International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Q.816.2

(03/2007)

SERIES Q: SWITCHING AND SIGNALLING

Q3 interface

CORBA-based TMN services: Extensions to support service-oriented interfaces

ITU-T Recommendation Q.816.2

ITU-T Q-SERIES RECOMMENDATIONS

SWITCHING AND SIGNALLING

SIGNALLING IN THE INTERNATIONAL MANUAL SERVICE	Q.1–Q.3
INTERNATIONAL AUTOMATIC AND SEMI-AUTOMATIC WORKING	Q.4–Q.59
FUNCTIONS AND INFORMATION FLOWS FOR SERVICES IN THE ISDN	Q.60–Q.99
CLAUSES APPLICABLE TO ITU-T STANDARD SYSTEMS	Q.100–Q.119
SPECIFICATIONS OF SIGNALLING SYSTEMS No. 4, 5, 6, R1 AND R2	Q.120–Q.499
DIGITAL EXCHANGES	Q.500–Q.599
INTERWORKING OF SIGNALLING SYSTEMS	Q.600–Q.699
SPECIFICATIONS OF SIGNALLING SYSTEM No. 7	Q.700–Q.799
Q3 INTERFACE	**Q.800–Q.849**
DIGITAL SUBSCRIBER SIGNALLING SYSTEM No. 1	Q.850–Q.999
PUBLIC LAND MOBILE NETWORK	Q.1000–Q.1099
INTERWORKING WITH SATELLITE MOBILE SYSTEMS	Q.1100–Q.1199
INTELLIGENT NETWORK	Q.1200–Q.1699
SIGNALLING REQUIREMENTS AND PROTOCOLS FOR IMT-2000	Q.1700–Q.1799
SPECIFICATIONS OF SIGNALLING RELATED TO BEARER INDEPENDENT CALL CONTROL (BICC)	Q.1900–Q.1999
BROADBAND ISDN	Q.2000–Q.2999
SIGNALLING REQUIREMENTS AND PROTOCOLS FOR THE NGN	Q.3000–Q.3999

For further details, please refer to the list of ITU-T Recommendations.

ITU-T Recommendation Q.816.2

CORBA-based TMN services: Extensions to support service-oriented interfaces

Summary

ITU-T Recommendation Q.816.2 defines a set of TMN CORBA services required to support service-oriented interfaces. It specifies how the ORB and common object services should be used in a lightweight fashion for supporting service-oriented interfaces, and defines extensions to the TMN-specific support services defined in ITU-T Recommendations Q.816 and Q.816.1. A CORBA IDL module defining the interfaces to the new TMN-specific support services is provided. The new services and the lightweight use of other CORBA services, along with ITU-T Recommendation X.780.2, compose a framework for CORBA-based service-oriented TMN interfaces with a wide range of applications.

Source

ITU-T Recommendation Q.816.2 was approved on 16 March 2007 by ITU-T Study Group 4 (2005-2008) under the ITU-T Recommendation A.8 procedure.

Keywords

Common object request broker architecture (CORBA), CORBA services, distributed processing, façade, interface definition language (IDL), managed object, managed system, managing system, service orientation, service-oriented façade object, TMN interface.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure e.g. interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

CONTENTS

		Page
1	Scope	1
	1.1 Purpose	2
	1.2 Application	2
2	References	3
3	Definitions	5
4	Abbreviations	7
5	Conventions	10
6	Service-oriented interface design considerations	10
	6.1 Flexible use of façade design pattern	11
	6.2 Lightweight use of ORB	12
	6.3 Use of session service by managed and managing systems	15
	6.4 Lightweight use of naming service	15
	6.5 Lightweight use of notification service	18
	6.6 Lightweight use of telecom log service	20
	6.7 Relationship to coarse-grained interface design considerations	22
7	Service-oriented framework and requirements overview	22
	7.1 Framework overview	22
	7.2 Framework constituents overview	24
	7.3 Relationships between service-oriented, coarse-grained and fine-grained frameworks	30
8	Framework ORB and common object services usage requirements for supporting service-oriented interfaces	32
	8.1 ORB usage requirements	33
	8.2 Naming service	33
	8.3 Notification service	34
	8.4 Telecom log service	40
	8.5 Concurrency control and object transaction services	41
	8.6 CORBAsecurity	42
9	Framework support services requirements for supporting service-oriented interfaces	44
	9.1 Session service	44
	9.2 Other ITU-T support services	49
10	Service-oriented compliance and conformance	49
	10.1 Standards document compliance	49
	10.2 System conformance	50
	10.3 Conformance statement guidelines	53

Page

Annex A – Service-oriented framework support services IDL 54

A.1 Module `idlVersion` 55

A.2 Module `session` 55

A.3 Module `nmsSession` 56

A.4 Module `emsSession` 58

A.5 Module `emsSessionFactory` 59

Bibliography 61

ITU-T Recommendation Q.816.2

CORBA-based TMN services: Extensions to support service-oriented interfaces

1 Scope

The TMN architecture defined in ITU-T Rec. M.3010 (2000) introduces concepts from distributed processing and includes the use of multiple management protocols. ITU-T Recs Q.816 and X.780 subsequently define within this architecture a framework for applying the common object request broker architecture (CORBA) as one of the TMN management protocols. In this approach, manageable network resources are modelled as software objects accessible using CORBA. Information models written in the CORBA interface definition language (IDL) describe the object interfaces. In the original CORBA TMN framework, each managed object class (MOC) is a CORBA IDL interface and so each manageable resource is an independent CORBA object identified and accessed by an interoperable object reference (IOR) that allows for location-transparency. CORBA-based TMN interfaces using the approach where each manageable resource is addressable with a unique IOR have become known as "fine-grained" interfaces. This approach flexibly allows each managed object to reside anywhere – but at the expense that managing systems have on hand an IOR for each managed object they wish to access.

CORBA-based TMN interfaces where not an IOR need be assigned to each manageable resource have become known as "coarse-grained" interfaces. A coarse-grained TMN interface exposes a coarser level of abstraction between a managing system and a managed system for accessing manageable resources but without loss of any detail. At a coarse-grained TMN interface a CORBA object (with an IOR), which is used to access manageable resources having no IOR and invoke operations on them, is referred to as a façade, while an object that is accessed through a façade is referred to as a lightweight object. ITU-T Recs Q.816.1 and X.780.1 extend the framework to support coarse-grained interfaces where one or more façades are defined for each MOC.

This Recommendation, along with ITU-T Rec. X.780.2, add specifications to the framework to enable it to support a service-oriented style of interaction between managing systems and managed systems in addition to the fine-grained and coarse-grained styles specified in the other framework documents. This style of interaction has certain benefits. For example, it can relieve a managing system from having to separately retrieve an identifier or a location information for each type or even each instance of manageable resource it wishes to access, and it can provide a more efficient and very flexible separation of behaviour and state of managed objects. By introducing the so-called "service-oriented façades" it also changes somewhat the way software may be structured on the managed systems, a flexibility which some managed system suppliers may prefer.

The service-oriented framework adopts a lightweight specific use of CORBA to maximize interoperability and interface performance. As a consequence, the use of the naming service can be minimalistic and the use of the ORB and all common object services is lightweight by nature.

A service-oriented architecture (SOA) is an architectural style that aims at maximizing service sharing, reuse and interoperability in distributed environments through loose coupling among interacting components that expose their behaviour through interfaces. In anticipation of the frequent discovery of new business opportunities or threats, an SOA aims at providing open and agile business solutions that can rapidly extend or change on demand, and so SOA-based solutions are composed of reusable services with published and standards-compliant interfaces. The service-oriented approach to CORBA interface design is intended to support efficient SOA interfaces and allow for growth and change as technologies and services are evolving.

The scope of this Recommendation is the same as the fine-grained and coarse-grained TMN CORBA frameworks. These frameworks and the service-oriented extensions cover all interfaces in the TMN where CORBA may be used. These interfaces are OS-OS interfaces according to ITU-T Rec. M.3010, where one OS takes a client/manager role (e.g., an NMS) and the other OS takes a server/agent role (e.g., an EMS). To be concrete, the service-oriented framework support services IDL of Annex A refers to an NML-EML interface but it can be applied to any managing system and managed system. It is expected, however, that not all capabilities and services defined here are required in all TMN interfaces. This implies that the framework can be used for interfaces between management systems at all levels of abstractions (inter- and intra-administration at various logical layers) as well as between management systems and network elements.

1.1 Purpose

The purpose of this Recommendation and the companion ITU-T Rec. X.780.2 is to extend the TMN CORBA framework to enable it to be used in a wider range of applications. The extensions enable a lightweight mode of interaction between the managing and managed systems which may be preferred in many situations. They also enable the endorsement of *de facto* CORBA services and information models usage which are recognized in the telecommunications industry. Thus, this Recommendation is intended for use by various groups specifying network management interfaces.

1.2 Application

ITU-T Rec. X.780.2 accompanies this Recommendation and extends the object modelling guidelines, the superclasses, and the standard sets of data types, exceptions, notifications and constants defined in ITU-T Recs X.780 and X.780.1. Collectively, ITU-T Rec. X.780.2 and this Recommendation define a *framework* for CORBA-based service-oriented TMN interfaces.

ITU-T Rec. M.3010 provides, in its Amendment 1 (2003), conformance definitions for TMN interfaces between physical blocks. When CORBA is used as the TMN management protocol, the conformance criteria refer to the CORBA framework which provides more than one paradigm choice for ORB and CORBA services usage (Q.816-series ITU-T Recommendations) and information modelling in IDL (X.780-series ITU-T Recommendations). Currently these choices are the fine-grained and coarse-grained approaches. According to ITU-T Rec. M.3010 an operations system (OS) interface may make a claim, by level, of *TMN interface information conformance* for each management capability that the interface supports. The supported management capability sets shall be specified by an information model document. Level A, Level B and Level C are defined and distinguished only by the source of the information models that are applied to specify the managed object classes the OS interface supports. The applicable information models need to be specified and well-documented in:

- ITU-T Recommendations, in case of Level A conformance;
- standards of other *de jure* or *de facto* standards bodies, in case of Level B conformance;
- a non-standard way, in case of Level C conformance.

In all cases, implementation conformance statements proformas following the X.781-series of ITU-T Recommendations, as appropriate, shall be provided.

Because ITU-T Recs X.780, X.780.1 and X.780.2 define slightly different approaches to modelling manageable resources on fine-grained, coarse-grained and service-oriented interfaces, interface model specifications will be slightly different for the fine-grained, coarse-grained and service-oriented framework paradigms. While information modelling according to ITU-T Recs X.780 and X.780.1 will always lead to Level A TMN interface information conformance, the lightweight information modelling according to ITU-T Rec. X.780.2 may lead to any conformance level.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[1] ITU-T Recommendation M.3010 (2000), *Principles for a telecommunications management network (including Amendment 1 (2003); Amendment 2 (2005)).*

[2] ITU-T Recommendations M.3050-series (2004), *Enhanced Telecom Operations Map (eTOM)*.

NOTE – This series of Recommendations has the following structure:

M.3050.0 – *eTOM – Introduction.*

M.3050.1 – *eTOM – The business process framework.*

M.3050.2 – *eTOM – Process decompositions and descriptions.*

M.3050.3 – *eTOM – Representative process flows.*

M.3050.4 – *eTOM – B2B integration: Using B2B inter-enterprise integration with the eTOM.*

M.3050/Suppl.1 – *eTOM – ITIL application note.*

M.3050/Suppl.2 – *eTOM – Public B2B Business Operations Map (BOM).*

M.3050/Suppl.3 – *eTOM to M.3400 mapping*.

[3] ITU-T Recommendation Q.816 (2001), *CORBA-based TMN services (including Corrigendum 1 (2001); Corrigendum 2 (2002); Amendment 1 (2001); Amendment 2 (2002)).*

[4] ITU-T Recommendation Q.816.1 (2001), *CORBA-based TMN services: Extensions to support coarse-grained interfaces*.

[5] ITU-T Recommendation Q.821.1 (2001), *CORBA-based TMN alarm surveillance service*.

[6] ITU-T Recommendation X.734 (1992) | ISO/IEC 10164-5:1993, *Information technology – Open Systems Interconnection – Systems management – Event report management function (including Corrigendum 1 (1994); Corrigendum 2 (1999); Amendment 1 (1995) and its Corrigendum 1 (1996))*.

[7] ITU-T Recommendation X.735 (1992) | ISO/IEC 10164-6:1993, *Information technology – Open Systems Interconnection – Systems management: Log control function (including Corrigendum 1 (2001); Amendment 1 (1995) and its Corrigendum 1 (1996))*.

[8] ITU-T Recommendation X.780 (2001), *TMN guidelines for defining CORBA managed objects (including Corrigendum 1 (2001); Corrigendum 2 (2002); Amendment 1 (2002)).*

[9] ITU-T Recommendation X.780.1 (2001), *TMN guidelines for defining coarse-grained CORBA managed object interfaces (including Corrigendum 1 (2002); Amendment 1 (2002))*.

[10] ITU-T Recommendation X.780.2 (2007), *TMN guidelines for defining service-oriented CORBA managed objects and façade objects*.

[11] ITU-T Recommendation X.920 (1997) | ISO/IEC 14750:1999, *Information technology – Open distributed processing – Interface definition language*.

[12] OMG Document formal/98-07-01, *The Common Object Request Broker: Architecture and Specification*, Revision 2.2.

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

[14] OMG Document formal/01-02-33, *The Common Object Request Broker: Architecture and Specification*, Revision 2.4.2.

[15] OMG Document formal/00-06-22, *Property Service Specification*, Version 1.0. See also *Chapter 13* of [34], July 1996.

[16] OMG Document formal/04-10-01, *Lightweight Services Specification*, Version 1.0.

[17] OMG Document formal/04-10-03, *Naming Service Specification*, Version 1.3. See also *Chapter 3* of [34], March 1995, and *Chapter 5* of [16], October 2004.

[18] OMG Document formal/04-10-02, *Event Service Specification*, Version 1.2. See also *Chapter 4* of [34], March 1995, and *Chapter 6* of [16], October 2004.

[19] OMG Document formal/04-10-13, *Notification Service Specification*, Version 1.1. See also OMG TC Document telecom/98-11-01, *Notification Service Joint Revised Submission*, November 1998.

[20] OMG Document formal/03-07-01, *Telecom Log Service Specification*, Version 1.1.2.

[21] OMG Document formal/02-05-15, *The Common Object Request Broker Architecture and Specification*, Revision 2.6.1.

[22] OMG Document formal/04-03-12, *Common Object Request Broker Architecture: Core Specification*, Revision 3.0.3. See also OMG TC Document ptc/02-01-14, *Draft CORBA Core 3.0 consisting of CORBA Core 2.6 + Core and Interop RTF 12/2000 Changes + Components FTF Changes*, January 2002.

[23] ETSI TS 132 150 V6.5.0 (2006), *Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); Telecommunication management; Integration Reference Point (IRP) Concept and definitions* (3GPP TS 32.150 version 6.5.0 Release 6).

[24] ETSI TS 132 303 V6.6.0 (2005), *Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); Telecommunication management; Notification Integration Reference Point (IRP): Common Object Request Broker Architecture (CORBA) Solution Set (SS)* (3GPP TS 32.303 version 6.6.0 Release 6).

[25] ETSI TS 132 333 V6.1.0 (2006), *Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS), Telecommunication management; Notification Log (NL) Integration Reference Point (IRP): Common Object Request Broker Architecture (CORBA) Solution Set (SS)* (3GPP TS 32.333 version 6.1.0 Release 6).

[26] ETSI TS 132 111-3 V6.7.0 (2007), *Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); Telecommunication management; Fault Management; Part 3: Alarm Integration Reference Point (IRP): Common Object Request Broker Architecture (CORBA) Solution Set (SS)* (3GPP TS 32.111-3 version 6.7.0 Release 6).

[27] TM Forum TMF814 Version 3.0, Multi-Technology Network Management (MTNM) NML-EML Interface: CORBA IDL Solution Set, *Supporting Document "Programmatic Versioning"*, file "versioning.pdf", March 2004.

[28] TM Forum TMF814 Version 3.0, Multi-Technology Network Management (MTNM) NML-EML Interface: CORBA IDL Solution Set, *Supporting Document "Guidelines for Using the OMG Notification and Telecom Log Service"*, file "OMGservicesUsage.pdf", March 2004.

[29] OASIS Document soa-rm (2006), *Reference Model for Service Oriented Architecture 1.0*.

3 Definitions

The following terms used in this Recommendation are defined in ITU-T Rec. M.3010:

– **interface**;

– **operations system (OS)**.

The following term used in this Recommendation is adapted from ITU-T Rec. X.700:

– **managed object (MO)**: The management view of a resource, as defined by ITU-T Rec. M.3050.1, that may be managed through the use of a management protocol (such as CORBA).

NOTE 1 – ITU-T Rec. X.700 restricts the semantics of "resource" to resources which "provide interconnection capabilities" and "allow communications to take place"; it explicitly excludes resources which "provide data storage or processing capabilities". ITU-T Rec. M.3050.1 draws resources from the eTOM's application, computing and network domains.

NOTE 2 – A managed object is characterized in terms of attributes it possesses, operations that may be performed upon it, notifications that it may issue, and its relationships with other managed objects.

The following terms used in this Recommendation are adapted from ITU-T Rec. X.701:

– **agent**: An OS which has taken an agent role.

– **agent role**: A role taken by an OS in which it is capable of performing management operations on managed objects and of emitting notifications on behalf of managed objects.

– **managed object class (MOC)**: A named set of managed objects sharing the same (possibly named) set of attributes (the MOC's state) and the same (possibly named) set of operations and notifications (the MOC's behaviour).

– **managed system**: A synonym for agent.

– **manager**: An OS which has taken a manager role.

NOTE 3 – Some products that implement the service-oriented framework use this term, or the term "object manager", as a synonym for service-oriented façade.

– **manager role**: A role taken by an OS in which it is capable of issuing management operations and of receiving notifications.

– **managing system**: A synonym for manager.

NOTE 4 – ITU-T Rec. X.701 defines the term "(management) interaction" and uses it to state more sophisticated definitions of agent and manager.

The following term used in this Recommendation is defined in ITU-T Rec. X.703:

– **notification**.

NOTE 5 – ITU-T Rec. X.703: "*Information technology – Open Distributed Management Architecture*" provides the ODMA definition, or ODP-RM definition, of notification. ITU-T Rec. X.703/Amd.1: "*Support using Common Object Request Broker Architecture (CORBA)*" relates this definition to the OMG event definition [18] (generic or typed event message), and hence to the OMG notification definition [19] (generic or typed or structured event message).

The following term used in this Recommendation is defined in ITU-T Rec. Q.816 and in [18]:

– **event channel**.

The following term used in this Recommendation is defined in [19]:

– **notification channel**.

The following terms used in this Recommendation are defined in ITU-T Rec. X.734:

– **event forwarding discriminator (EFD)**;

– **event report**.

The following terms used in this Recommendation are defined in ITU-T Rec. X.735 and in [20]:

– **log**;

– **log record**.

The following term used in this Recommendation is defined in [20]:

– **notification log**.

The following term used in this Recommendation is defined in ITU-T Rec. X.780.1:

– **façade**.

The following terms are defined in this Recommendation in harmony with [29]:

– **service broker (SB)**: An entity, also known as service registry, which allows SPs to register (and maintain) SDs and allows SCs to find SDs in a location-transparent way.

– **service consumer (SC)**: An entity which seeks to satisfy a particular need through the use of capabilities offered by means of a service.

– **service description (SD)**: The information needed in order to use, or consider using, a service. See section 3.3.1 of [29].

NOTE 6 – The most important part of a service description is the association of one or more service interfaces that enable access to the capabilities provided by the service.

NOTE 7 – When a service description includes constraints and policies, it is called a service contract.

– **service interface (SI)**: The means by which all or part of the underlying capabilities of a service are accessed. It is a means for interacting with a service (see section 3.3.1.4 of [29]).

NOTE 8 – A service interface delineates and exposes an external view of functionality of a service; it defines all or part of that service's action boundary.

– **service-oriented approach/architecture (SOA)**: An ICT/IT&T architecture of services, policies, best practices and frameworks in which components can be reused and repurposed rapidly in order to achieve shared and new functionality.

