de interfaz, la IDL OMG sólo tiene una estructura de secuencia y ningún tipo de conjunto. Para tener esto en cuenta, se definen diferentes operaciones para conjuntos y secuencias. Cuando se aplica un operador de secuencia a una pareja de secuencias (ya sean valores literales o de atributo), las secuencias se tratan como verdaderas secuencias. Es decir, se tiene en cuenta el orden. Cuando están implicadas dos secuencias en una operación de conjunto, sin embargo, las secuencias se tratan realmente como conjuntos. Es decir, el orden de los valores en el conjunto no importa. Asimismo, mientras que los objetos gestionados nunca deben devolver duplicados en el valor de un atributo set-valued, cualquier duplicado se deberá ignorar.

7.4.2.3.2 Operadores de conjunto

Para incluir los atributos set-valued en expresiones de filtro, se precisan operadores que funcionan en conjuntos. Esta cláusula amplia el lenguaje de restricción del servicio de notificación definiendo como se deben los operadores ya definidos para ese servicio aplicar a conjuntos. Para comprobar la intersección de dos conjuntos se define un nuevo operador utilizando el símbolo de intercalación ('^'). Asimismo, se definen dos funciones incorporadas que toman conjuntos como argumentos.

Cabe destacar que el lenguaje de restricción del servicio de notificación por defecto ya define un operador que funciona con conjuntos, el operador "en". La expresión "A en B" sólo se puede aplicar si A es un tipo simple como se ha definido anteriormente y B es una secuencia del mismo tipo simple. La expresión se evalúa como verdadera si el valor representado por A es igual al valor en B. Asimismo, el lenguaje de restricción del servicio de notificación por defecto soporta la utilización de la operación "existe" en parámetros valorados por conjuntos. Este comportamiento también será soportado para operaciones de múltiples objetos.

Generalmente, para utilizar cualquiera de los operadores de conjunto en una expresión "A <operator> B", uno o ambos operandos tiene que ser una secuencia de uno de los tipos enumerados anteriormente en la cláusula de los conjuntos de valores literales. Si uno de los operandos es una secuencia del tipo X, el otro tiene que ser o una secuencia de tipo X o un valor de tipo X. Los operandos tienen que ignorar cualquier valor duplicado dentro de una secuencia y no pueden depender del orden de los valores en una secuencia. A continuación se definen los operadores adaptados para funcionar en atributos valorados por conjuntos:

A == B	Verdadero si todos los valores en cada operando están presentes en el otro.
A !=B	Falso si todos los valores en cada operando están presentes en el otro.
A < B	Si todos los valores en A están en B y B contiene por lo menos otro valor que no está en A.

A <= B Verdadero si todos los valores en A están en B. (Sí A es un atributo con un único valor, este es el mismo que "A en B").

A > B Verdadero si todos los valores en B están en A y A contiene por lo menos otro valor que no está en B.

A >= B Verdadero si todos los valores en B están en A.

A ^ B Verdadero si cualquier valor en A está presente en B (la intersección es nula).

Además de estas operaciones, se definen dos funciones incorporadas que toman un conjunto o una secuencia como argumento y devuelven un valor único de ese conjunto o secuencia:

MAX(<conjunto o secuencia de valores>) Devuelve el valor más alto del conjunto o secuencia.

MIN(<conjunto o secuencia de valores>) Devuelve el valor más bajo del conjunto o secuencia.

Si no se puede obtener del conjunto un máximo o mínimo (porque los valores no sean numéricos) el valor devuelto será indeterminado y cualquier comparación con este valor indeterminado será evaluada como falsa.

7.4.2.3.3 Operadores de secuencia

Para soportar la inclusión de atributos valorados por secuencias en expresiones de filtro, se precisan operadores que funcionen con secuencias. Esta cláusula amplía el lenguaje de restricción del servicio de notificación definiendo operadores que funcionan con secuencias.

Para secuencias sólo se define un operador único, puesto que el único requisito es establecer concordancia de igualdad entre secuencias. El operador definido para trabajar con operandos valorados con secuencias es:

A % B Verdadero si A y B tienen el mismo número de valores y todos los valores en A concuerdan con los de B, en su orden.

La función incorporada MAX y MIN, definida en la cláusula anterior, puede también aplicarse a las secuencias.

7.4.3 Requisitos del servicio MOO

Esta cláusula resume los requisitos del servicio de operación de objetos múltiples.

(R) MOO-1 Un sistema gestionado instanciará por lo menos un objeto de servidor MOO. Asimismo, cada contexto de denominación de raíz local en un sistema tendrá por lo menos una vinculación de nombre para un objeto de servicio MOO. El valor de la cadena ID en esta vinculación identificará sencillamente al servidor, quizás con un valor similar a "MOO1". La cadena clase en la vinculación identificará la clase de objeto ("itut_q816::BasicMOOService" o una subclase).

(R) MOO-2 La interfaz soportada por el objeto u objetos del servidor MOO será la interfaz del servidor MOO "básica" descrita anteriormente y definida en la IDL CORBA en el anexo A.

(R) MOO-3 Facultativamente, la interfaz soportada por el objeto u objetos de servidor MOO puede ser la interfaz de servicio MOO "avanzada" descrita anteriormente y definida en la IDL CORBA en el anexo A.

(R) MOO-4 El objeto u objetos de servidor MOO soportarán por lo menos el lenguaje de restricción por defecto definido anteriormente para la especificación de filtros aunque puede soportar otras gramáticas. El lenguaje de restricción por defecto, identificado como "MOO 1.0" es el lenguaje de

restricción por defecto definido por el servicio de notificación CORBA pero ampliado como se ha descrito anteriormente:

- el filtrado de valores de atributo de objeto en lugar de valores de miembros de la estructura de notificación;
- concordancia de expresiones normales;
- filtrado de atributos que contienen conjuntos o secuencias de valores.

7.5 El servicio de latido

El servicio de latido se utiliza para verificar el funcionamiento de los canales de notificación de un sistema gestionado, así como la red de comunicación entre el sistema gestionado y el sistema de gestión. Envía periódicamente una pequeña notificación a un sistema de gestión interesado en recibirlo que identifica el sistema que emitió el latido, así como el canal de notificación a través del cual se emitió. Tras configurar este servicio, un sistema de gestión puede establecer entonces un filtro para notificaciones de latido en cualquiera de los canales que le interesa asegurar que están funcionando. Como estas notificaciones circulan a través de los mismos canales, soporte lógico y redes que la notificación de otros recursos, verifican periódicamente el funcionamiento de estos recursos.

El servicio de latido se halla consultándolo en el servicio de denominación.

El siguiente IDL (sin comentarios) describe la interfaz del servicio de latido.

```
interface Heartbeat {

        attribute SystemLabelType systemLabel;

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

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

}; // end of Heartbeat interface

interface Notifications {
…
        void heartbeat (
                in SystemLabelType      systemLabel,
                in ChannelIDType        channelID,
                in HeartbeatPeriodType  period,
                in GeneralizedTimeType  timeStamp
        );
…
}; // end of Notifications interface
```

Como puede verse, el servicio de latido tiene un atributo denominado *systemLabel*, y operaciones para fijar y obtener el periodo entre latidos. *SystemLabel* es un identificador suministrado por el usuario. Su uso previsto es permitir que un sistema de gestión inserte una etiqueta para identificar el sistema que provee el latido.