NOTE 9 – An SOA provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations. It is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It is an architectural style that aims at maximizing service sharing, reuse and interoperability in distributed environments through loose coupling among interacting components that expose their behaviour through interfaces. In anticipation of the frequent discovery of new business opportunities or threats, SOAs aim at providing open and agile business solutions that can rapidly be extended or changed on demand, and so SOA-based solutions are composed of reusable services with published and standards-compliant interfaces. SOA entities are accessible only through interfaces and connected by service descriptions. The SOA paradigm enables fast adaptation to changing business needs as well as cost reduction in the deployment of new services and maintenance of deployed services.

– **service-oriented façade (SO façade)**: A synonym for service-oriented façade interface or for service-oriented façade object, depending on the context.

– **service-oriented façade interface (SO façade interface)**: An object interface defined to provide access to a set of service-oriented managed objects, and optionally also to other objects.

– **service-oriented façade object (SO façade object)**: An object which exists in the managed system and provides, in a steward role, access to allocated service-oriented managed objects, and optionally also to other objects, of the managed system. It is an instance of an SO façade.

– **service-oriented managed object (SO MO)**: A managed object which is only accessible via an assigned service-oriented façade and provides behaviour (operations and notifications) to managing systems not directly but only through behaviour of its assigned façade.

NOTE 10 – A managed object represents a manageable resource in terms of state (attributes) and behaviour (operations and notifications). Functional components of the managed object such as state and behaviour can be distributed either in an implementation or in its interface definition or in both. A service-oriented managed object is a managed object whose state and behaviour are distributed in the interface definition; its state is left with the object while its behaviour is delegated to a steward object, namely its assigned façade. Whether this also implies a separation of the managed object implementation is not visible at the interface; it is an implementation-defined issue that is left to the managed system's discretion.

– **service provider (SP)**: An entity that offers the use of capabilities by means of a service.

– **service resource (SR)**: A physical or logical source or sink of data, or a state-aware either synchronous or asynchronous ICT/IT&T data/signal processing capability.

NOTE 11 – Service resources are used by services to deliver and maintain (persistent or temporary) data. When the data is specified by an information model, service resources can be used to construct service interfaces and processable service descriptions (i.e., metadata). Examples of service resources are network elements, network infrastructure items (e.g., connectivity or profile information), software/firmware, database systems, and COTS technology components.

– **SOA service**: A means by which the needs of a consumer are brought together with the capabilities of a provider, and which is realized as a set of behaviours that are accessible through, or are using, one or more service interfaces. Refer to section 3.1 of [29] for details.

NOTE 12 – An SOA incarnates any service as one or more entities that expose service interfaces. A service in a provider role constitutes an action boundary regarding access and control between one or more service resources and one or more consumers of the resources.

4 Abbreviations

This Recommendation uses the following abbreviations:

3GPP	3rd Generation Partnership Project
AMI	Asynchronous Messaging Invocation
ATM	Asynchronous Transfer Mode
BPON	Broadband Passive Optical Network
CBTS	CORBA-Based TMN Services (ITU-T Rec. Q.816)
CCBTS	Coarse-grained CORBA-Based TMN Services (ITU-T Rec. Q.816.1)
CCS	Concurrency Control Service
CM	Configuration Management
CMIP	Common Management Information Protocol
CORBA	Common ORB Architecture
COS	Common Object Service(s)
COTS	Commercial Off-The-Shelf technology
CSI	Common Secure Interoperability
DIOA	Distributed Interface-Oriented Architecture
DN	Distinguished Name

DSL	Digital Subscriber Line
EFD	Event Forwarding Discriminator
EML	Element Management Layer
EMS	Element Management System
eTOM	enhanced Telecom Operations Map
fd	filterable data
FIFO	First In, First Out
GDCCMO	Guidelines for Defining Coarse-grained CORBA Managed Object Interfaces (ITU-T Rec. X.780.1)
GDCMO	Guidelines for Defining CORBA Managed Objects (ITU-T Rec. X.780)
GDMO	Guidelines for the Definition of Managed Objects (ITU-T Rec. X.721/X.722)
GDSOCMO	Guidelines for Defining Service-Oriented CORBA Managed Objects (ITU-T Rec. X.780.2)
GIOP	General Inter-ORB Protocol
ICT	Information and Communications Technology
ID	Identifier
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IIOP/SSL	IIOP over SSL
IIOP/TLS	IIOP over TLS
IKE	Internet Key Exchange
IOP	Inter-ORB Protocol
IOR	Interoperable Object Reference
IPSec	IP Security
IRP	Integration Reference Point
IT	Information Technology
IT&T	Information Technology & Telecommunication(s)
LDAP	Lightweight Directory Access Protocol
MO	Managed Object
MOC	Managed Object Class
MOO	Multiple Object Operation
MTNM	Multi-Technology Network Management
NE	Network Element
NGOSS	New Generation Operations Systems and Software
NL	Notification Log
NML	Network Management Layer
NMS	Network Management System

NV pair	name/value pair with name being a string and with a "name=value" semantics
OAM&P	Operations, Administration, Maintenance & Provisioning
OASIS	Organization for the Advancement of Structured Information Standards
OCN	Object Creation Notification
ohf	optional header field
OMG	Object Management Group
OO	Object-Oriented
OOAD	Object-Oriented Analysis and Design
ORB	Object Request Broker
OS	Operations System
OTS	Object Transaction Service
OTS	OMG Transaction Service
PIM	Platform Independent Modelling
PKI	Public Key Infrastructure
PM	Performance Management
POA	Portable Object Adapter
PSM	Platform Specific Modelling
QoS	Quality of Service
RFP	Request For Proposal(s)
SB	Service Broker
SC	Service Consumer
SD	Service Description
SECIOP	Secure Inter-ORB Protocol
SECP	Security Protocol
SI	Service Interface
SO	Service-Oriented
SOA	Service-Oriented Approach/Architecture
SOCBTS	Service-Oriented CORBA-Based TMN Services (ITU-T Rec. Q.816.2)
SOF	Service-Oriented Façade
SP	Service Provider
SR	Service Resource
SS	Solution Set
SSL	Secure Sockets Layer
SSL/TLS	SSL or TLS
TC	Technical Committee
TCA	Threshold Crossing Alert
TCP	Transmission Control Protocol

TII	Time-Independent Invocation
TLS	Transport Layer Security
TMN	Telecommunications Management Network
TS	Technical Specification
UML	Unified Modelling Language
UMTS	Universal Mobile Telecommunications System
UTRAD	Unified TMN Requirements, Analysis and Design

5 Conventions

A few conventions are followed in this Recommendation to make the reader aware of the purpose of the text. While most of the Recommendation is normative, paragraphs succinctly stating mandatory requirements to be met by a management system (managing and/or managed) are preceded by a boldface "R" enclosed in parentheses, followed by a short name indicating the subject of the requirement, and a number. For example:

(R) EXAMPLE-1 An example mandatory requirement.

Requirements that may be optionally implemented by a management system are preceded by an "O" instead of an "R". For example:

(O) EXAMPLE-2 An example optional requirement.

The requirement statements are used to create compliance and conformance profiles.

Examples of CORBA IDL are included in this Recommendation, and normative IDL specifying the TMN-specific framework support services, and associated data types, is included in Annex A. The IDL is written in a 9-point courier typeface:

```
// Example IDL
   void getManager(
         in string managerName,
         out common::Common_I managerInterface)
        raises(globaldefs::ProcessingFailureException);
```

Annex A contains source code that implementers will want to extract and compile. It is normative and should be used by developers implementing systems that conform to this Recommendation. Refer to clause 5.2/Q.816.1 for recommendations with regard to cutting and pasting the IDL of this Recommendation into plain text files that may then be compiled. For example, the entire IDL file "itut_q816_2.idl" can be generated by cutting Annex A from the Microsoft® Word® version of this Recommendation and saving as "Text Only with Line Breaks". This IDL file contains some ordinary comments and a large number of formatted comments to be parsed by compilers used to convert IDL to HTML for easier reading. A formatted comment begins with /** and ends with */ and is associated with the next IDL construct. HTML formatting tags are allowed with these comments as are certain keywords (preceded by a '@' symbol) that are converted by the IDL-to-HTML compilers into additional formatting. Users of the IDL may want to generate HTML as the resulting HTML files have links that make for quick navigation through the files.

6 Service-oriented interface design considerations

This clause identifies several design considerations that should be addressed by the framework as support for lightweight use of CORBA through service-oriented interfaces is added.

CORBA realizes a distributed interface-oriented architecture (DIOA) where location-transparent objects expose one or more interfaces for access to their encapsulated state and behaviour. The concepts of coarse-grained object modelling and service-oriented architecture (SOA) resolve issues

with object-oriented analysis and design (OOAD) best practices. Classical OOAD focuses on the class level, i.e., encapsulates behaviour and related data in the same object. This has become known as "fine-grained" approach. A general "coarse-grained" approach uses the *façade* design pattern[1] to separate state and behaviour (and share the state) of some classes, i.e., the behaviour of some (shared) objects is no longer controlled by themselves but by façade objects. In a coarse-grained object-oriented approach, data is still encapsulated but data access and control are organized at a separate upper level which results in coarser "grains" being accessed and controlled.

ITU-T Recs Q.816.1 and X.780.1 use a specific form of the façade design pattern (e.g., an X.780.1 façade can only provide access to a single MOC) and other design considerations to define support for coarse-grained interfaces. This Recommendation, along with ITU-T Rec. X.780.2, define a lightweight generic use of the façade design pattern which, together with other lightweight design considerations, generalize the coarse-grained approach (and therefore indirectly also the fine-grained approach). This novel approach to interface design is called "service-oriented" since it paves the way for introducing SOA principles to the TMN interface specification methodology. CORBA-based service orientation requires the flexibility of application-specific access granularity where well-defined sets of TMN entity types are accessed through assigned façades with IORs (see clause 4.2.1/X.780). Refer to clause 7.3 below and clause 12/X.780.2 for a thorough description of the relationships between the three approaches. Additional information is available from [53].

The design considerations related to CORBA services concern the lightweight and service-oriented use of the ORB, the use of session objects to control the communication between managing and managed systems, and the lightweight use of naming, notification and telecom log services.

6.1 Flexible use of façade design pattern

The coarse-grained framework introduces façades but also requires certain coupling conditions between a façade and the managed objects that are accessible through the façade. For example, a managed system shall provide at least one façade interface for each class of managed objects that may be instantiated on it, even if these objects also support direct CORBA interfaces, and may provide multiple façade interfaces for a given class of managed objects. These requirements effectively reduce the coarse-grained approach to a sort of class-grained approach. Refer to Appendix I/Q.816 for the concepts of class-grained approach and grain-neutral approach.

A coarse-grained interface is created by first defining a fine-grained interface according to the guidelines defined in ITU-T Rec. X.780 which cover both creating GDMO-like IDL interfaces from scratch as well as translating a GDMO interface to IDL. Once a fine-grained interface is defined, façade interfaces for each of the managed object interfaces are developed according to rules defined in ITU-T Rec. X.780.1. The fine-grained managed object interface specifications shall be retained. All of the other constructs defined for the fine-grained interface, including data types, value types, exceptions, notifications, and factories, are reused without modification on the coarse-grained interface. As a major example, ITU-T Rec. M.3120 applies the X.780 and X.780.1 guidelines to ITU-T Recs M.3100 and G.855.1 and translates manually all of their GDMO interfaces to first fine-grained and then coarse-grained CORBA IDL interfaces.

A service-oriented interface *can* depend on predefined coarse-grained or fine-grained interfaces but usually does not. The service-oriented approach requires the introduction of *service-oriented façades*, which are CORBA objects, and *managed objects*, which are accessed and controlled

---

[1] The façade design pattern [52] is a structural pattern with the intent to provide a unified interface to a set of interfaces in a subsystem. A façade defines a higher-level interface that makes the subsystem easier to use. A façade promotes loose coupling between the subsystem and its clients. Loose coupling makes it possible to vary the components and interfaces of the subsystem without affecting its clients.

(preferably exclusively) through service-oriented façades[2]. Both types of objects are instantiated at the managed system. Only loose coupling between managed objects and service-oriented façades is required from the outset (see clause 7.2.1.1) leaving detailed relationships to the discretion of the managed system, which may also support dynamic binding of managing systems to one or more service-oriented façades (i.e., systems interactions are established and broken dynamically at run time and not yet defined statically at design time).

While transition from a fine-grained model to a coarse-grained model is defined *bottom-up* through deterministic rules for coarsening of a fine-grained interface, transition from a service-oriented model to a coarse-grained or fine-grained model would be defined (if required someday) *top-down* through rules for stepwise refining of a service-oriented interface according to ITU-T Recs X.780.1 and X.780. However, the service-oriented approach mainly promotes coexistence of fine-grained, coarse-grained and (information model-dependent) service-oriented interfaces, and initially does not aim at transition of one TMN interface kind to another.

6.2 Lightweight use of ORB

The OMG CORBA specifications 2.2 [12], 2.3 [13] and 2.4 [14] consist of several parts for several aspects of CORBA that shall or should be offered by compliant ORB vendors. See also ITU-T Rec. X.920. Most important for use by the TMN frameworks are the following *two aspects of CORBA*:

- **IDL modelling repertoire of ORB products**: these are mainly the chapters 3 "OMG IDL Syntax and Semantics" and 5 "Value Type Semantics" of the CORBA specifications;
- **built-in basic CORBA system services of ORB products**: these are mainly the chapters 4 "ORB Interface" and 11 "The Portable Object Adapter" of the CORBA specifications.

The lightweight use of the first aspect, *the IDL repertoire and modelling*, including IDL style considerations (in compliance with [51] and similar to Annex D of [23]), is specified in the companion ITU-T Rec. X.780.2 whilst the lightweight use of the second aspect, *the basic CORBA system services*, is specified in this Recommendation.

Revisions 2.2 and 2.3 of CORBA introduced the *portable object adapter (POA)* and *value types*, respectively. Both capabilities are required by the fine-grained and coarse-grained frameworks. These requirements are implied in the explicit requirement that the supported version of CORBA shall be 2.3.1 or later. The fine-grained and coarse-grained frameworks do specify neither separate POA features nor individual value type features required for ORB compliance.

CORBA Revisioºn 2.4 introduced among other things *Minimum CORBA* and *CORBA Messaging*, in particular asynchronous messaging invocation (AMI) including time-independent invocation (TII). Minimum CORBA aims for ORB size reduction and overall performance improvements. It omits the features from the `CORBA` and `PortableServer` modules of the CORBA specification that support the run-time aspects of CORBA (e.g., dynamic interfaces at client side or server side, interceptors, dynamic mode of POA operation) and indicates conformant ways for vendor-specific optimizations. CORBA messaging standardizes and quality assures asynchronous requests, which previously could be issued only with the "ORB best effort" semantics of one-way operations using

---

[2] Some products that implement the service-oriented framework use the term "object manager" or just "manager", as a synonym for service-oriented façade. Another synonym sometimes used is "managing object" in contrast with managed object. Evolving managed objects to "service-oriented managed objects" means separating their state and behaviour, and outsourcing the behaviour to assigned "managing objects" that take a steward role regarding the operations and notifications of their allocated managed objects. This fundamental SOA principle can also be described as separating the *definition and storage of data* from the *delivery of data*. Since the terms "managing object" and "manager" may give rise to misunderstandings with regard to the managing role of systems and subsystems, this Recommendation mainly uses the term "service-oriented façade" but may sometimes use the synonyms for explanation purposes.

application-specific callback objects. It also provides support for "persistent" requests. It is based on an IDL-to-"implied-IDL" mapping that is carried out by messaging-enabled IDL code generators when generating language-specific client stubs. Every (synchronous) IDL operation is copied to an asynchronous operation whose signature is changed by adding either a callback interface (reply handler) as input parameter or a (queriable) poller value type (reply wrapper) as return value.

6.2.1 Use of dynamic and asynchronous capabilities

This clause discusses *the basic CORBA system services aspect* of CORBA with regard to the service-oriented framework.

Though being CORBA-specific, the service-oriented framework supports OMG's platform independent modelling (PIM) in UML and aims at the introduction of SOA principles to the TMN interface specification methodology UTRAD (see M.3020-series ITU-T Recommendations). SOA is interface-oriented and defines two views on components: the inward view of a service consumer, who requires and consumes one or more interfaces, and the outward view of a service provider, who implements and provides interfaces. A core SOA principle is a loose coupling of components that interact according to the publish-find-negotiate-bind-execute paradigm, which implies the possibility of dynamic binding, unbinding and rebinding between a required interface and a located provided interface. Refer to clause 7.2.1.2 for a summary of the basic SOA artefacts.

As a consequence, the dynamic and asynchronous aspects of CORBA are very important for the specification of service-oriented interfaces even though some of them are a bit sophisticated. But another SOA principle is ease of component development and deployment which is enabled by lightweight features. This framework therefore adopts a stepwise approach to run-time and asynchronous CORBA features, which extends the fine-grained and coarse-grained interface design approaches to ORB usage. The following steps are defined:

- mandatory use of the POA as defined by Minimum CORBA (Note 1);
- optional use of dynamic mode of POA operation;
- optional use of other dynamic aspects of CORBA omitted by Minimum CORBA;
- use of one-way operations with application-specific callback objects only in exceptional and provisional cases and then only in conjunction with a synchronization policy as specified by CORBA Messaging; as a rule TMN IDL interfaces are defined to be synchronous;
- optional use of transient asynchronous requests as defined by the AMI specification and further detailed in clause 6.4/Q.816 (Note 2);
- optional use of persistent asynchronous requests as defined by the TII specification.

The service-oriented framework therefore concurs with the fine-grained and coarse-grained frameworks regarding the use of POA and AMI except that it allows quality assured one-way operations in exceptional and provisional cases. It adds a prioritization of optional capabilities whose details and impacts on CORBA services usage (e.g., transactional asynchronous requests) are for further study. As a consequence, version 2.2 or later of CORBA shall be supported. From an IDL modelling point of view, CORBA 2.3 is only needed when value types shall be used (see clause 6.2.2/X.780.2). From a run-time performance point of view CORBA 2.3 may be beneficial for the processing of unbounded sequences, which are heavily used in COS and telecom IDL specifications (and even with complex members such as nested strings), since ORB products that implement CORBA 2.3 are said to treat all kinds of variable-length structures more efficiently. CORBA 2.3 also offers improved type code techniques for introspection and manipulation of the IDL type `any`. Furthermore ORB interoperability strongly depends on the ORB version (see clause 8.6).

NOTE 1 – Support of basic POA capabilities and exclusion of preceding object adapters ("BOA begone") is important to all frameworks because it enables implementations based on a framework to scale up to millions upon millions of instantiated objects, a magnitude required for telecommunications network management applications, and ensures portability between ORB products. The object adapter is the built-in CORBA mechanism that connects a request with the right code to service that request. A programming language entity such as a C++ object that implements requests on one or more CORBA objects, i.e., operations of one or more CORBA interfaces, is called a servant. The POA achieves maximum portability between ORBs, minimizes for a request the implementation code route covered to connect to the incarnating servant and process the request, and allows to persistently store object states between operation invocations. Using the POA is a *conditio sine qua non* when implementing fine-grained interfaces. For coarse-grained and service-oriented interfaces the POA provides, besides ORB portability, also considerable performance advantages when few servants are called with exceedingly high frequency. These considerations do show that the object adapter will no longer be a potential bottleneck for scaleability and overall performance but they need to be supplemented by considerations for the lightweight use of the Naming Service (see clause 6.4).

NOTE 2 – Asynchronous requests are a client side (i.e., managing system) programming language mapping issue. The server side is not affected by invocation policies of clients but sticks to always expecting to be invoked synchronously. To allow nevertheless synchronous clients to make non-blocking requests on a CORBA object, which may be required to avoid poor performance due to network latency, the service-oriented framework recommends that CORBA servers (i.e., managed systems) implement multithreading (e.g., by using a thread pool policy with appropriate locking granularity). The fine-grained and coarse-grained frameworks require the client side to implement multi-threading in case the AMI is not used. Within the service-oriented framework, multi-threaded managing systems are optional.

6.2.2 IDL repertoire and ways of MOC specification

This clause addresses *the IDL repertoire and modelling aspect* of CORBA with regard to the service-oriented framework.