El servicio de latido emite periódicamente una notificación sobre cada canal de eventos que puede encontrar en el servicio buscador de canal. El servicio buscador de canal proporciona un listado de cada canal en el sistema, con un ID de canal para cada uno, así como información adicional sobre el uso del canal. (El buscador de canal no incluirá en la lista la notificación de latido como una de las notificaciones que maneja. Las notificaciones de latido no son enviadas por objetos gestionados, y se envían a todos los canales.) Al final de cada periodo, el servicio de latido envía una notificación por cada uno de los canales listados. La notificación enviada a cada canal incluye el ID de canal de ese canal.

El periodo entre latidos se controla utilizando la operación *periodSet*. El valor sometido a esta operación es el periodo, en segundos, que el servicio de latido espera entre la emisión de notificaciones. La actualización del periodo hace que el servicio emita inmediatamente una notificación con el nuevo valor del periodo, y comience un nuevo periodo. Fijar el periodo a cero hace que el servicio emita una notificación final con un valor de periodo nulo, y luego nada más (hasta que se fije de nuevo el periodo).

Cada notificación incluye el valor del atributo *systemLabel*, el ID del canal a través del cual se envió la notificación, el valor presente del periodo y una indicación de hora.

(R) HEARTBEAT-1 Un sistema gestionado puede instanciar al menos un objeto del servicio de latido. Si el servicio de latido es soportado, cada contexto de denominación raíz local en un sistema tendrá al menos una vinculación de nombres para un objeto del servicio de latido. El valor de la cadena ID en esta vinculación simplemente identificará al servidor, con un valor similar a "Heartbeat1". La cadena *kind* en la vinculación identificará la clase del objeto. ("itut_q816::Heartbeat").

(R) HEARTBEAT-2 El objeto u objetos de servidor de latido soportará la interfaz de latido arriba descrito y definido en el CORBA IDL en el anexo A. Se soportará la funcionalidad arriba descrita.

(R) HEARTBEAT-3 La actualización del periodo hará que el servicio entregue una notificación a todos los canales con el nuevo valor de periodo y comience luego un nuevo periodo. Fijar el periodo a cero hace que el servicio emita una notificación final con un valor de periodo nulo, y luego nada más (hasta que se fije de nuevo el periodo).

(R) HEARTBEAT-4 Hasta que el periodo cambie, las notificaciones de latido se enviarán a todos los canales una vez dentro de cada periodo. El tiempo entre notificaciones de latido enviadas a un canal nunca será superior al doble del periodo.

7.6 Otros servicios de soporte

Este marco prevé la necesidad de otros servicios de soporte de gestión de red pero reconoce que no resulta práctico incluirlos todos como parte de un documento de marco. Aunque resulta un tanto arbitrario decir donde se corta. Debido a que se centra en la RGT y a la necesidad de soportar modelos de información existentes, este marco incluye servicios iguales a los proporcionados por el protocolo CMIP y las capacidades de información de gestión RGT más básicas. De la misma forma que con CMIP, se espera que se definan servicios de soporte adicionales, probablemente en documentos separados.

8 Cumplimiento y conformidad

Esta cláusula define los criterios que deben cumplir otros documentos de normalización que pretendan cumplir este marco y las funciones que se tienen que implementar en sistemas que pretendan su conformidad con esta Recomendación.

8.1 Conformidad de sistema

8.1.1 Puntos de conformidad

Esta cláusula resume las funciones individuales descritas anteriormente en esta Recomendación. Estos puntos de conformidad se combinan en perfiles que declaren la conformidad con esta especificación.

1) Una implementación que declare conformidad con los requisitos del servicio de denominación tiene que:
   - Soportar la versión del servicio de denominación CORBA especificada en 5.2.
   - Soportar todos los requisitos del servicio de denominación especificados en 6.1.
2) Una implementación que declare conformidad con los requisitos del servicio de notificación tiene que:
   - Soportar:
     – la versión del servicio de notificación CORBA especificada en 5.2; o bien
     – la interfaz 3GPP NotificationIRPOperations especificada en [13].
   - Soportar todos los requisitos del servicio de notificación especificados en 6.2.

NOTA – Se requiere más estudio para identificar un subconjunto mínimo de capacidades del servicio notificación que deban ser soportadas para el cumplimiento del marco.

3) Una implementación que declare conformidad con los requisitos del servicio de registro cronológico de telecomunicaciones tiene que:
   - Soportar la versión del servicio de registro cronológico de telecomunicaciones CORBA especificada en 5.2.
   - Soportar todos los requisitos del servicio de registro cronológico especificados en 6.3.
4) Una implementación que declare conformidad con los requisitos del servicio de seguridad tiene que:
   - Soportar la versión del servicio de seguridad especificada en 5.2.
   - Soportar todos los requisitos del servicio de seguridad especificados en 6.5.
5) Una implementación que declare conformidad con los requisitos del servicio de transacción tiene que:
   - Soportar la versión del servicio de transacción CORBA especificada en 5.2.
   - Soportar todos los requisitos del servicio de transacción especificados en 6.6.
6) Una implementación que declare conformidad con el servicio buscador de factorías tiene que:
   - Soportar la interfaz del servicio buscador de factoría especificada en 7.1 y definida en el IDL CORBA del anexo A.
7) Una implementación que declare conformidad con el servicio buscador de canal tiene que:
   - Soportar la interfaz del servicio buscador de canal especificada en 7.2 y definida en el IDL CORBA del anexo A.
8) Una implementación que declare conformidad con el servicio terminador tiene que:
   - Soportar la interfaz del servicio terminador especificada en 7.3 y definida en el IDL CORBA del anexo A.
9) Una implementación que declare conformidad con el servicio MOO básico tiene que:
   - Soportar los requisitos obligatorios MOO descritos en 7.4.3.
10) Una implementación que declare conformidad con el servicio MOO avanzado tiene que:
   - Soportar los requisitos obligatorios y opcionales MOO descritos en 7.4.3.
11) Una implementación que declare conformidad con el servicio de latido debe:
   - Soportar la interfaz del servicio de latido descrita en 7.5 y definida en el IDL CORBA del anexo A.

8.1.2 Perfil básico de conformidad

Un sistema que declare conformidad con el perfil básico de UIT-T Q.816 soportará:

1) La versión de CORBA especificada en 5.2.
2) Los requisitos del servicio de denominación (véase el punto 1 de conformidades).
3) Los requisitos del servicio de notificación (véase el punto 2 de conformidades).
4) El servicio buscador de factorías (véase el punto 6 de conformidades).
5) El servicio buscador de canal (véase el punto 7 de conformidades).
6) El servicio terminador (véase el punto 8 de conformidades).
7) El Servicio MOO básico (véase el punto 9 de conformidades).

8.2 Directrices para la declaración de conformidad

Los usuarios de este marco tienen que prestar atención cuando escriban declaraciones de conformidad. Puesto que los módulos IDL se están utilizando como espacios de nombre, pueden dividirse entre ficheros, como lo permiten las reglas IDL OMG. Por lo tanto, cuando se está ampliando un módulo no cambiará su nombre. En su lugar, se añadirá simplemente un nuevo fichero IDL. Establecer sencillamente el nombre de un módulo en una declaración de conformidad, por lo tanto, no bastará para identificar un conjunto de interfaces IDL. La declaración de conformidad tiene que identificar un documento y un año de publicación para asegurar que se está identificando la versión correcta de IDL.