The CORBA IDL-related constituents of the service-oriented framework are described in the companion ITU-T Rec. X.780.2, including the minimum IDL repertoire that should be used, based on the service-oriented object model IDL and the service-oriented CORBA modelling guidelines. A particularly important part is the modelling of managed objects and service-oriented façades, which inherit from a set of common attributes called *Common_T* (which is a virtual struct) respectively from a root service-oriented façade interface called *Common_I* (which is virtual). The *Common_I* façade interface specifies among other things the setter/mutator functions for the settable common attributes. All three frameworks group the attributes of managed object types into separate entities, the managed object classes (MOCs), which can be interfaces or be instantiated (by the managed system or the managing system or both, as specified) to non-CORBA managed objects that are identifiable by distinguished names. The fine-grained and coarse-grained frameworks require value types as the grouping mechanism, while the service-oriented framework allows for more flexibility in TMN interface design and defines *four alternative ways of MOC specification*:

- an MOC may be an `interface` that has the attributes as CORBA attributes or explicitly defined accessor (and mutator) functions;
- an MOC may be a `struct` that has the attributes as record members;
- an MOC may be a `valuetype` that has the attributes as public state members;
- an MOC may be a CORBA sequence of name/value pairs with `string` names and `any` values, i.e., an MOC may be of type `sequence<struct {string name; any value;};>`, that has the attributes as name/value pairs with a "name=value" semantics.

Refer to ITU-T Rec. X.780.2 for details and recommendations for the use of the respective way of MOC specification. As a rule, OMG specifications define properties of entities as name/value pairs with `string` names and `any` values [12], [13], [14], [15] and [19] but also introduce such name/value pairs with `string` values [17]. To support and enable extensibility, the service-oriented framework makes vigorous use of free format name/value pairs where the name and the value are both a simple `string`. Companion ITU-T Rec. X.780.2 also specifies rules and guidelines for *the*

lightweight definition of operations, exceptions and notifications. Since the three TMN CORBA frameworks are meant to be used for the management of telecommunications networks, ITU-T Rec. X.780.2 moreover specifies rules for the IDL modelling of multiple telecom transmission technologies according to the principles of ITU-T Recs G.805 and G.809. This is *the multi-technology part* of the service-oriented framework, which can also be used together with the coarse-grained and fine-grained frameworks.

6.3 Use of session service by managed and managing systems

The session service provides capabilities to manage a client/server connection between a managing system and a managed system, which is called a session. It enables either a client or a server to detect the loss of communication with the associated party. For a single session between a managing system and a managed system, there are two session objects. One is maintained on the managing system (client); the other one is maintained on the managed system (server).

A session object has a read-only attribute that contains a reference to the session object on the other side (managing/managed system) to which the object is associated. This attribute allows for callback invocation of the associated session object and to check in case of communication failure if the association is still valid. Loss of communication may be detected through a `ping` operation, and a one-way `endSession` operation allows for a controlled disconnect between parties that releases all resources allocated for this session object and destroys the object.

A client session provides operations for callback by the server in case of notification losses and termination of a loss period. A server session provides operations to query the names of the supported managing objects (manager interfaces), to retrieve for each supported manager its object reference to gain access to it without using the naming service, and to retrieve the object reference of the unique event channel to be used by the managing system in this session.

To enable the session service, the managed system (server) registers one or more instances of the EMS session factory interface with the naming service, which then constitutes the unique entry point(s) to the managed system. The managing system instantiates a client session object and calls the `getEmsSession` operation of an EMS session factory, together with authentication data, to request instantiation and association of a server session object and return of its object reference. Before establishing a session with the managed system, the managing system may use the version check facility of the EMS session factory to avoid connection with an unpopular IDL version.

6.4 Lightweight use of naming service

A CORBA *name component* is a pair of strings that is called an id/kind pair since it is of type `struct NameComponent {string id; string kind;};`. A CORBA *name* is a list of name components, that is a CORBA name is of type `sequence<NameComponent>`. An OMG naming service [17] maps CORBA names, or name components, to IORs. A name-to-IOR, or name component-to-IOR, association is briefly called a (name, or name component,) *binding*. A *naming context*, or just a *context* for short, is a CORBA system object, namely an instance of the `CosNaming::NamingContext` interface, which stores bindings by implementing a table that maps unique CORBA names, or name components, to IORs. Different names, or no name, can be bound to any individual IOR in the same or different naming contexts. A binding can refer to an application object or another naming context of the same or a federated naming service.

The bindings of a naming service are usually depicted as a CORBA *naming graph* that shows context IORs with hollow nodes, application IORs as solid/filled nodes, and arrows between nodes representing the bindings. Each binding arrow is labelled with the binding's name part and points to its IOR part, and so collectively the arrows originating from an individual context node represent its binding table. In a given context, any CORBA name is represented by a sequence of nodes that originates from this context and traverses the naming graph until a leaf is reached. In particular, each node of a naming graph represents a CORBA name component. The ORB can be configured

to initialize with whatever naming service instance one wants, and provides an operation to obtain, at run time during the bootstrap phase of the managing or managed system, the IOR of the configured *Initial Naming Context* of this local naming service, which is often orphaned. An *Initial Naming Context* is not a root context, i.e., CORBA names are only relative distinguished names, except when only a single naming service or a hierarchical federation structure is used in the environment of the considered managing and managed systems, but a fully connected federation structure guarantees the same name regardless of which *initial naming context* is used.

A context provides operations to construct and modify the naming graph segment originating from it: binding and rebinding to objects, binding and rebinding to contexts, unbinding from objects, creating a new unbound context, creating a new bound context, and destroying a context. The incomplete usage of these operations can result in orphaned contexts (being unreachable from any federated naming service) and dangling bindings (having invalid IORs). A context also provides the `resolve` operation to retrieve the IOR bound to a CORBA name in the given context, and the `list` operation to iterate through the (potentially extraordinarily large number of) bindings of the given context by using a binding iterator (see clause 9.4/X.780.2).

The OMG has defined a *Lightweight Naming Service* specification (see chapter 5 of [16], chapter 3 of [17]), which is a compatible subset of the fully-fledged "heavyweight" service and intended to make the naming service suitable for use in resource-constrained environments. The definition and most design considerations are also suitable for supporting the design goals of service-oriented environments. Whilst the heavyweight service should be seen as an extension of the lightweight service, for backward compatibility reasons the lightweight service is formally defined by disabling certain operations and entire interfaces of the fully-featured service (through preprocessor directives, or additional exceptions, or just documentation). The following capabilities of a context are disabled: binding and rebinding to contexts, creating a new unbound context, listing a context (i.e., all bindings of the given context). A *lightweight naming service* also disables the interfaces `BindingIterator` (used by the `list` operation) and `NamingContextExt : NamingContext` (used for converting between CORBA names, stringified names and URL schemes), and it does in fact hide the bindings of all of its naming contexts by disabling the pertinent type definitions.

The disablement of naming service capabilities as defined by the *lightweight naming service* specification has the following consequences:

- a lightweight CORBA naming graph is a tree, a CORBA *naming tree*, since names cannot be bound to contexts but only new bound contexts can be created;
- the current bindings cannot be retrieved, even in case of a small naming tree where the use of a binding iterator would not be required.

The first consequence is appreciated for use by the service-oriented framework but would be too restrictive for usage in a federation environment where initial naming contexts need to be bound to contexts of other federated naming services. The second consequence is generally considered too lightweight, since it requires auxiliary means to keep track of the construction of the naming tree or too rigid naming tree specifications. Instead the implementation of a `list` operation without use of a binding iterator is recommended at least. Also the stringification and destringification of CORBA names may be needed (see clause 10.3/X.780.2). Refer to clause 8.2 for the detailed specification of the *service-oriented naming service* and to clause 6.4.3 for examples.

6.4.1 Minimalistic use of naming service

When the managed system (server) implements the session service, it registers only EMS session factory interface instances with the naming service. Access to all other CORBA objects, especially service-oriented façades, is provided through objects of the session service without using the naming service. This lightweight paradigm is called "minimalistic use of naming service".

6.4.2 Naming of managed objects and service-oriented façade objects

To achieve the objective of lightweight and SOA-styled use of CORBA through service-oriented interfaces, the resources in a managed system that need to be managed are defined as service-oriented managed objects (also known as second-level or lightweight objects). These managed objects are accessed and controlled through service-oriented façade objects (i.e., CORBA objects). CORBA objects possess an IOR. The managed objects (usually) are not CORBA objects and do not possess an IOR and thus relieve the management systems of the burden of storing and maintaining huge numbers of IORs for managed objects. Service-oriented managed objects are identified by names. Refer to the companion ITU-T Rec. X.780.2 for details. The CORBA naming of façade objects and the relationship between managed object names and façade object names depends on whether the managed system implements the session service, which allows for a minimalistic use of the Naming Service. Refer to clauses 6.4.3 and 8.2.1 for details.

6.4.3 Lightweight CORBA naming trees

The service-oriented framework recommends the CORBA naming graph to be a *lightweight tree* by restricting the possible kinds of its nodes (i.e., the admissible values of the `kind` attribute) and their ordering, and adding simultaneously multi-vendor capability for identifying in a unified way the vendors of the registered managed systems that implement the service-oriented façades. The following *kinds of CORBA name components* are defined by the service-oriented framework (see clause 10.5.4/X.780.2 for further details):

- `"Class"`: denotes an OS-OS interface class, which is specified in CORBA IDL;
- `"Vendor"`: denotes an OS vendor;
- `"EmsInstance"`: denotes a server OS (e.g., an EMS);
- `"Version"`: denotes an IDL version of the parent OS-OS interface class;
- `"EmsSessionFactory_I"`: denotes a server session factory (see 9.1.2.5);
- `"Common_I"`: denotes a generic service-oriented façade (see clause 9.3/X.780.2);
- `<SO façade interface name>`: denotes a specific service-oriented façade;
- `<EventChannel>`: denotes an OMG event channel (e.g., a notification channel, a NotifyLog).

An example of a service-oriented CORBA naming tree is depicted in Figure 1 below.

The figure uses `<EMSvalue>` to indicate the name value of an EMS name and `<CompanyName>` to indicate a vendor name both according to the conventions of clause 10.3/X.780.2. It shows four OSes of the same vendor, which all implement the same IDL version. The OS-OS interface class `"TMF_MTNM"` specifies the TM Forum MTNM NML-EML interface TMF814 (see [27] and [28]). The left-hand OS does not implement the session service but the three shown service-oriented façades of MTNM. The third and fourth OSes are bound to the second OS indicating that they are subordinate to it (e.g., the second OS could be a domain management system that federates two element management systems). Note that the figure can easily be extended to multiple vendors.

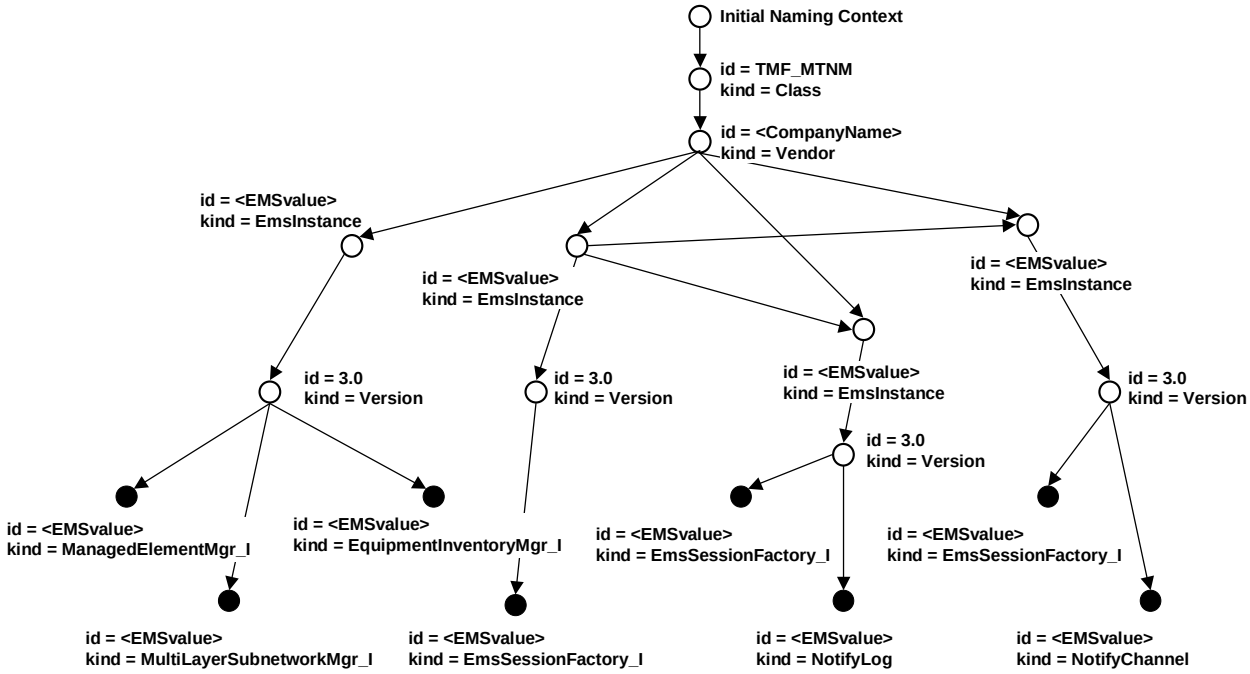

Figure 1 – Example of service-oriented CORBA naming tree

Refer to clause 8.2 below and clause 10.5.4/X.780.2 for further details. Figure 5/X.780.2 specifies in detail the above-listed kinds of lightweight CORBA name components together with their `id` values. Table 4/X.780.2 specifies the hierarchical order of the lightweight CORBA naming tree.

6.4.4 Storage of names and object containment relationships

In the coarse-grained framework, the façade interfaces are bound to names in an OMG naming service, much the same way the support service interfaces are. On interfaces that use façades, however, the managed objects' names are not required to be bound to IORs in the naming service. Instead, a new service for façade-accessible managed objects is introduced, the containment service, as a common place to store managed object names and containment relationship information. Clause 9.6.1/Q.816.1 provides a certain rationale for the containment service, which explicitly excludes the alternative to store managed object names in the façade objects.

The service-oriented framework stores as few façade IORs in a naming service as reasonably possible and allows for the façade objects as alternative storage locations of managed object names and containment relationships. This option is not only a more lightweight means of managed object name administration but also eliminates the threat for performance issues with the containment service, which basically is a more complicated Naming Service, that are well-known with regard to the naming service in the context of fine-grained interfaces. It also provides an element of federation (instead of centralization) since the storage is distributed across multiple façades.

6.5 Lightweight use of notification service

The OMG event service [18] allows for asynchronous exchange of event messages between clients by introducing *Event Channels* and *Typed Event Channels* that broker events, *Event suppliers* that provide events, and *Event consumers* that process events. It distinguishes *Generic Events* and *Typed Events*. It specifies the *push model*, where suppliers communicate event data by invoking the `push` operation on the `PushConsumer` interface, and the *pull model*, where consumers request event data by invoking the `pull` or `try_pull` operation on the `PullSupplier` interface. Both models realize a

state synchronization mechanism between the consumer (e.g., a CORBA object of a managing system) and the supplier (e.g., a CORBA object of a managed system) according to the publish-subscribe paradigm (also known as observer design pattern, see clause 9.1.1).

The push model could be used without an event channel, if the supplier-side system provides an interface with an `attach` operation that can be called by the consumer-side system with the (stringified) IOR of a `PushConsumer` instance. The pull model could be used without an event channel, if the supplier-side system provides an interface with an `attach` operation that returns the (stringified) IOR of a `PullSupplier` instance and can be called by the consumer-side system with the (stringified) IOR of an interface that provides a parameterless `update` operation. This usage, where consumers and suppliers are callback objects (see also clause 9.1), is not specified for the event service, however. Instead event channels and associated Admin(istrative) and Proxy objects are used to manage connections between suppliers and consumers and to deliver events. Event channels separate concerns by decoupling the communications between suppliers and consumers; they avoid tight couplings of callback objects that do not scale well as their number increases.

The OMG has also defined a *Lightweight Event Service* specification (see chapter 6 of [16], chapter 3 of [18]), which is a compatible subset of the fully-fledged "heavyweight" service and intended to make the event service suitable for use in resource-constrained environments. It is formally defined by disabling both the pull model and typed events. These lightweight event service design considerations are also suitable for supporting the design goals of service-oriented environments, when seen in conjunction with the lightweight capabilities of the notification service.

The OMG notification service [19] enhances the simplistic Event Service by *Structured Events* and *Event Batches* (i.e., sequences of Structured Events), *Notification channels*, a *Notification channel factory*, Notification Service style *Proxy* objects and *Admin* (≅ Proxy group) objects, message *QoS properties configuration* capabilities (at channel, Admin, Proxy and event levels), channel *Admin properties configuration* capabilities, comprehensive *Event Filtering* capabilities (at Admin and Proxy levels, based on event content or properties settings), and reliable event delivery with *Event Acknowledgement* based on *Sequence numbers*. Notification channels support well-defined translations between the message formats Any (i.e., Generic), Typed, and Structured. In case of "Any → Typed" and "Structured → Typed", a specific format is expected for the input message that is also used for the output message in case of "Typed → Any" and "Typed → Structured".

The structured event header includes an *Event type* attribute with syntax `CosNotification::_EventType` consisting of a *Domain name* for the vertical industry domain the event supplier belongs to, or the OS-OS interface class that defines the event in CORBA IDL, and a *Type name* that shall be unique within a given domain (e.g., an OS-OS interface class such as TMF MTNM or a 3GPP IRP). The notification service provides the optional capability to *share subscriptions and offers* between channels and clients (i.e., suppliers and consumers), i.e., to convey end-to-end the knowledge of which event types are required from suppliers and which might be produced by them. This event discovery and coordination capability can be a great efficiency booster, in particular when used together with an *Event Type Repository* as specified by the Notification Service.

The push and pull models of the Notification Service could be used without channels by enabling consumers and suppliers to be tightly coupled callback objects. Consumers and suppliers would exchange IORs of `PushConsumer` and `PushSupplier`, respectively `PullConsumer` and `PullSupplier`, interface instances (as defined by the `CosNotifyComm` module). But this framework recommends the use of notification channels and Proxy objects to enable loose coupling of consumers and suppliers thereby avoiding the disadvantages of the callback paradigm.

The fine-grained and coarse-grained frameworks require use of structured or typed events with typed events being the preferred choice (and even declared an intended direction). Event batches are optional and generic events (i.e., Anys) are not allowed. The use of typed events has the advantage

of avoiding the general use of the IDL type `any`, which relies on type code techniques and detailed documentation, but requires the definition and use of an interface that provides the events as operations (and further provisions for the pull model). Therefore ITU-T Rec. X.780 defines the interface `Notifications` whose operations represent typed events and whose operations parameter names and values represent the contents of the filterable body of structured events.

The fine-grained and coarse-grained frameworks also support the 3GPP Notification IRP [24]. This specification uses event batches, an event batch with only one element being equivalent to a structured event, and defines the interface `NotificationIRP` to be used collectively with notification service interfaces and with, or without, notification channels. It requires support of the push model and offers support of the pull model as an option. It points out that use of notification channels and related Proxy and Admin objects could allow a malicious IRPManager (i.e., managing system) to compromise the integrity of the IRPAgent (i.e., managed system), if no appropriate authentication and authorization mechanisms are in place (see clause 7.2.4).

The service-oriented framework requires use of the push model with structured events. Typed events and event batches are optional and generic events (i.e., Anys) are not allowed. The framework therefore complies with OMG's *Lightweight Event Service* specification (replacing generic events by structured events though). It recommends the efficient support of on-demand pulling of events, and alarms in particular, through IDL operations without using the Notification Service. It supports the options of the fine-grained and coarse-grained frameworks, including the 3GPP Notification IRP, but enhances them with a more lightweight alternative.

Instead of an IDL interface that would define (typed) events as operations, ITU-T Rec. X.780.2 defines the CORBA module `notifications` whose data types are used to specify the event types and the content of most items of the filterable body portion of structured events (see clause 8.7/X.780.2). As a consequence, the service-oriented framework is able to fully support the guidelines for using the OMG Notification Service defined by TM Forum's MTNM specification [28].

The IDL definition of events as operations is a valuable option to fix all details of event definitions in a compilable way though this objective could easily be achieved by using structs or value types. But these approaches are not well extensible. So the actual use of typed events has extensibility, and hence backward compatibility, issues and is far less lightweight and flexible than the use of structured events, which were introduced just because of issues with typed events and in order to support highly optimized event filtering. Using filter objects with typed events is less efficient since upon receipt of a typed event a notification channel will disassemble the event into a sequence of name/value pairs, where each name and value identifies an input parameter value of the operation that was invoked to transmit the event to the channel. This name/value pair sequence is then used to evaluate the constraints of all filter objects associated with relevant Proxy and Admin objects.

The service-oriented framework also recommends a unified and lightweight use of the event filtering and QoS and Admin properties configuration capabilities of the notification service, which complies with the fine-grained and coarse-grained frameworks and the 3GPP and MTNM specifications. Refer to clause 8.3 below and clause 8.7/X.780.2 for further details.

6.6 Lightweight use of telecom log service

The companion ITU-T Recs X.735 and X.734 define the related *Log control function* and *Event report management function*, which are based on managed object classes specified in ITU-T Rec. X.721 (top; log; discriminator, event forwarding discriminator (EFD); log record, event log record). Figure 2 below, which is taken from these Recommendations, shows the impact of notifications emitted by managed objects (due to MO-specific events) to log processing and event processing. Event reporting is the ability to specify conditions to be satisfied by a potential event report emitted by a particular managed object in order to be sent to specified destinations. Logging is the ability to preserve information about events that may have occurred or operations that may

have been performed by or on managed objects or management support objects, and to determine which potential log records or received event reports are to be logged. While the notification service realizes X.734 event reporting, the telecom log service realizes X.735 logging.

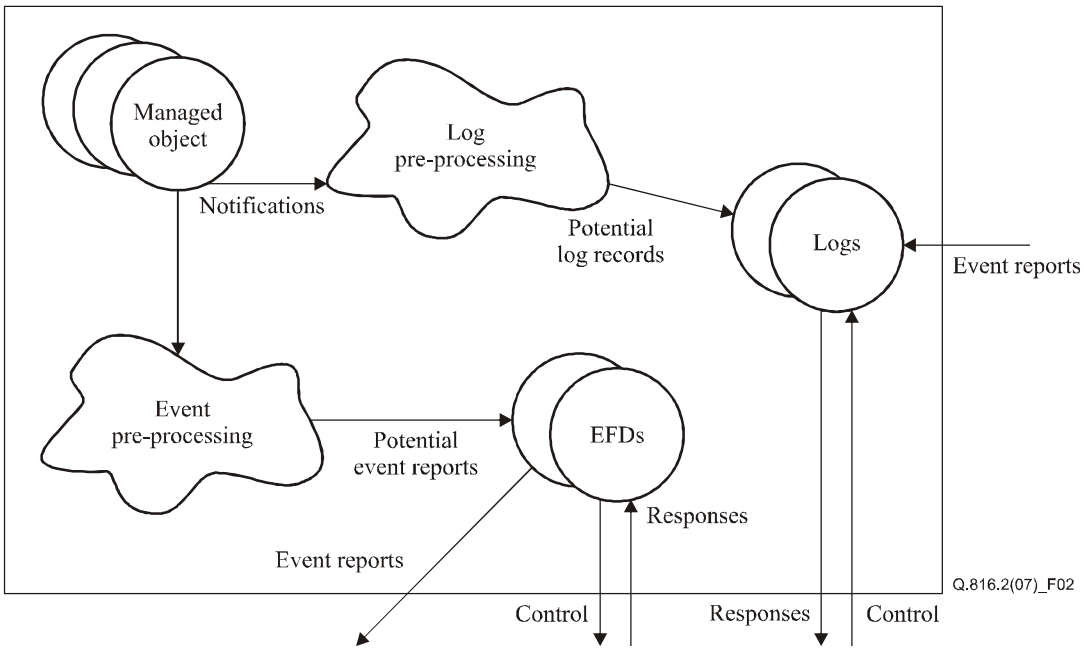

Figure 2 – Relationship between event report and log management models

During a pre-processing phase, discriminator input objects (potential log records or potential event reports) are generated and then tested according to the log's or EFD's discriminator construct attribute, which encapsulates filtering constraints. The log control or event report management function creates and deletes logs and log records or discriminators and retrieves attributes of log records or discriminators according to ITU-T Rec. X.730, and the log control function reports log alarms according to ITU-T Rec. X.733. Both management functions notify state changes according to ITU-T Rec. X.731, and attribute value changes as well as object creations and deletions according to ITU-T Rec. X.730. Further details about these management functions are specified in ITU-T Recs X.735 and X.734.

The OMG telecom log service [20] defines interfaces for *Logs*, *Log managers*, and *Log factories*, and the structure of *Log records* and *Log events*. A log factory is a log manager which is also a consumer Admin object as defined by the event or notification service. *Event logs* or *Notification logs* are event or notification channels as defined by the event or notification service. Notification logs can have associated filters. Event or notification log factories are collection managers for event or notification logs and manage supplier Proxy objects for their logs (i.e., event or notification channels) with associated filters in the notification case. The telecom log service also defines typed log records and typed event or notification logs. By leveraging features of event and notification channels, event and notification logs can form a federated log network that supports "log-and-forward" scenarios. Built-in lightweight features of the telecom log service are those of the event service, i.e., disabling the pull model and typed events. The OMG *lightweight log service* [35] is not at all suitable for telecom applications.

The fine-grained and coarse-grained frameworks state for the notification part of the telecom log service the same requirements as for a plain notification service and provide a profile for the operations of the log part. The service-oriented framework requires almost the same usage and recommends that managing systems use logs owned by managed systems in a more lightweight

fashion. It adds the 3GPP notification log IRP [25] and some clarifications with regard to the representation of log events and of event data in log records. Refer to clause 8.4 for details.

6.7 Relationship to coarse-grained interface design considerations

A service-oriented interface design provides considerably more flexibility than a coarse-grained interface design. Therefore a given service-oriented design need not be specializable to a coarse-grained design. But for the following reasons, a given coarse-grained design can always also be considered a service-oriented design:

- a façade, as defined in ITU-T Rec. X.780.1, is a service-oriented façade, as defined by this Recommendation, if and only if its allocated managed objects (all of which instantiate the same MOC) are all service-oriented, i.e., are only accessible via the façade;
- ORB usage as specified by clauses 5.2/Q.816 and 6.4/Q.816 for fine-grained interfaces, and adopted by ITU-T Rec. Q.816.1 for coarse-grained interfaces, is an option for service-oriented interfaces though a more lightweight use of the ORB is preferred (see clause 6.2);
- use of the OMG naming service on coarse-grained interfaces (see clause 6.1/Q.816 and clause 8.1/Q.816.1) is an option for service-oriented interfaces though a more lightweight or even minimalistic use of the Naming Service is preferred (see clause 6.4);
- use of the notification, telecom log, transaction and security OMG services as specified by clauses 6.2/Q.816, 6.3/Q.816, 6.6/Q.816 and 6.5/Q.816 for fine-grained interfaces, and adopted by ITU-T Rec. Q.816.1 for coarse-grained interfaces, are options for service-oriented interfaces though a more lightweight use is preferred (see clauses 6.5 and 6.6 and clause 8);
- use of the factory finder, channel finder, terminator, basic MOO, advanced MOO, heartbeat and containment ITU-T services as specified by subclauses of clause 7/Q.816 and clause 9/Q.816.1 for fine-grained and coarse-grained interfaces are not needed for service-oriented interfaces when other options of the SO framework are chosen (see clause 7.2.5).

Therefore when a TMN interface is designed according to the coarse-grained framework (which depends on the fine-grained framework), a service-oriented interface design will result in any case. By choosing in the course of the design process suitable options of the service-oriented framework (e.g., lightweight use of CORBA services), an appropriate lightweight design will result.

7 Service-oriented framework and requirements overview

Clause 6 outlined the design considerations that should be resolved as support for service-oriented interfaces is added to the framework. This clause and the rest of this Recommendation provide details on how the framework will be extended to address these issues. This Recommendation focuses on the framework support services for service-oriented interfaces, while ITU-T Rec. X.780.2 defines guidelines for developing or adopting/endorsing information models for service-oriented interfaces including lightweight generic façades to access and control managed objects. First, an overview of the service-oriented framework is presented, then an overview of its constituents.

7.1 Framework overview

The service-oriented framework for CORBA-based TMN interfaces is a collection of capabilities. A central piece of the framework is a set of lightweight OMG common object services. The framework defines their role in network management interfaces, and defines conventions for their use. The framework also defines lightweight support services that have not been standardized as OMG common object services but, as a rule, are expected to be standard on network management

interfaces conforming to the service-oriented framework. IDL interfaces for these services are defined in Annex A. To support the software objects representing manageable resources, the framework recommends that they implement some *common* basic capabilities. Therefore, two base classes are defined in ITU-T Rec. X.780.2 for use in modelling the state and behaviour of network management resources. Managed and managing object classes in IDL models shall either inherit and implement a basic set of capabilities from these superclasses in order to operate within this framework, or refer to a well-established and standardized information model that specifies the *common* attributes of managed objects and the *common* operations of managing objects.

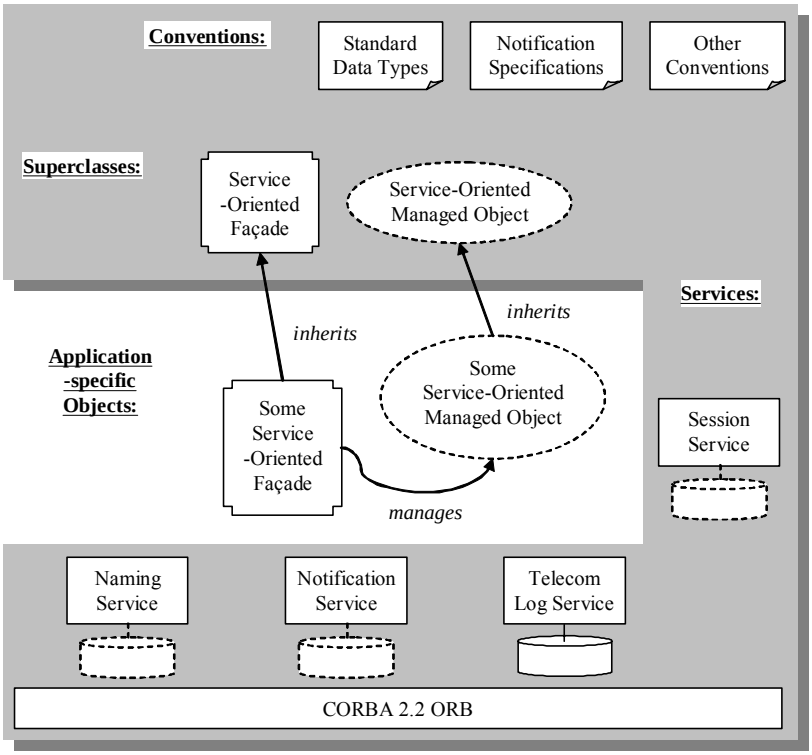

Figure 3 – Overview of service-oriented framework

The framework is depicted graphically in Figure 3 above. The figure shows the framework in grey. In the middle are the **application-specific objects** that are supported by the framework. Since the service-oriented framework is capable of supporting any information model (see clause 1.2), only *some* service-oriented façade that manages *some* managed object (resource) are shown.

Along the bottom is a box representing the CORBA ORB. Above that are a number of boxes with names in them representing the **CORBA-based TMN services** that are part of the service-oriented framework. Some also have icons depicting the databases they would have to maintain to perform their functions. The managed object name database maintained by the naming service is shown with dotted lines, indicating that it need not store the names and IORs of managed objects. The naming service is still required, however, to enable managing systems to find façade interfaces and support service references or at least an entry point to the managed system. The authentication and access log database of the session service is optional. The use of a database by the notification service to persistently store events, in case of connection loss between a channel and a consumer and configured guaranteed delivery, is optional. The use of these CORBA services, along with ORB version requirements, are defined in this Recommendation.

Along the top of the figure are icons representing two **superclasses**, one for managed objects and one for managing objects (service-oriented façades), which are derived from each specified information model. Each of the managed objects and managing objects supported by this framework shall ultimately inherit from these superclasses, respectively. The managed objects are shown drawn with dotted lines, to indicate that they need not be directly accessible. Also shown on the figure are icons of pages with up-turned corners representing standard object modelling **rules and conventions** that are defined for information modellers developing IDL models for use with this framework. These conventions and the two superclasses are defined in ITU-T Rec. X.780.2, along with a couple of other service-oriented information modelling guidelines.

7.2 Framework constituents overview

The *CORBA services-related* constituents of the framework (ORB, OMG services, ITU-T services) are described in this Recommendation and its *CORBA IDL-related* constituents are described in the companion ITU-T Rec. X.780.2 (superclasses, rules and conventions, naming of managed objects and service-oriented façades, service-oriented CORBA modelling guidelines, IDL repertoire and style guide for use by IDL modellers, modelling of multiple transmission technologies according to ITU-T Recs G.805 and G.809, standardized modelling of extensions, *et al.*).

7.2.1 Façade design pattern and service-oriented façades

The most significant change to the fine-grained framework required to support coarse-grained interfaces is the way managed objects are accessed and controlled, namely through the use of the façade design pattern (see clause 6.1). Using the façade design pattern, a managed system will support a small number of façade interfaces, at least one but usually no more than a few for each type of managed object on the system. A managing system will then (logically) invoke an operation on a managed object by actually invoking the operation on a façade for that type of managed object on that system. In the façade design pattern, the managed objects do not have to expose a CORBA interface and hence may not have individual IORs. This means a managed system that supports the façade approach does not need to implement the fine-grained managed object interfaces. These principles of the coarse-grained framework are retained for the service-oriented framework in a more flexible way with additional options.

It is best to think of a service-oriented façade, abbreviated as "SO façade", not as a managed object, though it is managed by a managing system, but as an intermediary object that enables a managing system to manage proper managed objects representing manageable resources. A service-oriented façade is therefore also called a "managing object" or a "manager". The façade object has a CORBA interface and is accessible using CORBA. The proper managed objects, however, usually do not have CORBA interfaces and so are not directly accessible using CORBA. The façade itself does not represent a manageable network resource; its purpose is to enable or facilitate interaction with the objects that do represent manageable resources. A façade exposes behaviour of the managed objects it "manages", i.e., controls and provides access to, but may also expose own behaviour (e.g., present its capabilities). All façade objects are instantiated by the managed system during startup or restart or during session instantiation, and destroyed during shutdown or ending of the session. There are no service-oriented façade factories. Multiple façades for the same type of managed objects may exist on a coarse-grained interface, but usually not on a service-oriented interface. Any managed object shall always be accessible through only one (and always the same) façade. Figure 4 below summarizes *the managing and managed entities of the service-oriented framework*, i.e., the managing and managed system, the managed objects and the façades.

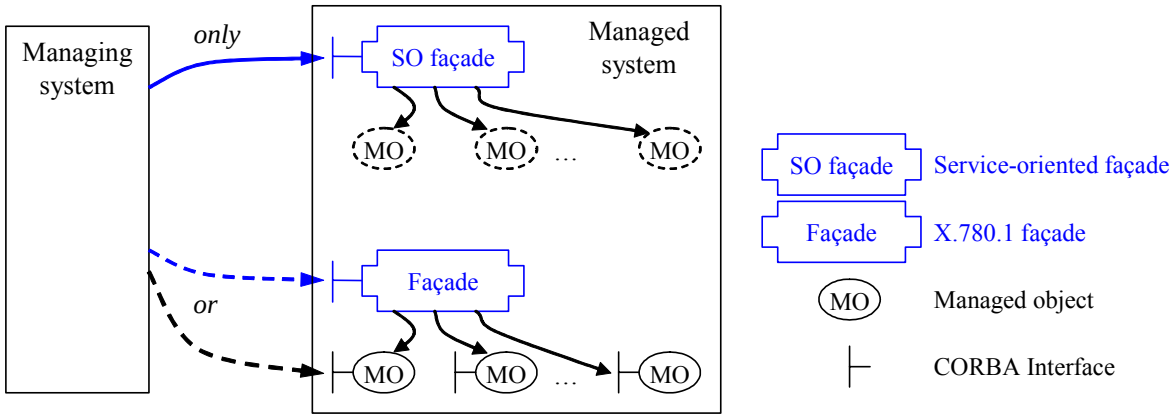

Figure 4 – SO façade and façade of a managed system

The figure shows a managing system accessing a managed system that supports the service-oriented approach. The managed system has two façade interface instances that enable the managing system to access two different sets of managed objects. The managed objects at the top of the figure can only be accessed through the SO façade. The managed objects at the bottom also support direct CORBA interfaces and can be accessed either through the façade or directly. They are not service-oriented. Direct CORBA access of managed objects is optional and only specified for reasons of compatibility with the coarse-grained framework. A managed system that supports the real service-oriented façade approach shall provide SO façade interfaces (i.e., managing object classes) for each of its managed object classes and instantiate only SO managed objects.

A façade may use a managed object's CORBA interface, if available, to invoke an operation on it, or some other implementation-specific means. How SO façades invoke operations on SO managed objects is not exposed at the interface. A managed system, in fact, need not even implement managed objects as individual objects internally. By implementing a CORBA TMN interface based on this framework, however, it will give the illusion that managed objects are internally implemented as objects since they are exposed at the interface as objects of their own.

When an operation is invoked on a managed object through a façade, the façade must then logically invoke the operation on the actual managed object. Because very many managed objects will be accessed through a single façade, the façade must know which managed object is the actual target of the operation. This will be handled by adopting the convention of including the name of the target managed object as the first parameter of every façade operation directed at a managed object. See for example the set functions of the *Common_I* interface in Annex A/X.780.2.

7.2.1.1 Relating managed objects to their service-oriented façade

The coarse-grained framework defines a means to determine the assigned façade of a managed object solely from its name by extending the name definition. The service-oriented framework supports this mechanism but also defines a lightweight alternative through a set of information modelling rules. Refer to clause 7.2.1.1 of the companion ITU-T Rec. X.780.2 for details.

7.2.1.2 Relationship to SOA concepts

The service-oriented framework specifies a novel approach to CORBA TMN interface design that paves the way for introducing SOA principles [29] to the TMN interface specification methodology. A *service* (i.e., an SOA service) is a means by which the needs of a consumer are brought together with the capabilities of a provider, and which is realized as a set of behaviours that are accessible

through, or are using, one or more service interfaces. Therefore a *service interface* (SI) is a means for interacting with the service that exposes this SI for the purpose of accessing all or part of the service's underlying capabilities, or indicating all or part of the service's needs. SIs are the most important part of a *service description* (SD), which provides all information needed in order to use, or consider using, a service. When an SD includes constraints and policies, it is called a *service contract*, and its SIs are termed contract-defined or contractual. SOAs incarnate any service as one or more entities that expose SIs. A *service resource* (SR) is defined here as a physical or logical source or sink of data, or a state-aware synchronous or asynchronous data processing capability. SRs are used by services to deliver and maintain data. When the data is specified by an information model, SRs can be used to construct SIs and processable SDs (i.e., service metadata).

SOAs introduce two views on services (and resources): the outward view of a *service provider* (SP), who implements and provides interfaces (depicted as balls or lollipops), and the inward view of a *service consumer* (SC), who requires and consumes interfaces (depicted as sockets or crescents). A service may assume either a client/consumer role or a server/provider role with respect to another service, depending on situation and intended use. A service in a provider role constitutes an action boundary with regard to access and control between the allocated service resources and potential consumers of the resources. A provided service interface delineates and exposes an external view of functionality of a service and defines all or part of that service's action boundary.

The *conceptual metamodel of an SOA architectural style* is based on the concepts of consumer and provider that interact according to the publish-locate-[negotiate-]bind-execute paradigm:

- SPs *publish* SDs, or optionally register SDs with a *service broker* (SB) (also known as *service registry*). SPs own and maintain their published or registered SDs.
- An interested SC can *locate* an SD, which always comes with one or more SIs, by using either public location information obtained from the SP or optionally a public SB to find the SD.
- The SC parametrizes the SIs according to the SD, or optionally *negotiates* parameters for the behaviours of the needed SIs, in order to *bind* the behaviours and establish a concrete service contract with the SP. The SC then establishes a technology-specific *binding* to a (secure) transport mechanism supported by the SP (according to the SD). Finally the SC invokes the needed behaviours of SIs, in order to *execute* service capabilities. The SP does not disclose how the invocation of a behaviour is implemented, delegated or composed.
- That way the service capabilities provided by the SP through the SIs of the published service (description) can be consumed by the SC according to the SD. The capabilities allow the SC to access and control the SRs that are allocated to the service (according to the SD).