ANEXO A

Servicios soporte del marco IDL

```
/* This IDL code is intended to be stored in a file named "itut_q816.idl"
located in the search path used by IDL compilers on your system. */

#ifndef ITUT_Q816_IDL
#define ITUT_Q816_IDL

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

#pragma prefix "itu.int"

module itut_q816 {

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

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

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

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

// Data Types and Structures

```
/** ScopeChoice enumerates four possible choices for a scope.  A
scope may include just the named base object, the entire subtree
of object below and including the base object, the objects at a certain
level below the base object (level 1 objects are directly contained by
the base object), or all of the objects down to a level, including the
base object and the level.
*/
enum ScopeChoiceType {baseObjectOnly, wholeSubtree, individualLevel,
                      baseToLevel};

/** Scope is used to convey which objects contained under the base
object, if any, are to be included in the scope of a scoped and
filtered operation. A level does not make sense for the baseObjectOnly
and wholeSubtree choices, but does for the other two.  To illustrate
the difference between the two options that include a level, a
scope choice of individualLevel with level = 1 would include all
of the objects directly contained by the base object.  A scope choice
of baseToLevel with level = 1 would include all of the objects
directly contained by the base object, and the base object.
*/

union ScopeType switch (ScopeChoiceType)
{
        /* The baseObjectOnly and wholeSubtree cases carry no value. */
        case individualLevel:   /* fall through */
        case baseToLevel:       short level;
};


/** BaseAndScopeType combines the name of a base managed object with
a "scope" of objects contained below it.  */

struct BaseAndScopeType {
        NameType        base;
        ScopeType       scope;
};

/** BaseAndScopeSetType is a set of BaseAndScopeTypes.  */

typedef sequence <BaseAndScopeType> BaseAndScopeSetType;

/** ChannelIDType is a string used to contain a channel ID.  */

typedef string ChannelIDType;

/** EventSetType is a list of event types.  It is actually just a list
of strings.  The values of the strings are the names of the event types
(the strings that go in the "type_name" field of the structured event),
which are the same as the scoped names of the operation defined on the
Notifications interfaces to send the events.  For example:
itut_x780::Notifications::objectCreation  */

typedef sequence <ScopedNameType> EventSetType;

/** A channelInfo structure contains information about an event
channel.
@member channelID       A string identifier for the channel.
@member channelClass    the channel's scoped class name.
@member baseAndScopes   The objects and the scopes of managed objects
                        below them sending events to this channel.
                        A null list indicates that all base managed
                        objects on the system are covered by this
                        channel.
```

```
@member eventTypes      The list of event types handled by this
                        channel.  A null list indicates all event types
                        are handled by this channel.
@member excludedEventTypes If the eventTypes parameter is null, this
                        can be used to exclude event types.  If
                        eventTypes is not null, this should be null
                        and is ignored.
@member sourceClasses   The list of types of objects that send events
                        to this channel.  A null list indicates all
                        types of managed objects send events to this
                        channel.
@member excludedSourceClasses If the sourceClasses parameter is null,
                        this can be used to exclude source classes.
                        If sourceClasses is not null, this should be
                        null and is ignored.
@member channel         a reference to the channel.
*/

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

/** A channel info set contains a list of channel references and the
data associated with them.  */

typedef sequence <ChannelInfoType> ChannelInfoSetType;

/** ChannelModification indicates the type of event channel
modification.  */

enum ChannelModificationType {ChannelCreate, ChannelDelete,
                              ChannelUpdate};

/** The DeleteResultsType holds, for a single object, the results
of a scoped delete operation.  If both boolean flags in the result
are false, the object was deleted.
@member name            The name of the object to which these results
                        apply.
@member notFilterable   This flag will be true if the service could not
                        interact with the object to see if it even
                        passed the filter.
@member notDeletable    This flag will be true if the object could not
                        be deleted due to its delete policy or because
                        it raised an exception.
*/

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

/** The DeleteResultsSetType is a set of results returned by the
scoped delete operation.  */

typedef sequence <DeleteResultsType> DeleteResultsSetType;

/** A factory info structure contains information about a managed
object factory.
@member factoryClass    the factory's scoped class name
@member factoryRef      a reference to the factory
*/
```

```
struct FactoryInfoType {
        ObjectClassType                 factoryClass;
        itut_x780::ManagedObjectFactory factoryRef;
};
/** A factory info set contains a list of factory references and
their class names.  */

typedef sequence <FactoryInfoType> FactoryInfoSetType;

/** A FilterType parameter conveys the filter expression used in a
scoped and filtered operation.
*/
typedef string FilterType;

/** GetResults structures hold a list of attribute values per object.
@member name            The CORBA name of the object
@member notFilterable   This flag will be true if the service could not
                        interact with the object to see if it even
                        passed the filter.  If true, the attributes and
                        failedAttributes members will be empty.
@member attributes      The list of attributes retrieved from the
                        object.
@member failedAttributes The list of attributes whose values could
                        not be retrieved from the object.
*/

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

/** The GetResultsSet is a set of results returned by a scoped get
operation.  */

typedef sequence <GetResultsType> GetResultsSetType;

/** HeartbeatPeriodType contains the length of the interval between
heartbeats emitted by the Heartbeat Service.  Using an unsigned short
to contain this interval limits the longest possible interval to
a little over 18 hours.  */

typedef unsigned short HeartbeatPeriodType;

/** A LanguageType parameter conveys the filter expression language
used in a scoped and filtered operation.
*/

typedef string LanguageType;

/** A LanguageSetType parameter contains a sequence of Languages. */

typedef sequence <LanguageType> LanguageSetType;

/** ModificationOp is used to indicate the type of update to be made to
an attribute. */

enum ModificationOpType {set, add, remove};

/** Modification structures identify an attribute and a modification to
be made to it.  Multiple updates may be made to a single attribute by
including multiple structures with the same attribute name in the
modification Set.
@member attrib          The name of the attribute to update.
@member op              The operation to be performed on the attribute.
@member val             The value to be used for the update operation.
                        It's type will depend on the attribute.
*/
```

```
struct ModificationType {
        string                  attrib; // the name of the attribute
        ModificationOpType      op;     // operation to be performed
        any                     value;  // value used to update attrib.
};

/** The Modification Sequence contains a sequence of modifications to
be made (in order) to each object in a scoped update operation. */

typedef sequence <ModificationType> ModificationSeqType;

/** Update Results structures hold the name of an object, a boolean
flag indicating if all modifications to that object were successful,
and a list of the attributes that could not be updated on that object.
The list will be null if the success flag is true.
@member name            the CORBA name of the object
@member notFilterable   This flag will be true if the service could not
                        interact with the object to see if it even
                        passed the filter.  If true, the client will
                        know no attributes could be set, so
                        the failedAttributes member will be empty.
@member failedAttributes the list of attributes that were not
                        correctly updated.
*/