The distinction between service interfaces and service resources reflects the **SOA principle** of separating the definition and storage of (shared) data (through service resources) from the delivery of data (through service interfaces) (see clause 6.1). A service-oriented managed object, as defined by this Recommendation, is a *managed entity* that represents a manageable service resource in terms of shared state (attributes) and behaviour (operations, notifications, transactions) where state and behaviour are separated through outsourcing of the behaviour to an assigned so-called "*managing entity*" (e.g., a service and its SIs) that takes a steward role with regard to the behaviours of its allocated managed entities. Another **SOA principle** is loose coupling of service components that make up a composite service and interact according to the publish-locate-[negotiate-]bind-execute paradigm across internal SIs. This implies the potentiality of dynamic binding, unbinding and rebinding at run time between an implemented and provided SI and a fitting required and consumed SI according to an orchestration conducted by the underlying business processes.

The presented SOA artefacts are depicted graphically in Figure 5 below.

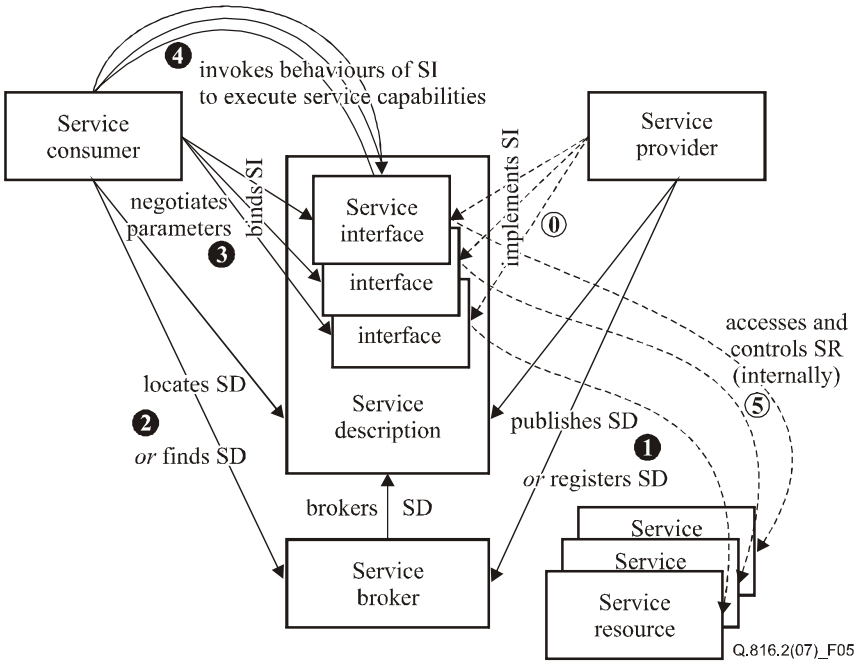

Figure 5 – Conceptual metamodel of an SOA architectural style

The SOA concepts correspond to the concepts of the service-oriented TMN framework as follows:

- service provider ≈ managed system (server);
- service consumer ≈ managing system (client);
- SOA service ≈ set of one or more service-oriented façade interface instances (i.e., managing objects), or other CORBA interface instances (i.e., CORBA objects);

  NOTE – For example, any CORBA interface instance that inherits from the *Common* interface (i.e., the service-oriented façade superclass – see [ITU-T Rec. X.780.2]) is an SOA service. Another example of an SOA service is the session service that implements an *EMS session factory* interface, an *EMS session* interface and an *NMS session* interface (see clause 7.2.3). Further examples of SOA services are OMG's common object services [34] that implement one or more CORBA interfaces.
- service interface ≈ CORBA IDL interface;
- service description with associated service interfaces/contracts ≈ all CORBA IDL interfaces of the façades, or other CORBA objects, that make up the service;
- service resource ≈ SO managed object (i.e., manageable (network) resource that is accessible and provides behaviour only through its assigned SOA service);
- service broker ≈ ORB together with naming service or session service (or trading service).

Applying this mapping to Figure 4 results in Figure 6 below. Note that direct access to service resources is not allowed for real SOAs.

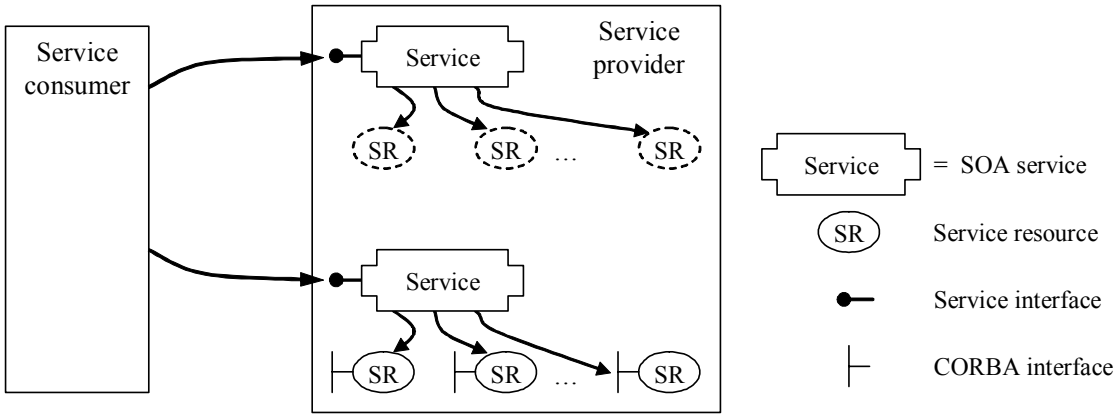

Figure 6 – SOA concepts of the service-oriented TMN framework

The managed system is therefore *decomposed into services* and the TMN interface offered by the managed system is *decomposed into provided service interfaces* (lollipops). The managing system is not decomposed, i.e., required service interfaces (crescents) are not explicitly modelled.

7.2.2 ORB version and IDL repertoire

Clause 6.2 describes the two aspects of CORBA, to be provided by ORB vendors, that are most important for the TMN frameworks: *the IDL modelling aspect* according to the supported IDL repertoire and *the CORBA services aspect* according to the supported ORB system interfaces. The scope of each aspect depends on the ORB version, and the three Revisions 2.2 [12], 2.3 [13] and 2.4 [14] were identified as basic ORB versions to be considered for particular features of the aspects.

The service-oriented framework mandatorily requires the *use of the POA as defined by Minimum CORBA*, and so requires at least CORBA Revision 2.2. Further features of the two aspects that are beyond CORBA 2.2 (e.g., use of *value types* or *CORBA Messaging*) are optional.

7.2.3 Session service

Refer to clause 6.3 for a general description of the session service that applies to any managing system and managed system. This clause describes in more detail the major example where the managing system is an NMS and the managed system is an EMS.

A session between a managing system (NMS) and a managed system (EMS) consists of a pair of mutually associated session objects, an *NMS session* that is instantiated at the NMS and an *EMS session* that is instantiated at the EMS on request from the NMS. To enable such requests, the EMS registers an *EMS session factory* with the naming service, which then serves as the unique entry point to the EMS for all NMSs. The NMS invokes the `getEmsSession` operation of the EMS session factory using the NMS session and authenication data (user name and password) as input parameters, and the EMS returns the associated EMS session. The inheritance and containment relationships of these CORBA interfaces are shown in Figure 7 below.

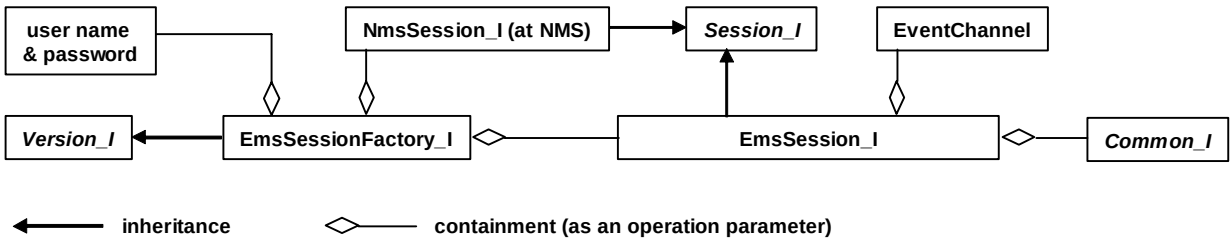

Figure 7 – Session service interfaces and common interface

Each manager interface (i.e., managing object class) that is supported by the EMS inherits from the *Common* interface and is instantiated as *singleton*. Refer to the companion ITU-T Rec. X.780.2 for details. An EMS session provides operations to:

- query the names of the supported manager interfaces;
- retrieve the IOR of each supported manager in order to gain access to it without using the naming service;
- retrieve the IOR of the (session-specific) event channel to be used by the NMS.

The session service also includes an extensible security model for client authentication to the server, a versioning concept, a kind of heartbeat service, a capability model that allows to check at run time the availability of managers and their operations, and vendor-specific extensibility of session capabilities through extensions of the EMS session or NMS session interfaces. Refer to clause 9.1 for a detailed description of the session service requirements.

7.2.4 OMG CORBAservices and CORBAsecurity

The service-oriented framework is dedicated to identifying lightweight usage options of the ORB and certain CORBAservices in support of CORBA-based service-oriented TMN interfaces. These CORBAservices include all services considered by the fine-grained and coarse-grained frameworks. Clause 6 summarizes lightweight design considerations for the naming, notification/event and telecom log services [17], [19] and [20], which are detailed in clause 8. Clause 8 also states prioritized usage recommendations for the transaction and security services [37] and [38], which do not require specific design considerations. They are not shown in Figure 3 since their use is entirely optional.

These services can be applied only to CORBA objects, and so they only affect façade objects in case of coarse-grained or service-oriented TMN interfaces where managed objects are protected indirectly through façade protection and transaction capabilities for them would be defined through façades (which act as resource managers). CORBA's rich transaction and security capabilities should be prioritized according to resulting complexity, and clauses 8.5 and 8.6 provide such recommendations on lightweight use of these services, in particular additional use of the concurrency control service (CCS) and the wider scope of CORBAsecurity beyond the current security service capabilities.

For example, the security issue of the notification service mentioned in clause 6.5 (see also section 6.3 of [24]) is solved by requiring that managing systems cannot access event channel factories (see 8.3/**NOTIF-2**) and transactional event transmission by the notification service is considered very heavyweight (see 8.5/**TRANS-1**). The lightweight security capabilities of the ITU-T session service (see 9.1.2.5/**SESSION-7**) are complementary to CORBAsecurity.

7.2.5 ITU-T support services

Figure 3 does not show any of the ITU-T support services of ITU-T Recs Q.816 and Q.816.1 (i.e., factory finder, channel finder, terminator, basic MOO, advanced MOO, heartbeat and containment service) since the use of all of these services can be completely optional on real service-oriented TMN interfaces due to the availability of other, more lightweight options.

Similar to the naming service's `NamingContext` interface, which can create further contexts, SO façades could act themselves as factories for their interface type if needed. They can also act as factories for their allocated managed object classes if it is required that the managing system can instantiate such MOCs (see also clause 8.4/X.780.1). Therefore, factories and the *Factory Finder Service* are not needed for the SO framework. Similarly the *Terminator Service* is not needed.

The SO framework recommends using as few event channels as possible. If the managed system implements a session service, each session has a dedicated unique event channel that is provided to the managing system through the associated server session object. If no session service is available, the event channels are registered with the naming service at prescribed locations close to the Initial naming context instead of registering a *Channel Finder Service* at each local root naming context (see clause 6.4.3). Therefore the channel finder service is not needed.

The *Heartbeat Service* is optional for the fine-grained and coarse-grained frameworks. The session service of the SO framework offers a kind of heartbeat service that could be extended with heartbeat notifications and heartbeat period operations if need be (see clause 9.1.1).

The SO framework allows for the SO façade objects as storage locations of managed object names and containment relationships. This option is a very lightweight means of name administration, which adds an element of federation and eliminates the threat for performance issues with the *Containment Service*. The SO framework federates the administration of names and containments across multiple SO façades instead of maintaining a central service that is a potential bottleneck. The containment service is basically a kind of naming service, which is well-known for bad performance when umpteen millions of managed objects are present. Therefore, the Containment Service is not needed, and not recommended, for the service-oriented framework.

Since the telecom industry has invested a great deal of effort in the development of information models for the CMIP network management protocol, a primary goal of the fine-grained and coarse-grained frameworks is the reuse of CMIP models by enabling their translation to CORBA IDL with very little change in semantics. Major examples of such reuse are the *Basic MOO Service*, which provides a scoped get operation, and the *Advanced MOO Service*, which provides in addition a scoped update operation and a scoped delete operation. Here "scope" refers to containment up to a specified level (and therefore covers managed objects from different MOCs), and all scoped operations provide filtering and attribute selection options. By contrast the get, update and delete capabilities for SO managed objects do not adhere to the CMIP paradigm but follow an efficient lightweight approach. Refer to clauses 6.4, 6.7 and 9.3 of the companion ITU-T Rec. X.780.2 for details.

7.3 Relationships between service-oriented, coarse-grained and fine-grained frameworks

Clause 6.7 describes that a coarse-grained design can always be considered a service-oriented design. Clause 7/Q.816.1 describes how the fine-grained framework is extended by the constituents of the coarse-grained framework, and depicts this relationship in Figure 3/Q.816.1. This clause provides a similar figure that depicts how the artefacts of the service-oriented framework, as shown in Figure 3 above, map to the constituents of the coarse-grained and fine-grained frameworks, as shown in Figure 3/Q.816.1.

Figure 8 below shows the constituents of all three frameworks for CORBA-based TMN interfaces: fine-grained parts are shown **in black**, coarse-grained parts are shown **in blue**, and service-oriented parts are shown **in green**. The figure shows dotted lines drawn around groupings of framework

constituents, inheritance relationships, and the specific service-oriented artefacts. The following constituent groups and further framework components are depicted:

- **Services (CBTS Q.816, CCBTS Q.816.1, SOCBTS Q.816.2)**: protocol requirements, CORBA common object service (COS) usage requirements, and TMN-specific support services for basic capabilities of managed systems, including support of coarse-grained and SO interfaces, regardless of the type of network technology being managed.
- **Superclasses (GDCMO X.780, GDCCMO X.780.1, GDSOCMO X.780.2)**:
  - root class `itut_x780::ManagedObject` from which all managed objects inherit core capabilities they shall implement to get enabled for operation within the framework;
  - generic managed object factory interface `itut_x780::ManagedObjectFactory` from which all MO factories should inherit to be able to operate within the framework;
  - root class `itut_x780::ManagedObject_F` from which all X.780.1 façades inherit basic capabilities they shall support for operation within the framework;
  - common attributes `common::Common_T` lightly inherited by all service-oriented managed object classes (name, user label, owner, native EMS name, additional info parameters);
  - an abstract common façade `common::Common_I` for the management of common attributes and individual capabilities from which all service-oriented façades shall inherit.
- **Inheritance relationships**: **black**, **blue** and **green** arrows indicate inheritance relationships of IDL interfaces or valuetypes, or light inheritances of IDL structs, from the respective (abstract) superclass.
- **Conventions (GDCMO X.780, GDCCMO X.780.1, GDSOCMO X.780.2)**: rules and conventions for defining application-specific managed object classes and façades; GDCMO translates GDMO/ASN.1 (X.721/X.722) to IDL, which is particularly important for the basic and advanced MOO services, and GDCCMO merely extends GDCMO so that the fine-grained approach becomes a mandatory prerequisite of the coarse-grained approach; GDSOCMO does not use the "GDMO to IDL" rules but defines service-oriented conventions for "Standard Data Types" and "Notification Specifications" as well as service-oriented "Other Conventions", which introduce lightweight and multi-technology principles to the framework.
- **Application-specific objects (ITU-T Recs M.3120, M.3170.3, etc.)**:
  - **ITU-T Rec. M.3120**: catalogue of generic MOCs and façade interfaces at NE and network level that lacks traits specific to any particular network technology but defines common concepts by mainly translating them from M.3100; some MOCs (e.g., ME, equipment shelf, circuit pack) can be used without further specialization, other MOCs (e.g., link, network, connection, TP) are abstract classes and must be specialized with traits specific to a particular network technology (e.g., ATM, DSL, BPON) before inclusion in a commercial interface specification.
  - **ITU-T Rec. M.3170.3**: MTNM adds MOCs and service-oriented façade interfaces, each derived from the respective X.780.2 superclass; while M.3120's NE and network MOCs and façades are generic and mostly network-technology agnostic, M.3170.3's MOCs and façades are multi-technology aware and capable (see clause 8.4/X.780.2), including an extension mechanism to add vendor-specific technologies and details (see clause 10.5/X.780.2), and they support all network technologies specified by the respective MTNM version in a unified way.
  - **Multi-technology nature of the service-oriented framework**: the SO framework removes the tiresome restriction to single-layer networks within TMN interface

specifications of the fine-grained and coarse-grained frameworks community by adding architectural elements that encompass multiple adjacent G.805/G.809 layer networks (see clause 8.4/X.780.2); such multi-layer NE and network views avoid modelling of different managed entities for each layer network and allow for the unified multi-technology management of NEs and networks.

- **Service-oriented interface design principles (SOCBTS Q.816.2, GDSOCMO X.780.2)**:

  SOCBTS and GDSOCMO include service-oriented interface design considerations that specify the lightweight use of CORBA-based services and IDL constructs; this is indicated in the figure by block arrows and ovals with a fill effect; **block arrows** indicate the transition from a coarse-grained design to a service-oriented design, and **ovals with a fill effect** indicate the service-oriented usage of services and conventions.

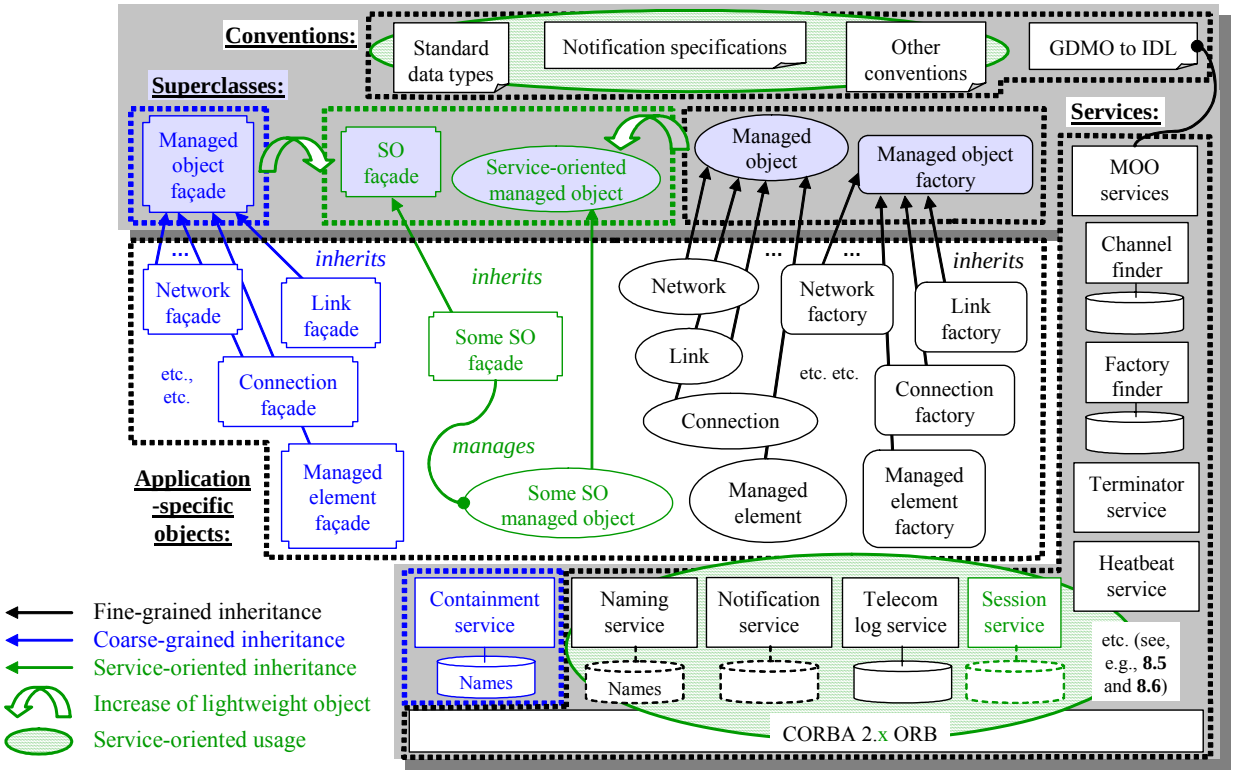

Figure 8 – Relationship of service-oriented and coarse-grained frameworks

The flexibility of the service-oriented interface design principles allows mapping of the service-oriented framework to other CORBA design approaches chosen by the telecom industry, notably the *MTNM model* of TM Forum's MTNM Team and the *IRP model* of 3GPP's OAM&P working group SA5. These model mappings are out of scope of the CORBA framework but are an essential part of the proof of TMN conformance according to ITU-T Rec. M.3010 for these models. In case of MTNM, this proof is carried out in ITU-T Rec. M.3170.3: "MTNM – CORBA IDL solution set (TMF814) with implementation statement templates and guidelines (TMF814A)".