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

/** An Update Results Set is returned in response to a scoped update
operation (one that sets, adds to, or removes from the value of an
attribute). */

typedef sequence <UpdateResultsType> UpdateResultsSetType;
```

// Constants

```
/** Default filter is to allow everything through the filter*/

const FilterType defaultFilter = "TRUE";

/** Default language is the grammar described in this document */

const LanguageType defaultLanguage = "MOO 1.0";
```

// Exceptions

```
/**  A channel already registered exception is returned when an attempt
is made to register a channel with multiple channel IDs.  */

exception ChannelAlreadyRegistered {};

/**  A channel not found exception is returned when an event channel
cannot be found.  */

exception ChannelNotFound {};

/** A FactoryNotFound exception is raised when a requested factory
can't be found.  */

exception FactoryNotFound {};

/** A Filter Complexity Limit is raised when a filter expression in a
scoped operation is valid, but too complex to be processed.  */

exception FilterComplexityLimit {};
```

```
/** An invalid filter exception is raised when a client includes a
filter expression that cannot be parsed.  The text surrounding the
syntax error should be returned for trouble-shooting purposes. */

exception InvalidFilter {string badText;};

/** An Invalid Parameter exception is raised when the value of a
parameter is not valid for the operation.
@param parameter        the name of the bad parameter
*/

exception InvalidParameter {string parameter;};
```

// Interfaces

// Factory Finder Interface

```
/**
This interface defines a simple service for locating a managed object
factory.
*/

interface FactoryFinder {

        /** This method is used to find a managed object factory.
        @param factoryClass     The scoped class name of the factory,
                                NOT the managed object to be created.
        */

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

        /** This method returns the list of factories registered
        with the factory finder. */

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

}; // end of FactoryFinder interface

/**
This interface extends the FactoryFinder interface to add methods
to support the registration and unregistration of factories.
*/

interface FactoryFinderComponent : FactoryFinder {

        /** This method is used by factories to register themselves.
        It should not be used by managing systems.
        @param factoryClass     The scoped class name of the factory,
                                NOT the managed object to be created.
        @param factoryRef       A reference to the factory.
        */

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

        /** This method is used by factories to unregister themselves,
        if necessary.  It should not be used by managing systems.
        @param factoryClass     The scoped class name of the factory,
                                NOT the managed object to be created.
        @param factoryRef       A reference to the factory.
        */

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

}; // end of FactoryFinderComponent interface
```

// Channel Finder Interface

```
        /**
        This interface defines a simple service for locating event channels.
        */

        interface ChannelFinder {

                /** This method returns the list of channels registered
                with the channel finder.  */

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

        }; // end of ChannelFinder interface


        /**
        This interface extends the ChannelFinder interface to add methods
        to support the registration and unregistration of channels.
        */

        interface ChannelFinderComponent : ChannelFinder {

                /** This method is used by channels to register themselves.
                It should not be used by managing systems.  Re-registering
                a channel (re-using an existing channelID) results in
                updating that entry.  The other information previously
                associated with that entry is overwritten.  A
                ChannelAlreadyRegistered exception may be raised when an
                attempt is made to register a channel with multiple channelIDs.
                This should not be done.  (The service cannot guarantee that
                because two object references differ, they do not reference
                the same object.  It is therefore required that the managed
                system ensure that the same channel is not registered twice.)
                A channel change notification is sent whenever calling this
                method results in a change.
                @param channelID        A string identifier for the channel.
                @param channelClass     The scoped class name of the event
                                        channel.
                @param baseAndScopes    The objects and the scopes of managed
                                        objects below them sending events to
                                        this channel.  A null list indicates
                                        that all base managed objects on the
                                        system are covered by this channel.
                @param eventTypes       The list of event types handled by
                                        this channel. A null list indicates one
                                        of the following:
- for event channel interfaces that do not provide an explicit query for the events types
handled by the channel (e.g. the OMG Notification channel),it implies all event types are
handled by this channel
- for event channel interfaces that do provide an explicit query for the event types
handled by the channel (e.g. 3GPP NotificationIRPOperations interface), it implies that
the explicit query must be used to determime the types of events handled.


                @param excludedEventTypes If the eventTypes parameter is null,
                                        this can be used to exclude event
                                        types.  If eventTypes is not null, this
                                        should be null and is ignored.
                @param sourceClasses    The list of types of objects that send
                                        events to this channel.  A null list
                                        indicates all types of managed objects
                                        send events to this channel.
                @param excludedSourceClasses If the sourceClasses parameter is
                                        null, this can be used to exclude
                                        source classes.  If sourceClasses is
                                        not null, this should be null and is
                                        ignored.
                @param channel          A reference to the channel.
                */
```

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

        /** This method is used by managed systems to unregister
        channels, if necessary.  It should not be used by managing
        systems.
        @param channelID        A string identifier for the channel.
        */

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

};  // end of ChannelFinderComponent interface
```

// Heartbeat Service Interface

```
/**
This interface defines a service used to periodically test the
operation of the notification channels on a system.  The service
supporting this interface periodically emits a short "heartbeat"
notification on each channel on the system.
*/

interface Heartbeat {

        /** The systemLabel attribute is sent in heartbeat
        notifications.  It is used to identify the heartbeat service
        instance from which the notification came.  Resetting this does
        not cause the service to immediately emit a notification, but
        the new value will be sent with the next notification.  */

        attribute SystemLabelType systemLabel;

        /**  The period is the interval, in seconds, at which the
        heartbeat service emits the heartbeat notification.  If it is
        zero, the service does not emit notifications.  */

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

        /** Updating of the period shall cause the service to deliver a
        notification to all channels with the new period value and then
        begin a new period.  Setting the period to zero shall cause the
        service to emit one final notification with a period value of
        zero, then no more (until the period is reset).
        Implementations may limit the range of values supported by this
        operation, particularly on the low end as excessive heartbeats
        would present a drain on the managed system.  An attempt to
        set the period to a value outside the range supported will
        result in an ApplicationError with the error code set to
        invalidParameter.  */

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

};  // end of Heartbeat interface
```

// Terminator Service Interface

```
/**
This interface defines a service that supports the deletion of managed
objects by clients.  A goal of the framework is to enable
implementations in which the managed objects do not have to maintain
the naming tree information.  The factories are one place to implement
the functions needed to create name bindings, and this service can be
used to clean up the naming tree after object deletion.  <p>

Also, this service can implement the rules for deleting objects based
on the delete policy of the managed objects.
*/

interface TerminatorService {

        /** This method is used to delete a managed object by
        specifying its name.  */

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

        /** This method is used to delete a managed object by
        reference.  */

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

}; // end of TerminatorService interface
```

// DeleteResultsIterator Interface

```
/** The Delete Results Iterator interface is used to retrieve the
results from a scoped delete operation using the iterator design
pattern. */

interface DeleteResultsIterator {

        /** This method is used to retrieve the next "howMany" results
        in the result set.
        @param howMany  The maximum number of items to be returned in
                        the results.
        @param results  The next batch of results.
        @return         True if there are more results after those
                        being returned.  The return value should not
                        be true if the results set is empty, as this
                        forces the client to poll for results.
                        Instead the call should block.
        */

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

        /** This method is used to destroy the iterator and release its
        resources.  */

        void destroy();

}; // end of Delete Results Iterator interface
```