8 Framework ORB and common object services usage requirements for supporting service-oriented interfaces

ITU-T Rec. Q.816 describes how the original framework includes several of the OMG's common object services. These are services defined by the OMG for use in generally any CORBA application. The (fine-grained) framework defines which of the OMG's common object services

shall be supported by a managed (or managing) system, and conventions on their use. ITU-T Rec. Q.816.1 provides additional conventions and requirements needed to support coarse-grained interfaces for each common object service (COS) in the framework. This clause provides further conventions and requirements needed to support service-oriented interfaces for each COS in the framework. It also provides conventions on the use of certain ORB capabilities including CORBA messaging, which is considered an ORB capability set and not a COS.

8.1 ORB usage requirements

The fine-grained and coarse-grained frameworks do not state explicit ORB usage requirements, neither regarding the IDL repertoire nor regarding the basic system services, but do require support of ORB v2.3.1 (see clause 8.1.2/Q.816) since support of the POA and value types are considered important (see clause 5.2/Q.816). The fine-grained and coarse-grained frameworks also include some optional requirements on CORBA messaging (see clause 6.4/Q.816). Considering the importance of dynamic and asynchronous capabilities for the SOA paradigm and the need to enable lightweight designs, the service-oriented framework breaks down the basic ORB capabilities, including CORBA messaging, into mandatory requirements and prioritized optional requirements.

(R) ORB-1 An operations system shall support CORBA 2.2 [12] and use the *Portable Object Adapter* [30] as defined by the *Minimum CORBA* specification [32].

(O) ORB-2 An operations system may use dynamic mode of POA operation and other dynamic aspects of CORBA omitted by the *Minimum CORBA* specification [32].

(O) ORB-3 An operations system may support value types as defined by the *Objects By Value* specification [31]. This requires support of CORBA 2.3.1 [13].

(O) ORB-4 As a rule, an operations system shall use only IDL interfaces that are synchronous. In exceptional and provisional cases, an operations system may use one-way operations with application-specific callback objects, and in these cases a synchronization policy as defined by the *CORBA Messaging* specification [33] should be implemented.

(O) ORB-5 An operations system may use transient asynchronous requests as defined by the *Asynchronous Messaging Invocation* (AMI) specification [33] and further detailed in clause 6.4/Q.816. This requires support of CORBA 2.4.2 [14].

(O) ORB-6 An operations system may use persistent asynchronous requests as defined by the *Time-Independent Invocation* (TII) specification [33]. This requires support of CORBA 2.4 [14].

NOTE – ORB usage requirements regarding the IDL repertoire and modelling beyond **ORB-3**, e.g., the alternative ways of MOC specification outlined in clause 6.2.2, are stated in ITU-T Rec. X.780.2.

8.2 Naming service

The fine-grained framework uses CORBA names and the naming service to define unique distinguished names (DNs) and store containment relationships between managed objects, introduces one or more *local root naming context*s besides the initial naming context, assigns a CORBA name to each local root naming context to achieve globally unique names per managed system, defines an LDAP DN string translation for (compound) CORBA names that complements the stringification capabilities of the (heavyweight) naming service, and specifies a naming service profile (see clause 6.1/Q.816, Q.816/Amd.2, and clause C.1 of Q.816/Amd.1). The coarse-grained framework adds naming rules for façades (see clause 8.1/Q.816.1).

The service-oriented framework adds the options to consider most parts of the lightweight naming service, to define lightweight naming trees, and even to adopt a minimalistic use of the naming service. This very lightweight use of the naming service is defined by the requirements below and called *Service-Oriented Naming Service*.

8.2.1 Names of managed objects and service-oriented façade objects

Service-oriented managed objects are identified by distinguished names whose structure is identical with the structure of CORBA names (see clause 8.5/X.780.2) but these names are not stored in a naming service since SO managed objects do not possess an IOR. Service-oriented façades always possess an IOR but whether they also have a name, which then is stored in a naming service, or do not have a name depends on the implementation of the session service by the managed system. If the session service is available, only server session factories are registered with the naming service, and server session objects then provide access to service-oriented façades and event channels. If the session service is not available, every service-oriented façade and event channel is registered with the naming service at the same locations where server session factories would be located.

8.2.2 Naming service usage requirements for SO interfaces

This clause defines the naming service-related requirements operations systems must meet when using service-oriented interfaces.

(R) NAME-1 An operations system shall use only the lightweight naming service capabilities [16] and [17] except that the `list` operation shall also be available for use. In case of reasonably small naming trees, it should generally be possible to list any naming context without using a binding iterator. It may also be required that the `NamingContextExt` interface is available for use.

(O) NAME-2 An operations system shall construct and/or use a lightweight naming tree, which has only one local root naming context, namely the initial naming context, and uses only the kinds of name components specified in clause 6.4.3, and only in the specified hierarchical order. This usage includes a multi-vendor capability, a versioning capability, an OS dependency capability, and naming rules for server session factories and service-oriented façades as well as event channels.

8.3 Notification service

The notification service [19], which enhances the brilliant but limited event service [18], allows for decoupled communications between managed systems and managing systems, according to the publish-subscribe pattern, and offers a rich set of powerful capabilities that are briefly summarized in clause 6.5. The fine-grained framework recommends and prioritizes a number of these capabilities and specifies how the managed system shall use them to communicate notifications of CORBA managed objects to managing systems including the definition of all notification details. The coarse-grained framework recommends the same usage of the notification service to convey notifications from IDL-defined managed objects that are accessible through a CORBA façade.

The service-oriented framework recommends a similar usage but with a stronger emphasis of the lightweight features, and adds further lightweight options. This clause presents the notification service requirements of the fine-grained, coarse-grained and service-oriented frameworks in a unified manner. It specifies formal mandatory and optional requirements "**(R) NOTIF-x**" and "**(O) NOTIF-x**", which refer as appropriate to the corrected requirements "**(R) NOTIF-x**" of clause 6.2/Q.816, clause 6.2 of ITU-T Q.816/Cor.1, and clause 6.2 of ITU-T Q.816/Cor.2.

(R) NOTIF-1 An operations system shall use a notification service, which is conformant with the OMG specification (see section 1.2 of [19]), except only for the support of the interfaces for the pull model. A managed system shall act as a push supplier to convey notifications from fine-grained or coarse-grained or service-oriented managed objects to managing systems.

NOTE 1 – This **NOTIF-1** requirement includes **NOTIF-1**/Q.816 and **NOTIF-11**/Q.816. It does not include the condition "When the OMG Notification Service is used as the Notification Service", of **NOTIF-11** of ITU-T Q.816/Cor.1, since notification service usage requirements cannot encompass the non-usage of the notification service. The non-usage option is part of the relevant conformance point (see clause 10.2.1).

NOTE 2 – While **NOTIF-1** requires support of most interfaces of the notification service, not all operations of an interface need be implemented but some could raise the standard exception `NO_IMPLEMENT`, and this behaviour could depend on the credentials of the client on a per-interface or per-operation basis.

NOTE 3 – Restriction of access to the `CosNotifyChannelAdmin::EventChannelFactory` interface for individual operations systems is implementation-defined (see clause 7.2.4 and **NOTIF-2**).

NOTE 4 – Interfaces, operations, attributes and constants, which are mandatorily or optionally required for use by the managing system, have been collected in clauses C.2 and C.3 of Q.816/Amd.1 as *OMG Notification Service profile* and *OMG Event Service profile*. Capabilities that are required for use by the managed system (e.g., `CosNotifyComm::StructuredPushConsumer::push_structured_event()`, `CosNotifyComm::NotifyPublish::offer_change()`) are not listed in these annexes.

(R) NOTIF-2 A managed system shall instantiate one or more notification channels (i.e., instances of `CosNotifyChannelAdmin::EventChannel` [19], or – in case of fine-grained and coarse-grained interfaces – of `CosTypedNotifyChannelAdmin::TypedEventChannel` [19], or of notification log interfaces (see clause 8.4)). It may use any notification channel together with an instance of the `NotificationIRPSystem::NotificationIRP` interface defined by 3GPP [24] or an instance of the `emsSession::EmsSession_I` interface (see clause 9.1.2.2). A managing system shall not have access to notification channel factories (nor to notification log factories).

NOTE 5 – This **NOTIF-2** requirement includes **NOTIF-2**/Q.816.

(R) NOTIF-3 If the channel finder service is available (e.g., in case of fine-grained and coarse-grained interfaces), each notification channel shall be registered with this service. If the session service is available, each session shall have its unique notification channel (see clause 9.1.2.2). Otherwise, each notification channel shall be registered with the naming service (see, for example, clause 6.4.3). The managed system may restrict the use of an individual notification channel, without using event filtering capabilities, to certain event types or to certain managed object classes.

NOTE 6 – This **NOTIF-3** requirement includes **NOTIF-3**/Q.816.

(R) NOTIF-4 The notification service shall support either structured events or structured and typed events. The managed system, however, must supply either only structured notifications or only typed notifications, irrespective of the notification channel used. The use of event batches (i.e., sequences of structured events) is optional. Generic events (i.e., Anys) must not be used. In case of fine-grained and coarse-grained interfaces, typed events are the preferred choice, and all necessary prerequisites for using typed events and translating between typed and structured events are provided (see [ITU-T Q.816] for details). In case of service-oriented interfaces, structured events are the preferred choice. The overall structure and contents of structured events is described in **NOTIF-10** with further details being added by ITU-T Recs Q.816 and X.780 in case of fine-grained and coarse-grained interfaces, and ITU-T Rec. X.780.2 in case of service-oriented interfaces.

NOTE 7 – This **NOTIF-4** requirement includes **NOTIF-4**/Q.816 and **NOTIF-6**/Q.816, as corrected by ITU-T Q.816/Cor.2, as well as **NOTIF-5**/Q.816.

(R) NOTIF-5 Conformance with the OMG notification service specification [19] includes support of the interfaces defined in the CORBA module `CosNotifyFilter` and support of the *default Notification Service constraint language*, i.e., support for event filtering with filter objects, and mapping filter objects, that encapsulate constraints expressed in the default filtering constraint grammar and determine whether events are to be forwarded. An operations system shall use such filter objects to the extent supported by the deployed notification service (i.e., creating and destroying filters, writing filter constraints based on event structure and content, configuring filter constraints and callbacks, adding filters to and removing filters from Proxy and Admin objects). Use of mapping filter objects, which affect QoS and Admin properties of events, is optional.

NOTE 8 – This **NOTIF-5** requirement includes **NOTIF-8**/Q.816, as corrected by ITU-T Rec. Q.816/Cor.1.

NOTE 9 – The scoped operations of the basic and advanced MOO services to be used for fine-grained and coarse-grained TMN interfaces (see clause 7.4/Q.816 and clause 9.4/Q.816.1) provide filtering and attribute selection options that are specified with the *MOO Service constraint language* (see clause 7.4.2/Q.816). This language is defined as an extension of the default notification service constraint grammar defined by [19].

(O) NOTIF-6 Conformance with the OMG notification service specification [19] includes optional support of the interfaces defined in the CORBA modules `CosNotifyCommAck` and `CosNotifyChannelAdminAck`, i.e., support for event acknowledgement based on sequence numbers that allows for reliable event delivery. It is recommended that the managed system uses these capabilities to the extent supported by the deployed notification service.

(R) NOTIF-7 An OS shall use all components of the QoS model of the notification service (see sections 2.5 and 3.7 of [19]). While an OMG notification service is expected to understand all standard QoS properties (see CORBA modules `CosNotification` and `CosNotifyCommAck`), it does not need to implement the full range of QoS that these properties are capable of representing. However, the fine-grained, coarse-grained and service-oriented choices of the CORBA-based TMN interface framework *require* the implementation (**R**) of certain QoS properties and *recommend* the optional implementation (**O**) of certain other QoS properties according to Table 1.

Table 1 – Standard QoS properties capabilities

QoS property	Potential values	Implementation	Comment, condition(s)
ConnectionReliability	Persistent	**R**	Refer to 2.5.5.1/[19], or **NOTIF-9**/Q.816, for the semantics of the required and recommended reliability settings. In case of service-oriented interfaces, both settings are recommended as needed. In case of fine-grained and coarse-grained interfaces, **EventReliability=BestEffort** requires additional support of Q.821.1-defined *alarm synchronization* [5], i.e., support of the interfaces `EnhancedCurrentAlarmSummaryControl[_F]`.
	BestEffort		
EventReliability	Persistent	**O**	
	BestEffort		
Priority	`short`		All notifications shall have the default priority `0` to avoid ordering problems. Consumers may use mapping filters to override the default setting as needed.
	LowestPriority		
	DefaultPriority	**R**	
	HighestPriority		
StopTimeSupported	`FALSE`	**O**	It is recommended to not use **absolute expiry times** (on a per-message basis) after which the event can be discarded.
	`TRUE`		
StopTime	`TimeBase::UtcT`		
Timeout	`TimeBase::TimeT`		`TimeT` is an `unsigned long long` with unit 100 nanoseconds. The **relative expiry time** *zero* would indicate that there is no timeout. However, for alarms and TCAs *30 min* is recommended, and for all other notification types *24 h*.
	`0` (digit zero)		
	`18 * 10`^{9} (30 min)	**O**	
	`864 * 10`^{9} (24 h)	**O**	
StartTimeSupported	`FALSE`	**O**	It is recommended to not use **earliest delivery times** (on a per-message basis).
	`TRUE`		
StartTime	`TimeBase::UtcT`		
MaxEventsPerConsumer	`long`		The default setting means that no limits are imposed on the maximum number of events that may be queued for consumers.
	`0` (default)		

Table 1 – Standard QoS properties capabilities

QoS property	Potential values	Implementation	Comment, condition(s)
OrderPolicy	AnyOrder		It shall be possible to deliver events in the order of their arrival. In case of fine-grained or coarse-grained interfaces, events may also be buffered in (consumer-defined) priority order and delivered accordingly, in which case events from the same managed object shall have the same priority, and correlated notifications [8] must not be in use.
	FifoOrder	R	
	PriorityOrder	O	
	DeadlineOrder		
DiscardPolicy	AnyOrder (default)		It shall be possible to discard events, in case of a buffer overflow, in the order of their arrival. It is recommended to ensure that this cannot happen on a per-channel basis, however, by setting the Admin property `RejectNewEvents` to `TRUE` (see **NOTIF-8**).
	FifoOrder	R	
	PriorityOrder		
	DeadlineOrder		
	LifoOrder		
MaximumBatchSize	`long`		The default settings mean that event batches are not supported. If the option to support event batches is required (see **NOTIF-4**), these QoS properties must be set as needed (see, for example, [24][25]).
	`1` (default)	O	
PacingInterval	`TimeBase::TimeT`		
	`0` (default)	O	
DeliveryReliability	None		The OS should consider configuring reliable event delivery using event acknowledgement with sequence numbers, if the notification service supports it (see **NOTIF-6**). See also clause 8.7.5.3/ X.780.2 with regard to event identifiers.
	Acknowledgement	O	
RetryInterval	`TimeBase::TimeT`	O	
Retries	`long`	O	

NOTE 10 – This **NOTIF-7** requirement includes **NOTIF-9**/Q.816 and **NOTIF-10**/Q.816, as corrected by ITU-T Rec. Q.816/Cor.1. It complies with the 3GPP specification [24] and [25] and the MTNM specification [28].

(O) NOTIF-8 While support of the `CosNotification::AdminPropertiesAdmin` interface by the notification service is required for the configuration of channel administration policies, its use by operations systems for the standard Admin properties `MaxQueueLength`, `MaxConsumers`, `MaxSuppliers`, and `RejectNewEvents` is optional, the default values of the properties being implementation-defined. It is recommended to set `RejectNewEvents` to `TRUE` though.

NOTE 11 – This **NOTIF-8** requirement complies with ITU-T Recs Q.816 and Q.816.1, the 3GPP specification [24] and [25], and the MTNM specification [28].

(O) NOTIF-9 An operations system may support an on-demand pull model for synchronization purposes in addition to the push model (which is triggered by the managed system whenever events originate). It is recommended that the pull mechanisms for event batches and typed events are combined and iterators be used to define highly efficient pull operations per event type with filtering capabilities, which are implemented by the managed system without event channels.

(R) NOTIF-10 Operations systems in the role of a supplier or a consumer of structured events shall follow the rules shown in Table 2 below for constructing and receiving structured events. Refer to the following standards for further details:

- Sections 2.1.4, 2.2, 3.1.1 and 3.7 of [19] – *in the general case*;
- **NOTIF-7**/Q.816, the comments of the IDL interface `itut_x780::Notifications` of ITU-T Rec. X.780, and [24], [25] and [26] – *in case of fine-grained and coarse-grained interfaces*;
- Clause 8.7/X.780.2, the comments of the IDL module `notifications` of ITU-T Rec. X.780.2, and [24], [25], [26] and [28] – *in case of service-oriented interfaces*.

NOTE 12 – Sections 2.2, 3.1.1, 3.7 and Appendix B of [19] contain the *normative* specification of the syntax and semantics of OMG structured events.

Table 2 – Frameworks mappings of notifications to structured events

Field name	Actual field type	Potential field values	Comment
Fixed event header			
event_type	`CosNotification::_EventType`	see Comment	The service-oriented framework extends OMG's and Q.816's understanding of *Domains* and *Event types* (Note 1). The values of the **type_name** field depend on the value of the **domain_name** field (Note 2).
domain_name	`string`	`"telecommunications"` [8] or OS-OS interface class (e.g., IRP document version number string [24], [25] and [26], `"tmf_mtnm"` [27] and [28])	
type_name	`string`	see Comment	
event_name	`string`	`""`	This field does not have a standardized semantics. It could uniquely identify the specific event instance being transmitted, or be used for OS-OS interface class-specific purposes (see, for example, [26]), or be used for vendor-specific purposes. For reasons of interoperability, it is recommended to ignore it.
Variable event header (a list of zero or more name/value pairs, initially with `any` values)			
EventReliability	`short`	see Table 1	The OMG defines the standard *optional* header fields listed here for QoS properties configuration at event level (see section 2.2 of [19]). The recommendations for using these QoS properties given in Table 1 apply likewise.
Priority	`short`	see Table 1	
StopTime	`TimeBase::UtcT`	see Table 1	
Timeout	`TimeBase::TimeT`	see Table 1	
StartTime	`TimeBase::UtcT`	see Table 1	
SequenceNumber	`long`	see **NOTIF-6** and section 3.7 of [19]	
ohf_name	*`any`*	*ohf_value*	generic *optional header field*: indicates further optional fields

Table 2 – Frameworks mappings of notifications to structured events

Field name	Actual field type	Potential field values	Comment
Filterable event body (a sequence of name/value pairs, initially with `any` values)			
see Comment	see Comment	see Comment	The potential contents of the filterable event body depends on the **event_type** (Note 3).
fd_name	`any`	*fd_value*	generic *filterable data* field: indicates further optional fields
Remaining event body (additional event data, initially an Any)			
remainder_of_body	`any`, or an overlay structure, or an MOC (see Comment)	`NULL`, or an overlay, or a managed object (see Comment)	Actual type and value of this field depend on the value of **event_type** (Note 4).