// GetResultsIterator Interface

```
/** The Get Results Iterator interface is used to retrieve the results
from a scoped get operation using the iterator design pattern. */
```

```
interface GetResultsIterator {

        /** This method is used to retrieve the next "howMany" results
        in the result set.
        @param howMany  The maximum number of items to be returned in
                        the results.
        @param results  The next batch of results.
        @return         True if there are more results after those
                        being returned.  The return value should not
                        be true if the results set is empty, as this
                        forces the client to poll for results.
                        Instead the call should block.
        */

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

        /** This method is used to destroy the iterator and release its
        resources.  */

        void destroy();

}; // end of Get Results Iterator interface
```

// UpdateResultsIterator Interface

```
/** The Update Results Iterator interface is used to retrieve the
results from a scoped update (set, add, remove) operation using the
iterator design pattern.
*/

interface UpdateResultsIterator {

        /** This method is used to retrieve the next "howMany" results
        in the result set.
        @param howMany  The maximum number of items to be returned in
                        the results.
        @param results  The next batch of results.
        @return         True if there are more results after those
                        being returned.  The return value should not
                        be true if the results set is empty, as this
                        forces the client to poll for results.
                        Instead the call should block.
        */

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

        /** This method is used to destroy the iterator and release its
        resources.  */

        void destroy();

}; // end of Update Results Iterator interface
```

// BasicMooService Interface

```
/** The basic scoping and filtering interface provides a common service
for performing attribute retrieval operations on multiple objects.
*/

interface BasicMooService {

        /** This operation is used to retrieve the list of filter
        languages supported by the service.  At the least, the
        list must include the value of the defaultLanguage constant
        defined above.  */
```

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

        /** This operation is used to retrieve attributes from multiple
        objects using a small number of method invocations.  The method
        returns the first batch of results, one per object.  Each
        result has the name of the object and a list of name-value
        pairs indicating the attributes that could be retrieved with
        their values.

        @param baseName The name of the object at the base of the scope
                        tree.
        @param scope    A value indicating the contained objects to
                        include in the scope of the operations. See
                        ScopeType for details.
        @param filter   A string containing an expression to be
                        evaluated with attribute values from each
                        object in the scope.  Attribute values are
                        returned only for those objects for which the
                        expression evaluates to true.
        @param language A string identifying the language in which the
                        filter expression is written.
        @param attributes The names of the attributes for which values
                        should be returned.  If this list is null, all
                        attributes are to be returned.
        @param howMany  The maximum number of objects for which results
                        should be returned in the first batch.
        @param resultsIterator A reference to an iterator that can be
                        used to retrieve the rest of the results.  This
                        reference will be null if all results were
                        returned in the first batch.
        */

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

};  // end of BasicMooService interface
```

// AdvancedMooService Interface

```
/** The advanced scoping and filtering interface provides a common
service for performing multiple-attribute updates on multiple objects,
and for deleting multiple objects.
*/

interface AdvancedMooService : BasicMooService {

        /** This operation is used to modify multiple attributes in
        multiple objects using a small number of method invocations.
        The method returns the first batch of results, a list of
        objects for which one or more modifications failed.  Each
        result indicates the attribute(s) on that object that could not
        be updated.

        @param baseName The name of the object at the base of the scope
                        tree.
        @param scope    A value indicating the contained objects to
                        include in the scope of the operations. See
                        ScopeType for details.
```

```
@param filter   A string containing an expression to be
                evaluated with attribute values from each
                object in the scope.  Updates are performed
                only on those objects for which the expression
                evaluates to true.
@param language A string identifying the language in which the
                filter expression is written.
@param modifications The list of modifications to be made to
                each object.
@param failuresOnly If true only results for failed objects
                will be returned.
@param howMany  The maximum number of objects for which results
                should be returned in the first batch.
@param resultsIterator A reference to an iterator that can be
                used to retrieve the rest of the results.  This
                reference will be null if all results were
                returned in the first batch.
*/


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


/** This operation is used to delete multiple objects using a
small number of method invocations.  The method returns the
first batch of results, a list of the objects that could not be
deleted.


@param baseName The name of the object at the base of the scope
                tree.
@param scope    A value indicating the contained objects to
                include in the scope of the operations. See
                ScopeType for details.
@param filter   A string containing an expression to be
                evaluated with attribute values from each
                object in the scope.  Only those objects for
                which the expression evaluates to true are
                deleted.
@param language A string identifying the language in which the
                filter expression is written.
@param failuresOnly If true only results for failed objects
                will be returned.
@param howMany  The maximum number of objects for which results
                should be returned in the first batch.
@param resultsIterator A reference to an iterator that can be
                used to retrieve the rest of the results.  This
                reference will be null if all results were
                returned in the first batch.
*/


DeleteResultsSetType scopedDelete (
        in NameType baseName,
        in ScopeType scope,
        in FilterType filter,
        in LanguageType language,
        in boolean failuresOnly,
```

```
                in unsigned short howMany,
                out DeleteResultsIterator resultsIterator)
                raises (InvalidParameter,
                        InvalidFilter,
                        FilterComplexityLimit,
                        itut_x780::ApplicationError);

};  // end of AdvancedMooService interface
```

// Notifications Interface

```
/**  The notifications interface defines the notifications emitted by
the framework services, not the managed objects themselves.
*/

interface Notifications {

        /** The Channel Change notification is a special notification
        because it is emitted by the framework (the Channel Finder) and
        not a managed object.  It reports the addition, deletion, or
        change of a registered event channel.
        @param channelModification      indicates if a channel has been
                                        added, removed, or updated.
        @param channelInfo              provides the information about
                                        the affected channel.
        */

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

        /** This operation signature defines the notification emitted
        by the heartbeat service.
        @param systemLabel      the current value of the Heartbeat
                                service systemLabel attribute.
        @param channelID        the ID of the channel through which the
                                notification was sent.
        @param period           the current value of the Heartbeat
                                service period attribute.
        @param timeStamp        the current time when the notification
                                is emitted.
        */

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

        /** These constants defines the names of the notification
        declared above and are provided to help reduce errors.  */

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

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

        /** These constants define the names of the parameters used in
        the notifications declared above and are provided to help
        reduce errors.  */

        const string channelIDName = "channelID";
        const string channelModificationName = "channelModification";
        const string channelInfoName = "channelInfo";
        const string periodName = "period";
        const string systemLabelName = "systemLabel";
        const string timeStampName = "timeStamp";
};  // end of Notifications interface
```

```
}; // end of module itut_q816

#endif  // end of #ifndef ITUT_Q816_IDL
```

ANEXO B

Lenguaje limitado BNF

El BNF en este anexo es una extensión del BNF utilizado por el lenguaje de filtro del servicio de notificación OMG. Las extensiones se muestran en negritas con carácter más grande.

B.1 Lenguaje de restricción adecuado en términos de testigos léxicos

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