NOTE 1 – The OMG specifies `domain_name` as an identifier for the vertical industry domain within which the type of event that characterizes a given structured event is defined. ITU-T Recs Q.816 and X.780 adopt the OMG example `"telecommunications"` for the event types of the fine-grained and coarse-grained framework (see IDL interface `itut_x780::Notifications` [8]). The service-oriented framework takes into consideration that there need not be a unique CORBA specification (set) for a given vertical industry domain if the domain has a broader scope. For example, in the telecommunications industry there are standard CORBA specifications for fixed networks, namely **TMF MTNM**, and mobile networks, namely **the 3GPP IRPs**. So the semantics of `domain_name` is extended to include OS-OS interface classes that are specified in CORBA IDL. For example, `"tmf_mtnm"` identifies TMF MTNM [27] and [28], and IRP document version number strings, or **IRPVersions** for short, identify 3GPP IRPs [24], [25] and [26].

NOTE 2 – The potential values of `type_name` are defined in the CORBA specification that is identified by `domain_name`. In case of `"telecommunications"`, the event type name is the scoped name of the operation defining the event. The interface `Notifications` of X.780 defines 15 such operations, and a constant for each scoped operation name, e.g., `const string attributeValueChangeTypeName = "itut_x780::Notifications::attributeValueChange";`. In case of `"tmf_mtnm"`, the event type name is one of 13 standard string literals documented in [28] and formally defined in the IDL module `notifications` of ITU-T X.780.2, e.g., `const string NT_OBJECT_CREATION = "NT_OBJECT_CREATION";`. In case of an IRPVersion (e.g., `"32.111-3_V6.5"`), the document identified by this IRPVersion will specify an interface `NotificationType` that defines standard string literals for the event type names, e.g., `const string NOTIFY_FM_NEW_ALARM = "x1";`.

NOTE 3 – The mandatory and optional NV pairs of the filterable event body depend on `event_type`. In case of domain_name = `"telecommunications"`, `itut_x780::Notifications` and **NOTIF-7**/Q.816 specify rules for translating each parameter of the event-defining operation into a name/value pair. Optional parameters are either excluded or have a `NULL` value if not supported. These rules comply with OMG's rules for translating typed events into structured events (2.1.4/[19]). In case of domain_name = `"tmf_mtnm"`, [28] specifies for each event type all name/value pairs explicitly together with the actual field types. In case of domain_name = <an IRPVersion> (e.g., = `"32.111-3_V6.5"`), the document identified by this IRPVersion will specify for each event type all name/value pairs explicitly.

NOTE 4 – The actual type and value of the remaining event body depend on `event_type`. In case of domain_name = `"telecommunications"`, **NOTIF-7**/Q.816 states that `remainder_of_body` shall be `NULL`. In case of domain_name = <an IRPVersion>, [24] defines for `remainder_of_body` the overlay structure `NotificationIRPNotifications::NonFilterableEventBody` consisting of a sequence of name/value pairs, which are explicitly considered non-filterable, and the remainder of the remainder, and the document identified by IRPVersion may specify such non-filterable name/value pairs. In case of domain_name = `"tmf_mtnm"`, [28] states that `remainder_of_body` shall be `NULL` except for object creation notifications (OCNs) in which case the actual type depends on the value of the filterable `objectType` field of the OCN and shall be the corresponding IDL `struct` defining the MOC identified by `objectType`. The service-oriented framework recommends this approach (see clause 8.7/X.780.2).

NOTE 13 – This **NOTIF-10** requirement includes **NOTIF-7**/Q.816.

8.4 Telecom log service

The telecom log service [20] enhances the notification service [19] by providing log objects that allow to turn events, which pass filtering constraints, into log records and to store these records persistently. Logs and log managers offer further capabilities according to and beyond ITU-T Rec. X.735 that are briefly summarized in clause 6.6. The fine-grained framework recommends a number of these capabilities for filtered logging of notifications of CORBA managed objects. The coarse-grained framework requires the same usage of the telecom log service for notifications generated by IDL-defined managed objects that are accessible through a CORBA façade.

The service-oriented framework recommends almost the same usage for notifications originating from service-oriented managed objects. It adds some clarifications though and recommends a more lightweight use through the managing system with regard to logs owned by the managed system.

(R) LOG-1 An operations system shall use a telecom log service, which is conformant with the OMG specification (see section 1.5 of [20]). The OMG telecom log service provides a superset of the capabilities of the OMG Notification Service [19] (which in turn is an extension of the OMG event service [18]) and can therefore be used *in lieu* of the notification service. Regarding the notification service capabilities, the telecom log service shall comply with **NOTIF-1**.

NOTE 1 – This **LOG-1** requirement includes **LOG-3**/Q.816 and corresponds to **NOTIF-1**.

NOTE 2 – Interfaces and operations, which are mandatorily or optionally required for use by the managing system, have been collected in clause C.4 of Q.816/Amd.1 as *OMG Telecom Log Service profile*. Further log capabilities that are required for use by the managed system, in particular the log creation and inherited consumer Admin object capabilities of log factories, are not listed in this annex. The service-oriented framework recommends a slightly more restrictive use of log capabilities by the managing system, however, in particular avoiding log modifications and filterings as far as at all possible (see **LOG-7**).

(R) LOG-2 When a telecom log service is used *in lieu* of the notification service, the logs shall be notification channels (i.e., instances of `DsNotifyLogAdmin::NotifyLog`). The support of the other types of logs and of typed log records is optional. **NOTIF-3** applies. A log record can be created from any notification originating from a fine-grained or coarse-grained or service-oriented managed object, and any notification log record can be translated back to a notification.

NOTE 3 – This **LOG-2** requirement includes **LOG-2**/Q.816 and a part of **LOG-1**/Q.816.

(R) LOG-3 A managed system shall instantiate one or more notification logs. It may use any notification log together with an instance of the `emsSession::EmsSession_I` interface (see 9.1.2.2) or an instance of the `NotificationLogIRPSystem::NotificationLogIRP` interface defined by 3GPP [25]. A managing system shall not have access to notification log factories.

NOTE 4 – This **LOG-3** requirement includes another part of **LOG-1**/Q.816 and corresponds to **NOTIF-2**.

(R) LOG-4 When the telecom log service is used *in lieu* of the notification service, it shall comply with **NOTIF-4**, **NOTIF-5**, **NOTIF-7** and **NOTIF-10**, it may comply with **NOTIF-6** and **NOTIF-8**, and the support of **NOTIF-9** is recommended as well.

NOTE 5 – This **LOG-4** requirement includes the remaining part of **LOG-1**/Q.816.

(O) LOG-5 Log records store event data in the attribute `info` of IDL type `any`. Structured and typed events shall be wrapped in and unwrapped from `info` according to the rules specified in section 2.1.4 of [19] for translating the respective message formats. In particular, when a structured event is logged, the type code of `info` shall be set appropriately to indicate the IDL type `struct` with detailed description according to `CosNotification::StructuredEvent`. Log records that store event data should either have no optional attributes, i.e., the attribute `attr_list` should be empty, or the potential content of `attr_list` should be documented in detail.

(O) LOG-6 Log generated events are defined in the CORBA module `DsLogNotification` as structures and generated by log factories as Anys (i.e., untyped/generic events). When generating a log event, log factories shall set the type code of the event value appropriately to indicate the IDL type `struct` with detailed description according to the respective `struct` definition of the event.

(R) LOG-7 Independently of the managed system, the managing system may use log factories of its own and create/instantiate, control and manage its own logs (of any type) and their log records. In case of service-oriented interfaces, when using a notification log owned by the managed system (see **LOG-3**), the managing system shall not destroy the log, not copy the log, not set attributes of the log, not write records to the log, not delete records from the log, not modify any attribute of any log record, and not change or modify the filter object associated with the log.

8.5 Concurrency control and object transaction services

In a distributed computing environment, such as CORBA, updates from one client could possibly be overwritten by undesired updates from a concurrent client unless suitable preventive measures are taken to control the access to shared resources and to guarantee continual data consistency for all concurrent clients. For example, several (authenticated and appropriately authorized) managing systems could be clients of the same managed system with overlapping session periods. The basic mechanism for concurrency control is locking of shared resources. More advanced techniques use the transaction paradigm, which includes lock management. The notification and telecom log services provide a basis for making a client aware that its update has been overwritten but do not (and hardly could) provide a locking mechanism to prevent the occurrence of undesired overwrites.

The fine-grained and coarse-grained frameworks recommend the optional support of the OMG (object) transaction service (OTS) [37] to guarantee data consistency (see clause 6.6/Q.816 and clause 8.6/Q.816.1) but without specifying any OTS feature or any difference in usage for fine-grained and coarse-grained interfaces. They also state that the OTS is designed for high reliability and incurs additional overhead, which may not be required when using a CORBA-based TMN framework. The service-oriented framework recommends a levelled and more lightweight approach to concurrency control and transaction support, if required at a CORBA-based TMN interface.

The OMG has also specified the concurrency control service (CCS) [36], which mediates the access to CORBA objects such that the consistency of the object is not compromised when accessed by concurrent clients. The CCS may be used with or without transactions. It is rather a lock managing facility, the control of concurrent access being provided by the CCS user (e.g., the OTS, any managing system). A CCS-controlled resource, or resource manager, would be a CORBA object that creates and retains a unique lock set on which locks can be acquired, or attempted to acquire, in one of five modes (read, write, upgrade, intention read, intention write). Access to the resource, or the resources managed by the resource manager, is only possible after acquiring a lock and then only to the extent of the lock mode. If a requested lock cannot be granted due to a conflict with already existing locks, the caller will be blocked until the lock can be acquired, or the acquisition attempt will be rejected. The CCS enforces a FIFO order policy for the lock queue.

A service-oriented façade provides a particularly efficient means for concurrency control of its allocated managed objects. Since SO managed objects are only accessible via their assigned SO façade, CCS control of the façade will also provide CCS control of all allocated managed objects.

(O) CONCUR-1 CORBA-based TMN interfaces (fine-grained, or coarse-grained, or service-oriented) should support the OMG concurrency control service [36] in non-transactional mode to guarantee data consistency for managed objects (and management support objects).

(O) CONCUR-2 If data consistency for managed objects (and management support objects) and also certain transaction capabilities are required, CORBA TMN interfaces should use the OMG CCS in transactional mode together with the simplest application programming model of the OMG object transaction service [37] (indirect context management with implicit propagation).

NOTE 1 – The **CONCUR-1** and **CONCUR-2** requirements include **TRANS-1**/Q.816.

For coarse-grained or service-oriented interfaces *transactional capabilities* would be defined through façade operations, i.e., these interfaces have the potentiality to define transactions involving managed objects through transactions of façades. When to which extent, and above all with which major benefit, transactional capabilities should be used at CORBA-based TMN interfaces (fine-grained, or coarse-grained, or service-oriented) is for further study. Such requirements would affect the ORB and POA usage and lead to a refinement of the considerations of clause 6.2.1. Using the OTS and an OTS-aware ORB can be quite complex. For example, the notification service considers transactional event transmission as a QoS property (see section 2.1.5 of [19]) whose enabling requires encoding of the unshared transaction model (see section 2.12 of [37]). The OTS specification [37] offers a rich feature set with two transaction models (flat and nested), four application programming models, and three conformance levels (lite, lite-distributed and full).

(O) TRANS-1 If transaction support is required beyond concurrency control, CORBA TMN interfaces should use an OTS, which provides the required conformance levels and features, with the most lightweight application programming model that meets the requirements.

8.6 CORBAsecurity

CORBA shields applications from the details of networking, but when dealing with CORBAsecurity it is quite useful to have a basic understanding of CORBA's inter-ORB protocols GIOP and IIOP, and of how protocol-specific and CORBA/ORB services-related information are encoded in IORs. These topics go far beyond the two aspects of CORBA treated in clauses 6.2 and 8.1, and constitute the *third aspect of CORBA* that can be important for use by the TMN frameworks:

- **CORBA interoperability features of ORB products**: these are mainly chapter 13 "ORB Interoperability Architecture" (CORBA module `IOP`) and chapter 15 "General Inter-ORB Protocol" (CORBA modules `GIOP` and `IIOP`) of the CORBA specifications.

Assessment of the potential lightweight use of this aspect within the scope of the CORBA-based TMN frameworks is for further study. Related considerations should start with an analysis of the IOR information model regarding tagged profiles and tagged components, and the standard IIOP components. Such analysis shall be guided by the security requirements, services and mechanisms specified in the M.3016-series ITU-T Recommendations, and would need to dive into the details of the `IOP` [44], `GIOP` [45], `IIOP` [45], `SECIOP` [38], `SSLIOP` [38], `CSIIOP` [46] and `SECP` [50] modules in order to map M.3016 items to CORBA interoperability and security capabilities. This clause mainly clarifies and grades the approach of ITU-T Rec. Q.816, specifically with regard to the options for the transport layer that dispatches GIOP messages: TCP (with either IIOP or SECIOP on top), or SSL (with IIOP on top and TCP beneath), or TLS (with TCP below and IIOP above).

CORBAsecurity is provided by capabilities of the OMG security service [38], [39], [41] and [50] and interoperability capabilities of the ORB [42], [44], [45], [46], [47] and [48]. These capabilities encompass (and go beyond) communications security, authentication of principals (human users and objects), (role-based) authorization of access to objects by principals, security auditing, non-repudiation and security administration. Most of these basic capabilities may already be a little overkill for many applications though. Instead, applications might require, for reasons of out-of-the-box availability and simplicity, only the communications security and user/system-level authentication functionality based on transport layer security (TLS) technology [43], or its precursor secure sockets layer (SSL) 3.0 [40], or might require no security at all since they run in an already secured environment. SSL and TLS support by the ORB are optional but any optional support must comply with the CORBA specification. However, only CORBA 2.6 [21] and 3.0 [22] (and also 3.1 of [47] and [48]) are TLS-aware, i.e., support of older ORB versions means to condone SSL 3.0 (see also clause 5.2/Q.816).

Consequently the fine-grained framework recommends three levels of security: no security, use of SSL (with or without certificate-based access control), and use of capabilities of the OMG security service as appropriate (with or without SSL). The coarse-grained framework recommends the same security approach and notes that when using the façade approach, the security service can protect at most at the level of individual façade interface instances. If there are requirements to protect managed objects at the instance level using the capabilities of the OMG security service, then a fine-grained approach should be used. The service-oriented framework recommends a similar usage but emphasizes lightweight features and prioritizes security capabilities accordingly.

The security requirements for CORBA-based TMN interfaces (fine-grained, or coarse-grained, or service-oriented) stated below, therefore, reflect the following choices:

- No security measures at all.
- ORBs use SSL/TLS to provide communications security and user/system-level authentication.

  NOTE 1 – This is session security since SSL/TLS operates between the session/sockets and transport/TCP layers based on cryptographic principles such as public keys, digital signatures, message digests, digital envelopes and signed certificates. A more heavyweight alternative (without user-level authentication, however) would be channel security with IPSec/IKE, which operates between the transport/TCP and network/IP layers based on packet-level cryptography and cryptographic keys per TCP connection.
- Operations systems use the extensible ITU-T session service to protect sessions and façades (with their allocated managed objects) on a per session basis with user name and password authentication and vendor-specific security capabilities (see clauses 9.1.1 and 9.1.2.5).
- The CORBA environment supports specific security measures such as controlled access to CORBA object factories (see, for example, the notification service integrity issue described in clause 7.2.4) or CORBA-aware firewalls, which require at least CORBA 2.4 (see [42] and [14]) or, for the revised firewall traversal specification [49], [45] and [46], even CORBA 3.1 [47] and [48].
- ORBs use SECIOP and the underlying extension SECP of GIOP to provide the security levels and features of CORBA's common secure interoperability (CSI) specification [39], [46] and [50].
- ORBs and operations systems use the OMG security service [38] to provide and consume communications security, authentication, non-repudiation, access control lists for groups or individuals accessing objects and operations, etc., according to the CSI architecture.

The actual level and details of security service to be provided on a CORBA-based TMN interface, and by the deployed distributed environment, is left as a matter to be negotiated between the parties supplying the managed and managing systems, and so all security requirements are optional.

(O) SEC-1 The CORBA environment may optionally support either IIOP/SSL or IIOP/TLS or SECIOP as defined in the OMG security service and ORB interoperability specifications. This requires support of CORBA 2.3.1 [13], or even CORBA 3.0.3 [22] in case of IIOP/TLS.

NOTE 2 – This **SEC-1** requirement includes **SEC-1**/Q.816.

(O) SEC-2 When a secure transport layer for CORBA's GIOP messages according to **SEC-1** is supported, the support of authentication certificates (for clients and servers) and an associated PKI policy shall be an option left up to the administration of the CORBA environment.

NOTE 3 – This **SEC-2** requirement includes **SEC-3**/Q.816.

(O) SEC-3 An operations system may use the ITU-T session service (see clause 9.1) to protect sessions and façades together with their allocated managed objects.

(O) SEC-4 The distributed CORBA environment may support specific security measures such as controlled access to CORBA object factories or CORBA-aware firewalls. Firewall security requires support of CORBA 2.4.2 [14], or even CORBA 3.1 [47] and [48] for improved features.

(O) SEC-5 Managed systems, managing systems, and ORBs may use the OMG security service [38] to support its wide range of capabilities according to CORBA's CSI specification [39], [46] and [50]. This requires support of the appropriate CORBA revision [13], [14], [21], [22], [47] and [48].

NOTE 4 – This **SEC-5** requirement includes **SEC-2**/Q.816.

(O) SEC-6 In case of coarse-grained or service-oriented interfaces, an operations system should take into account that CORBAsecurity can protect only at the level of individual façades, which are CORBA objects providing access to allocated IDL-defined managed objects. If there are requirements to protect managed objects at the individual instance level using CORBAsecurity, then a fine-grained approach should be used where all managed objects have an IOR. If protection is required at the MOC level, then a coarse-grained or service-oriented approach should be used.

9 Framework support services requirements for supporting service-oriented interfaces

In addition to rules for using the ORB and certain OMG common object services in a lightweight fashion, this Recommendation also defines one new support service for use on CORBA TMN interfaces (fine-grained, or coarse-grained, or service-oriented) to manage a client/server connection between a managing system and a managed system. The IDL describing the interfaces to this new ITU-T TMN support service is provided in Annex A. No additional requirements are defined for the ITU-T support services of ITU-T Recs Q.816 and Q.816.1.

9.1 Session service

The session service provides capabilities to establish and maintain a client/server connection between a managing system (client) and a managed system (server), which is called a session. Such a "managing system/managed system session" is incarnated by two session objects, one at the server side (server or EMS session) and one at the client side (client or NMS session). The session service is enabled through server (or EMS) session factories.

(R) SESSION-1 A managed system shall instantiate one or more *Server Session Factory* objects, as defined in clause 9.1.2.5. Each *Server Session Factory* instance shall be registered with the OMG naming service according to the naming rules specified in **SESSION-8**. All access to CORBA objects (e.g., service-oriented façades), except the server session factories themselves, shall be managed through objects of the session service without using the naming service.

The session service can be used with any CORBA TMN framework paradigm (fine-grained, or coarse-grained, or service-oriented). Its use is optional for reasons of compatibility with already deployed framework implementations. It can be added to any basic conformance profile of the framework to provide an advanced conformance profile (see clause 10.2.2). The benefits of the session service are summarized in the next clause.

9.1.1 Session service rationale and potentialities

In a general ORB architecture, a client is an entity that invokes a request on a CORBA object while a server is an entity that implements one or more CORBA objects, and so first of all the terms client and server are meaningful only within the context of a particular request because the entity that is the client for one request may be the server for another request and conversely. Therefore, a general CORBA deployment consists of a set of distributed CORBA objects that are invoked across a CORBA bus. But for reasons of efficiency usually most CORBA objects are grouped into CORBA servers, and CORBA clients implement at most callback objects. CORBA-based TMN interfaces are OS-OS interfaces as per ITU-T Rec. M.3010, where one OS is a managing system that takes a manager or CORBA client role (e.g., an NMS) and the other OS is a managed system that takes an

agent or CORBA server role (e.g., an EMS). The session service provides basic and advanced capabilities to encapsulate and control such a client/server relationship between operations systems.