```
        | <seq_literal>
<seq_literal> := { <factor_list> }
<factor_list> := /*empty*/
        |  <factor_list> , <factor>
        | <factor>
```

B.2 "BNF" para testigos léxicos hasta aspectos del juego de caracteres

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

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

B.3 Aspectos del juego de caracteres

El BNF anterior ha sido completado hasta los no terminales <Leader>, <Follow>, <Alpha>, <Digit> y <Other>. Para un conjunto de caracteres particular, se deben definir los caracteres que conforman esas clases de caracteres.

Cada juego de caracteres cuyo servicio comercial debe soportar debe definir esas clases de caracteres. Este apéndice define esas clases de caracteres para el conjunto de caracteres ASCII.

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

APÉNDICE I

Casos de interfuncionamiento entre modelos que utilizan el marco UIT y los modelos ADSL/ATMF

I.1 Introducción

Este apéndice describe cómo sistemas diseñados para utilizar el planteamiento de este marco pueden interfuncionar con sistemas diseñados para utilizar el planteamiento especificado en el Foro ATM y en el Foro ADSL.

Los planteamientos siguientes se pueden utilizar para definir interfaces para objetos gestionados basados en CORBA:

– Un modelo granulado fino tiene ya una relación de uno a uno entre instancias de interfaz CORBA (es decir, que tiene su propia referencia de objeto interoperable, IOR) e instancias de objetos gestionados.

– Un modelo granulado de clases tiene una interfaz CORBA para cada clase de objeto gestionado. Algún otro mecanismo, (como situar el nombre de objeto gestionado como parámetro de entrada para cualquier operación en dicha interfaz) tiene que ser soportado por la interfaz granulada por clases CORBA para permitir la gestión de cada instancia de objeto gestionado.

Un planeamiento denominado de gránulo neutro utiliza una estructura que incluye tanto el nombre del objeto gestionado como la IOR utilizada para proporcionar acceso a cada objeto gestionado. Cabe destacar que el planteamiento de gránulo neutro aunque requiere que el cliente pase la instancia del objeto gestionado como un parámetro para cada operación (es decir, parece granulado por clases al cliente), permite que la implementación al servidor sea granulada por clases o de granulado fino.

El marco en esta Recomendación utiliza un planteamiento de granulado fino junto con una operación *attributesGet* basada en valor-tipo, para definir objetos gestionados basados en CORBA. El tipo de valor asociado con una subclase de objeto gestionado CORBA utiliza herencia de tipo de valor para ampliar los elementos del tipo de valor asociado con su superclase.

Las especificaciones de los foros ATM y ADSL incluyen la IOR para la clase y el nombre del objeto gestionado como una lista de parámetros emparejados para las operaciones (es decir, son de granulado neutro). El cliente utiliza el nombre para hacer referencia a cada entidad. Estas especificaciones también utilizan estructuras (es decir, no emplean herencia) para sus operaciones *attributesGet*.

I.2 Terminología

Se introducen los términos siguientes para que sean tratados:

– Servidor de gránulo neutro – Sistema gestionado que implementa objetos definidos utilizando un modelo de gránulo neutro con CORBA 2.1.

– Servidor de marco UIT – Un sistema gestionado que implementa objetos definidos utilizando el marco en esta Recomendación, que soporta por lo tanto características CORBA 2.3.

– Sistema de gestión de gránulo neutro – Un cliente capaz de gestionar objetos CORBA definidos utilizando un modelo de gránulo neutro con CORBA 2.1.

– Sistema de gestión de marco UIT – Un cliente capaz de gestionar objetos CORBA definidos de conformidad con el marco de la presente Recomendación, que soporta por lo tanto características CORBA 2.3.

I.3 Casos de interfuncionamiento

I.3.1 Servidor de gránulo neutro que se desplaza al servidor de UIT

Para desplazarse desde un servidor de gránulo neutro, un servidor de marco UIT tiene que añadir nuevas capacidades; sin embargo, tiene que mantener las antiguas capacidades que se encuentran en la versión de gránulo neutro.

El nuevo servidor implementará una función adaptador. Esta función presentará las interfaces de gránulo neutro a sistemas de gestión existentes. Un planteamiento de implementación puede utilizar delegación de operaciones invocadas en objetos de gránulo neutro a los objetos construidos de conformidad con el marco. Este tipo de planteamiento de delegación emitirá las mismas operaciones en los objetos individuales que serían invocadas por un marco UIT basado en el sistema de gestión.

Todos los parámetros de operación de atributo se convertirán al tipo de valor estructurado como en el marco de esta Recomendación.

El soporte lógico de interfuncionamiento debe realizar funciones más allá de las diferencias resultantes de la utilización de POA y de objetos de valor. El soporte lógico de delegación debe estar adaptado para considerar las diferencias en la estructura de denominación, incluido el uso en este marco del campo *Kind*. Por ejemplo, explicar cómo la referencia de objeto de contexto utilizada al construir el nombre debe ser sustituida por el campo *Kind* de la estructura de denominación COS.

I.3.2 Cliente de gránulo neutro que se desplaza al cliente de marco UIT

Este tipo de cliente de marco UIT necesita tratar tanto servidores de gránulo neutro como servidores de marco UIT.

El cliente de marco UIT con CORBA 2.3 necesita, además de utilizar el árbol de denominación para servidores del marco UIT, utilizar el árbol de denominación para servidores de gránulo neutro. La implementación CORBA 2.3 puede necesitar una función de adaptación en la que la aplicación que emite la petición a un objeto tiene que ser convertida para adaptarse al servidor de gránulo neutro. Si la aplicación utiliza el tipo valor tiene que ser descompuesta en la operación get específica de objeto para el antiguo servidor.

Para que un cliente basado en el marco UIT gestione tanto servidores basados en el marco UIT como servidores del marco pre-UIT, el cliente UIT necesitará soportar cepas tanto para servidores UIT como pre-UIT. Además, se necesitará alguna función de interfuncionamiento, como se ha indicado anteriormente y como se muestra en la figura siguiente.

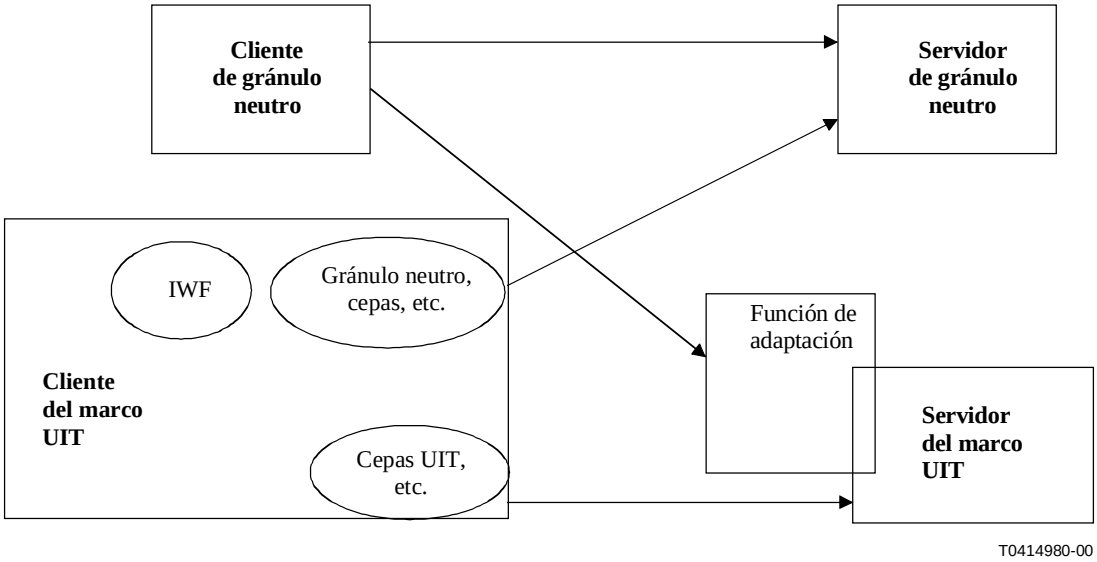

Figura I.1/Q.816 – Casos de interfuncionamiento

APÉNDICE II

Filtrado de eventos estructurados originales y trasladados

Esta Recomendación define notificaciones que utilizan signaturas de operación IDL, con el tipo de evento como nombre de operación, y con un parámetro *in* separado de la operación para cada elemento de datos de la notificación. Estas signaturas de operación se pueden utilizar para notificaciones depositadas directamente en un canal a través de consumidores mandatarios tipificados, como se define en el servicio de notificación OMG. Esto se muestra en la flecha identificada como (1) en la figura II.1.

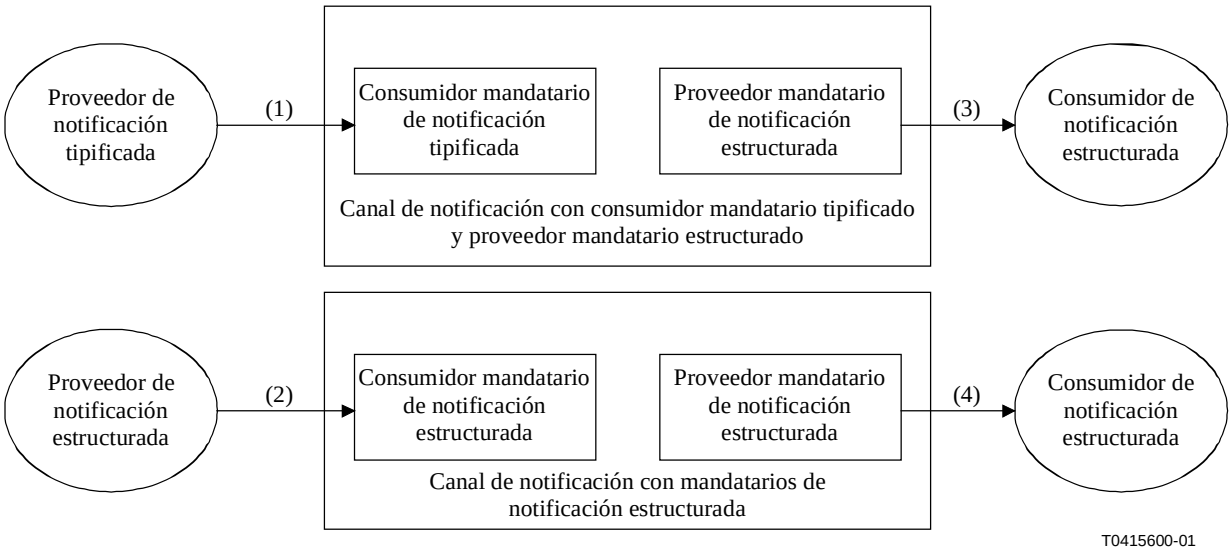

Figura II.1/Q.816 – Ejemplo de depósito de notificación tipificada y estructurada, con entrega estructurada

En razón de que los eventos estructurados están ampliamente soportados, esta Recomendación también proporciona un algoritmo para definir un informe de evento estructurado para cada una de las notificaciones definidas. Este formato definido se utiliza para notificaciones estructuradas originales depositadas por proveedores a consumidores mandatarios estructurados sobre canales, como se muestra en la flecha marcada (2) de la figura II.1, y para entregas no trasladadas, como se muestra en la flecha marcada (4) de la figura II.1.

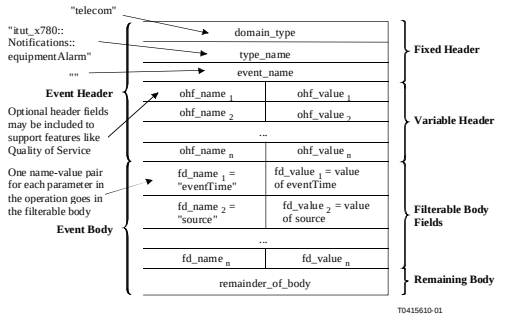

Figura X-2- Notificación estructurada original

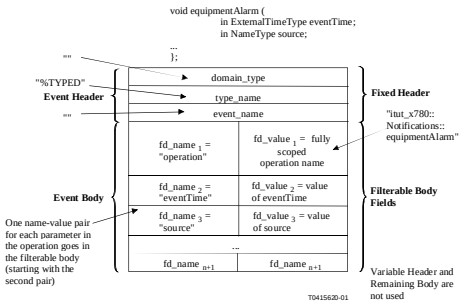

Figura X-3 - Notificación tipificada trasladada a notificación estructurada

Estos eventos estructurados originales se determinan según definiciones de notificación tipificadas, como sigue (resumidas en la figura 6):

1) El miembro de datos domain_type se fija en "Telecommunications".

2) El miembro de datos type_name se fija en la denominación completa de la operación que define el tipo de evento, por ejemplo, "itu_X780::Notifications::communicationsAlarm".

3) El miembro de datos event_name se fija en cadena vacía.

4) Los elementos de nombre de los pares valor-nombre del campo del cuerpo filtrable se fijan para indicar el nombre de cada parámetro (por ejemplo, "eventTime", "source", "sourceClass", etc.). El elemento de nombre n-ésimo contiene el nombre del n-ésimo parámetro.

5) Los elementos de valor de los pares valor-nombre del campo del cuerpo filtrable se fija para indicar el valor transmitido al parámetro.

Un ejemplo de un evento estructurado original, que pertenece a la correspondencia estándar, que utiliza X.780 normal, tendrá la siguiente estructura:

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

El servicio de notificación OMG define procedimientos para que los canales de notificación conviertan automáticamente los eventos tipificados recibidos de los proveedores (véase la flecha marcada (1) de la figura II.1) en notificaciones estructuradas para entrega a los consumidores (véase la flecha marcada (2) de la figura II.1), resumida como sigue:

1) El miembro de datos domain_type se fija en cadena vacía.

2) El miembro de datos type_name se fija en la cadena "%TYPED".

3) El miembro de datos event_name se fija en cadena vacía.

4) El elemento nombre del primer par nombre-valor del campo del cuerpo filtrable se fija en la cadena "operation".

5) El elemento valor del par nombre-valor del campo del cuerpo filtrable se fija en una cadena que contiene el nombre de la operación totalmente delimitada. En este caso, por ejemplo, "itu_X780::Notifications::communicationsAlarm".

6) Los elementos de nombre restantes de los pares nombre-valor del campo del cuerpo filtrable se fijan para indicar el nombre de cada parámetro (como por ejemplo, "eventTime", "source", "sourceClass", etc.). El n-ésimo elemento de nombre contiene el nombre del parámetro n-ésimo –1.

7) Los elementos de valor restantes de los pares nombre-valor del campo del cuerpo filtrable se fijan para indicar el valor que fue pasado para cada parámetro (como por ejemplo, valor de eventTime, valor de source, valor de sourceClass, etc.). El n-ésimo elemento de valor contiene el valor del n-ésimo parámetro.

Utilizando el mismo ejemplo que el indicado anteriormente para la notificación estructurada original, para un evento tipificado trasladado, el canal ha de distribuir algo similar a lo siguiente (donde fd_namei: y fd_valuei indican el nombre y el valor del i-ésimo elemento de la lista del par nombre-valor de datos filtrables):

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

Cabe señalar que el nombre de la operación de delimitación completa se inserta como el primer elemento de la lista del par nombre-valor de datos filtrables, y cada parámetro *in* de la operación de evento tipificado es un elemento subsiguiente.

Se indica a continuación un ejemplo de descripción de un constructivo de filtro, que utiliza variables de ejecución, que se probarían para el tipo de evento `itut_x780::Notifications::attributeValueChange` y que funcionarían para ambas formas del evento estructurado (tipificado original y traslado del canal) precedente:

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

or

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

APÉNDICE III

Bibliografía

La siguiente Recomendación contiene información que fue utilizada para el desarrollo de este marco. Como se indicó en la introducción, el objetivo de diseño primordial de este marco es permitir la reutilización de los modelos de información de gestión de red existentes, al menos sin modificaciones semánticas significativas. Estas Recomendaciones proporcionan muchos de los detalles del marco CMIP del UIT-T, y por lo tanto, define algunas de las funcionalidades que debe soportar el marco CORBA.

– UIT-T M.3010 (2000), *Principios para una red de gestión de las telecomunicaciones*.

– UIT-T M.3120 (2001), *Modelo de información de red genérico basado en la arquitectura de negociación de petición de objetos comunes y a nivel de elemento de red*.

– UIT-T Q.821 (2000), *Descripción de las etapas 2 y 3 para la interfaz Q3 – Vigilancia de alarmas*.

– UIT-T X.703 (1997), *Tecnología de la información – Arquitectura de gestión distribuida abierta*.

– UIT-T X.710 (1997) | ISO/CEI 9595:1998, *Tecnología de la información – Interconexión de sistemas abiertos – Servicio común de información de gestión*.

– UIT-T X.711 (1997) | ISO/CEI 9596-1:1998, *Tecnología de la información – Interconexión de sistemas abiertos – Protocolo común de información de gestión: Especificación*.

– UIT-T X.711/Cor.2 (2000) *Tecnología de la información – Interconexión de sistemas abiertos – Protocolo común de información de gestión: Especificación – Corrigendum técnico 2: Revisión del protocolo común de información de gestión para incluir ASN.1:1997*.

– UIT-T X.720 (1992) | ISO/CEI 10165-1:1993, *Tecnología de la información – Interconexión de sistemas abiertos – Estructura de la información de gestión: Modelo de la información de gestión.*

– UIT-T X.720/Cor.1 (1994) *Tecnología de la información – Interconexión de sistemas abiertos – Estructura de la información de gestión: Modelo de la información de gestión – Corrigendum técnico 1*.

– UIT-T X.721 (1992) | ISO/CEI 10165-2:1992, *Tecnología de la información – Interconexión de sistemas abiertos – Estructura de la información de gestión: Definición de la información de gestión.*

– UIT-T X.721/Cor.1 (1994) *Tecnología de la información – Interconexión de sistemas abiertos – Estructura de la información de gestión: Definición de la información de gestión – Corrigendum técnico 1*.

– UIT-T X.721/Cor.2 (1996) *Tecnología de la información – Interconexión de sistemas abiertos – Estructura de la información de gestión: Definición de la información de gestión – Corrigendum técnico 2*.

– UIT-T X.722 (1992) | ISO/CEI 10165-4:1992, *Tecnología de la información – Interconexión de sistemas abiertos – Estructura de la información de gestión: Directrices para la definición de objetos gestionados.*

– UIT-T X.722/Enm.1 (1995), *Tecnología de la información – Interconexión de sistemas abiertos – Estructura de la información de gestión: Directrices para la definición de objetos gestionados – Enmienda 1: Registro de "Set by create" y componentes.*

– UIT-T X.722/Enm.2 (1997), *Tecnología de la información – Interconexión de sistemas abiertos – Estructura de la información de gestión: Directrices para la definición de objetos gestionados – Enmienda 2: Adición del elemento de sintaxis NO-MODIFY y amplicación de directrices.*

– UIT-T X.722/Enm.3 (1997), *Tecnología de la información – Interconexión de sistemas abiertos – Estructura de la información de gestión: Directrices para la definición de objetos gestionados – Enmienda 3: Directrices para la utilización de Z en la formalización del comportamiento de objetos gestionados.*

– UIT-T X.722/Cor.1 (1996), *Tecnología de la información – Interconexión de sistemas abiertos – Estructura de la información de gestión: Directrices para la definición de objetos gestionados – Corrigendum técnico 1*.

– UIT-T X.722/Cor.2 (2000), *Tecnología de la información – Interconexión de sistemas abiertos – Estructura de la información de gestión: Directrices para la definición de objetos gestionados – Corrigendum técnico 2: Revisión de las directrices para la definición de objetos gestionados para incluir ASN1:1997.*

– UIT-T X.733 (1992) | ISO/CEI 10164-4:1992, *Tecnología de la información – Interconexión de sistemas abiertos – Gestión de sistemas: Función señaladora de alarmas*.

SERIES DE RECOMENDACIONES DEL UIT-T

Serie A	Organización del trabajo del UIT-T
Serie B	Medios de expresión: definiciones, símbolos, clasificación
Serie C	Estadísticas generales de telecomunicaciones
Serie D	Principios generales de tarificación
Serie E	Explotación general de la red, servicio telefónico, explotación del servicio y factores humanos
Serie F	Servicios de telecomunicación no telefónicos
Serie G	Sistemas y medios de transmisión, sistemas y redes digitales
Serie H	Sistemas audiovisuales y multimedios
Serie I	Red digital de servicios integrados
Serie J	Redes de cable y transmisión de programas radiofónicos y televisivos, y de otras señales multimedios
Serie K	Protección contra las interferencias
Serie L	Construcción, instalación y protección de los cables y otros elementos de planta exterior
Serie M	RGT y mantenimiento de redes: sistemas de transmisión, circuitos telefónicos, telegrafía, facsímil y circuitos arrendados internacionales
Serie N	Mantenimiento: circuitos internacionales para transmisiones radiofónicas y de televisión
Serie O	Especificaciones de los aparatos de medida
Serie P	Calidad de transmisión telefónica, instalaciones telefónicas y redes locales
Serie Q	**Conmutación y señalización**
Serie R	Transmisión telegráfica
Serie S	Equipos terminales para servicios de telegrafía
Serie T	Terminales para servicios de telemática
Serie U	Conmutación telegráfica
Serie V	Comunicación de datos por la red telefónica
Serie X	Redes de datos y comunicación entre sistemas abiertos
Serie Y	Infraestructura mundial de la información y aspectos del protocolo Internet
Serie Z	Lenguajes y aspectos generales de soporte lógico para sistemas de telecomunicación

Ginebra, 2001