Refer to clause 6.3 for an overview of the basic capabilities of the session service. The detection of loss of communication through a ping operation is a simple kind of heartbeat service, which can be used from either side of the client/server connection. If the server implements heartbeat notifications (see clause 6.5 and clause 8.7/X.780.2), the *Server Session* interface (see clause 9.1.2.2) could be extended by advanced operations to set and get the period between heartbeats similar to the heartbeat service of the fine-grained and coarse-grained frameworks (see 7.5/Q.816).

The session service interfaces generally allow for vendor-specific extensibility of capabilities. For example, vendors may use the authentication mechanism by user name and password provided by the *Server Session Factory* interface to implement an authentication and access log database, which is maintained by the session service, with the option to customize sessions per user through authorization policies. This lightweight security capability would complement the use of CORBAsecurity (see clause 8.5) and should include communications security (e.g., IIOP over SSL/TLS) or at least a challenge/response scheme to avoid transmitting password information across the network. Another example would be administration of the computing resources allocated to the session including garbage collection. A third example would be implementation of the *observer* design pattern[3] with server sessions as the subjects, or observables, which publish notifications through callback operations of their observers, and client sessions as the observers, which subscribe to receive notifications according to prescribed filter conditions.

The *Server Session Factory* interface instance serves as the entry point to the server OS for all client OSes. The clients gain access to this entry point through the naming service. However, there may be several such entry points, one for each IDL version implemented by the server OS. The clients should therefore use the `getVersion` operation of the server session factory before establishing a session to ensure that this entry point will provide the wanted IDL version.

The `getSupportedManager` and `getManager` operations of the server session interface and the `getCapabilities` operation of the common interface (see 9.3.2.7/X.780.2) provide a capability model that allows to check at run time the availability of service-oriented façades and their operations. They also enable a minimalistic use of the naming service (see clause 6.4.1).

9.1.2 Session service description

Overview descriptions of the session service's capabilities are contained in clauses 6.3, 6.4.1 and 7.2.3. The session service consists of five interfaces two of which are *virtual* (see Figure 7):

- interface *`Session_I`* (generic session);
- interface `EmsSession_I` (server session) that inherits from *`Session_I`*;
- interface `NmsSession_I` (client session) that inherits from *`Session_I`*;
- interface *`Version_I`* (generic IDL versioning);
- interface `EmsSessionFactory_I` (server session factory) that inherits from *`Version_I`*.

---

[3] The observer design pattern [52] is a behavioural pattern with the intent to define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. The pattern relates a subject, or observable, to any number of dependent observers. The subject provides operations for attaching and detaching observers, and an optional operation for notifying all attached observers of state changes. The observer provides an operation for updating, or synchronizing, its state when the state of an attaching subject changes. The update of the observer could be the triggering of the pull of the state of the subject or, when used as a callback operation, could be the push of a notification from the subject according to attachment policies.

These interfaces are described in the following clauses and defined in Annex A. They use interfaces from ITU-T Rec. X.780.2 and OMG's notification service. The IDL of Annex A therefore includes the IDL file of ITU-T Rec. X.780.2 and an OMG IDL file.

(R) SESSION-2 The interfaces supported by the session service objects shall be the interfaces listed above and described below. There are EMS sessions (server sessions), NMS sessions (client sessions), and EMS session factories (server session factories).

9.1.2.1 Session interface

The *Session* interface provides an attribute that contains a reference to the associated party on the other side (i.e., either a *Client Session* object or a *Server Session* object) and operations to allow for the detection of loss of communication and a controlled disconnect between associated parties. The IDL describing the *Session* interface is provided below (without comments).

```
interface Session_I
{
readonly attribute Session_I associatedSession;

void ping();

oneway void endSession();

};
```

The *associatedSession* attribute contains a reference to the `Session_I` object on the other side (NMS/EMS) to which the object is associated. This attribute can be checked to make sure the `NmsSession_I/EmsSession_I` association is still valid, in particular in case of communication failures. The *ping* operation allows for the detection of loss of communication. Differentiation of intermittent problems from loss of connection is implementation-specific for the managed and managing systems. The *endSession* operation allows for a controlled disconnect between associated parties. All resources allocated for this session (at the party) are deleted by the operation.

(R) SESSION-3 The session service shall support the attribute and operations of the *Session* interface as described above and documented in the IDL comments.

9.1.2.2 Server session interface

The *Server Session* interface inherits from the *Session* interface. It defines a type representing a list of names of SO façades, which are called managers, and operations to retrieve the names of the supported managers as well as the IOR of any supported manager. It also defines an operation to retrieve the IOR of the unique OMG event channel (i.e., a notification channel or a telecom log channel) that is associated with the server session. It further defines exceptions for each operation. The IDL describing the *Server Session* interface is provided below (without comments).

```
module emsSession
{
   typedef sequence<string> managerNames_T;

   interface EmsSession_I : session::Session_I
   {
   void getSupportedManagers(
         out managerNames_T supportedManagerList)
        raises(globaldefs::ProcessingFailureException);

   void getManager(
         in string managerName,
         out common::Common_I managerInterface)
        raises(globaldefs::ProcessingFailureException);
```

```
   void getEventChannel(
         out CosNotifyChannelAdmin::EventChannel eventChannel)
        raises(globaldefs::ProcessingFailureException);

   };

}
```

The *managerNames_T* type is used to define parameters that contain a list of manager names, i.e., names of service-oriented façades, as defined by interface specifications that extend this framework and preferably derived from the IDL names of the manager CORBA interfaces that inherit from *Common_I*. The *getSupportedManagers* operation provides the list of manager interfaces that the EMS implements. The *getManager* operation takes a manager name and provides its IOR as *Common_I* interface (see clause 9.3/X.780.2), which the client should narrow to the right façade interface. The *getEventChannel* operation provides the IOR of the unique event channel that the EMS maintains for this server session. The event channel is in fact a notification channel as specified by the OMG notification service, i.e., an event channel as specified by the OMG event service that can have both QoS and administrative properties assigned to it. When the server (EMS) supports the OMG telecom log service, the notification channel is also a log. All operations may raise the unique exception object *ProcessingFailureException* (see clause 8.6/X.780.2) with all admissible exception types being specified as IDL comments.

(R) SESSION-4 The session service shall support the type and operations of the *Server Session* interface as described above and documented in the IDL comments.

9.1.2.3 Client session interface

The *Client Session* interface inherits from the *Session* interface and is instantiated at the client as a callback object. It defines operations that are invoked from the server to inform the client in case of notification losses (events or alarms) and termination of a loss period. The IDL describing the *Client Session* interface is provided below (without comments).

```
   interface NmsSession_I : session::Session_I
   {
   void eventLossOccurred(
         in globaldefs::Time_T startTime,
         in string notificationId);

   void eventLossCleared(
         in globaldefs::Time_T endTime);

   void alarmLossOccurred(
         in globaldefs::Time_T startTime,
         in string notificationId);

   };
```

The *eventLossOccurred* operation should be called by the server (EMS) when an event loss period begins because the server fails to push events to the client (NMS) or discards events for other reasons. It takes as input the time and ID of the first notification lost. The *eventLossCleared* operation shall be called by the server to indicate that an event or alarm loss period is over, which was previously indicated by an *eventLossOccurred* or *alarmLossOccurred* operation. It takes the time of the end of the loss period as input. The *alarmLossOccurred* operation should be called by the server when a loss period begins for a notification type that is not a lifecycle event (but an alarm, a TCA, etc.) because the server discards events. It takes as input the time and ID of the first non-lifecycle event lost. Each of these callback operations must not raise any exception.

(R) SESSION-5 The session service shall support the callback operations of the *Client Session* interface as described above and documented in the IDL comments.

9.1.2.4 Version interface

The *Version* interface defines an operation that allows the client (NMS) to query the current version of the IDL interface implemented by the server (EMS). The format of the return value is normative. The IDL describing the *Version* interface is provided below (without comments).

```
interface Version_I
{
string getVersion();

};
```

The *getVersion* operation returns the version string of the IDL version associated with the *Server Session Factory* instance that implements the *Version* interface. Refer to the IDL comments in Annex A for the normative format of the version string.

(R) SESSION-6 The session service shall support the operation of the *Version* interface as described above and documented in the IDL comments.

9.1.2.5 Server session factory interface

The *Server Session Factory* interface inherits from the *Version* interface. It defines an operation that allows the client to request instantiation of a *Server Session* object and association of this object with a *Client Session* object provided by the client. It also defines exceptions for this operation. The IDL describing the *Server Session Factory* interface is provided below (without comments).

```
interface EmsSessionFactory_I : idlVersion::Version_I
{
void getEmsSession(
      in string user,
      in string password,
      in nmsSession::NmsSession_I client,
      out emsSession::EmsSession_I emsSessionInterface)
     raises(globaldefs::ProcessingFailureException);

};
```

The *getEmsSession* operation takes a user name, a password and the IOR of a previously instantiated *Client Session*, and provides the IOR of a *Server Session*, which is associated to the *Client Session*. The operation may raise the unique exception object *ProcessingFailureException* (see clause 8.6/X.780.2) with all admissible exception types being specified as IDL comments.

(R) SESSION-7 The session service shall support the operation of the *Server Session Factory* interface as described above and documented in the IDL comments.

After instantiation of a *Server Session Factory* object, one object per supported IDL version, the managed system registers the instance(s) with the naming service according to the naming rules specified in **SESSION-8** and **SESSION-9**. In case of service-oriented interface design, it does not register other CORBA objects (façades) with the naming service, and so the server session factories provide the unique, version-specific entry points for clients to interact with the server. This lightweight paradigm is called "minimalistic use of Naming Service". Access to other CORBA interface instances (façade objects) is provided to clients through the *getManager* operation of the returned *Server Session* object (see clause 9.1.2.2).

(R) SESSION-8 A managed system shall instantiate exactly one *Server Session Factory* object per supported IDL version. Also, each Initial Naming Context shall have at least one binding for a *Server Session Factory* object. The values of the *id* strings in this binding shall simply identify the server that implements the *Server Session Factory* object. The *kind* string of the last component in the binding shall identify the class of the object ("emsSessionFactory::EmsSessionFactory_I").

(R) SESSION-9 When the CORBA naming graph is a lightweight naming tree (see **NAME-2**), the name components of a *Server Session Factory* name below the initial naming context shall have the following kinds: "Class", "Vendor", "EmsInstance", "Version", "EmsSessionFactory_I".

NOTE 1 – The naming rules for *Server Session Factory* objects are more general than the naming rules for ITU-T TMN support services specified in ITU-T Rec. Q.816 (see FACTORY_FINDER-1, CHANNEL_FINDER-1, TERM-1, MOO-1, HEARTBEAT-1) and ITU-T Rec. Q.816.1 (see CONTAINMENT-1) where all support service objects are located directly under a local root naming context (e.g., the initial naming context).

NOTE 2 – The versioning concept of having one *Server Session Factory* object per supported IDL version does not prevent the use of the versioning guidelines clause 6.13/X.780 and clause 10.5.4/X.780.2.

9.2 Other ITU-T support services

In the lightweight approach to network management with service-oriented CORBA TMN interfaces, the use of the support services of ITU-T Recs Q.816 and Q.816.1 is optional for the reasons stated in clause 7.2.5. If used, no additional requirements need to be placed on the use of these support services for supporting service-oriented interfaces.

10 Service-oriented compliance and conformance

This clause defines the criteria that shall be met by other standards documents claiming compliance to the service-oriented CORBA framework and the functions of CORBA-based TMN services that shall be implemented by operations systems claiming conformance to this Recommendation. Since conformance to ITU-T Rec. Q.816.2 includes a more flexible conformance to ITU-T Recs Q.816 and Q.816.1, this clause also summarizes the conformance points of ITU-T Recs Q.816 and Q.816.1 in a convenient manner.

10.1 Standards document compliance

Any specification claiming compliance with the service-oriented CORBA framework shall:

1) Provide a description of how its constituent documents, or document sections, and further deliverables (such as CORBA IDL files) can be split between the two aspects "information modelling in IDL" and "ORB and CORBA services usage" of CORBA-based TMN interface specification.

2) Support all the standards document compliance requirements related to the "information modelling in IDL" aspect as stated in ITU-T Rec. X.780.2.

3) Meet the following **standards document compliance criteria** related to the "ORB and CORBA services usage" aspect:

   - specify a usage of the ORB, the telecom-related OMG common object services, and the ITU-T TMN support services that comply with ITU-T Rec. Q.816.1 (coarse-grained interfaces) or with this Recommendation (service-oriented interfaces) (see clause 10.2.1);
   - in case of coarse-grained interfaces, specify, along the lines of clauses 6.7 and 7.3, how the coarse-grained interface design can be redesigned gradually to become more and more service-oriented;
   - in case of service-oriented interfaces, specify how the relevant documents, or document sections, and further deliverables (such as CORBA IDL files) are related to the clauses of this Recommendation;

- if the specification includes IDL files that collectively define a session service according to clause 9.1, specify how the normative IDL of Annex A, including comments, can be obtained from the provided IDL files *without any syntactical changes*,

  except for *pragmas*, *module names*, and, if need be, a few minor *additional IDL constructs* that do not at all affect the constructs of Annex A,

  and *with only moderate and reasonable modifications with regard to* (ordinary or formatted) *comments* (for example, inclusion of references to additional parts of the specification that are out of scope of the service-oriented CORBA framework).

4) Follow the service-oriented interface design rules defined in clause 10/X.780.2.

10.2 System conformance

This clause first summarizes the conformance points of this Recommendation, ITU-T Recs Q.816.1 and Q.816 (i.e., the requirements on usage of the ORB, telecom-related OMG common object services, and ITU-T TMN support services) from the viewpoint of service-oriented TMN interface design, and then combines them in several conformance profiles that shall be supported by operations systems claiming conformance to ITU-T Recs Q.816, or Q.816.1, or this Recommendation.

10.2.1 Conformance points

The individual functions of CORBA-based TMN services described earlier in this Recommendation, or in ITU-T Recs Q.816 and Q.816.1, are summarized conveniently as conformance points. These conformance points are combined in the next clause in conformance profiles for the three CORBA framework paradigms (fine-grained, coarse-grained, and service-oriented).

1) An operations system claiming conformance to the *Basic ORB* requirements (including *Minimum CORBA*) shall:
   - in case of fine-grained and coarse-grained interfaces,

     support the version of CORBA (i.e., the ORB) specified in clause 5.2/Q.816;
   - in case of service-oriented interfaces,

     support the mandatory ORB requirements specified in clause 8.1.

2) An operations system claiming (stepwise) conformance to the *Advanced ORB* requirements (including *CORBA Messaging*) shall:
   - be Basic ORB conformant;
   - in case of fine-grained and coarse-grained interfaces,

     support the (optional) CORBA messaging requirements specified in clause 6.4/Q.816;
   - in case of service-oriented interfaces,

     support step-by-step the optional ORB requirements specified in clause 8.1.

3) An operations system claiming conformance to the *Naming Service* requirements shall:
   - support the OMG naming service with the version specified in clause 5.2/Q.816, and evolve towards the version specified by [17];
   - in case of fine-grained and coarse-grained interfaces,

     support the naming service requirements specified in clause 6.1/Q.816, and the naming service capabilities identified in detail in clause C.1/Q.816, **except** for the fact that in case of coarse-grained interfaces managed object names are not required to be bound to managed object IORs in the naming service;

- in case of coarse-grained interfaces,

  support the naming service requirements specified in clause 8.1/Q.816.1;

- in case of service-oriented interfaces,

  support the naming service requirements specified in clause 8.2 (which include the naming service capabilities identified in clause C.1/Q.816).

NOTE 1 – This conformance point includes the conformance point 1) of clause 8.1.1/Q.816.

4) An operations system claiming conformance to the *Notification Service* requirements shall:

- support either:
  - the OMG notification service with the version specified in clause 5.2/Q.816, and evolve towards the version specified by [19];
  - the 3GPP `NotificationIRP` interface specified in [24] together with an OMG notification service as specified by [19];
  - the 3GPP `NotificationIRP` interface specified in [24] without using a notification service but (partly) implementing some OMG notification service interfaces as specified in [19];
  - in exceptional and provisional cases an implementation of the observer design pattern (see clauses 9.1.1 and 6.5) that complies with **ORB-4** (see clause 8.1);
- if the OMG notification service is supported, support the mandatory notification service requirements specified in clause 8.3, and the notification service capabilities identified in detail in clauses C.2 and C.3/Q.816.

NOTE 2 – This conformance point includes and corrects the conformance point 2) of clause 8.1.1/Q.816.

5) An OS claiming conformance to the *Telecom Log Service* requirements shall:

- support either:
  - the OMG telecom log service with the version specified in clause 5.2/Q.816, and evolve towards the version specified by [20];
  - the 3GPP `NotificationLogIRP` interface specified in [25] together with an OMG notification service as specified by [19];
  - the 3GPP `NotificationLogIRP` interface specified in [25] without using a Notification Service but (partly) implementing some OMG Notification Service interfaces as specified in [25] and [19];
- if the OMG telecom log service is supported, support the mandatory telecom log service requirements specified in clause 8.4, and the telecom log service capabilities identified in detail in clause C.4/Q.816, **except** for conflicts with **LOG-7** in case of service-oriented interfaces.

NOTE 3 – This conformance point includes the conformance point 3) of clause 8.1.1/Q.816.

6) An operations system claiming conformance to the *Concurrency Control and Object Transaction Services* requirements shall:

- support the OMG concurrency control service (CCS) version specified by [36];
- optionally support the OMG object transaction service (OTS) version specified in clause 5.2/Q.816, and evolve towards the version specified by [37];
- support the concurrency control and object transaction services requirements specified in clause 8.5 as appropriate.

7) An operations system claiming conformance to the *CORBAsecurity* requirements shall:

- if the OMG security service is used, support the version specified in clause 6.5/Q.816, and evolve towards the version specified by [38];
- support the CORBAsecurity requirements specified in clause 8.6 as appropriate;
- support the required version of CORBA (i.e., the ORB).

8) An OS claiming conformance to the *Factory Finder Service* requirements shall:

- support the factory finder service interface described in clause 7.1/Q.816 and defined in the CORBA IDL in Annex A/Q.816.

9) An OS claiming conformance to the *Channel Finder Service* requirements shall:

- support the channel finder service interface described in clause 7.2/Q.816 and defined in the CORBA IDL in Annex A/Q.816.

10) An operations system claiming conformance to the *Terminator Service* requirements shall:

- support the terminator service interface described in clause 7.3/Q.816 and defined by the CORBA IDL in Annex A/Q.816;
- in case of coarse-grained interfaces,

  support the terminator service requirements specified in clause 9.3/Q.816.1.

11) An operations system claiming conformance to the *Basic MOO Service* requirements shall:

- support the mandatory MOO service requirements described in 7.4.3/Q.816;
- in case of coarse-grained interfaces,

  support the MOO service requirements specified in clause 9.4/Q.816.1.

12) An OS claiming conformance to the *Advanced MOO Service* requirements shall:

- be basic MOO service conformant;
- support the optional MOO service requirements described in clause 7.4.3/Q.816.

13) An operations system claiming conformance to the *Heartbeat Service* requirements shall:

- support the heartbeat service interface described in clause 7.5/Q.816 and defined in the CORBA IDL in Annex A/Q.816.

14) An OS claiming conformance to the *Containment Service* requirements shall:

- support the mandatory containment service requirements described in clause 9.6/Q.816.1 and the containment service interface defined in the CORBA IDL in Annex A/Q.816.1 including synchronization with the naming service where required.

15) An operations system claiming conformance to the *Session Service* requirements shall:

- support the session service requirements described in clause 9.1 and the session service interfaces defined in the CORBA IDL in Annex A taking clause 10.1 into consideration.

10.2.2 Conformance profiles

This clause combines the conformance points of the previous clause in conformance profiles for ITU-T Rec. Q.816 (see clause 8.1.2/Q.816), ITU-T Rec. Q.816.1 (see clause 10.1.2/Q.816.1) and this Recommendation.

An OS claiming conformance to the *Q.816 Core Profile* (fine-grained) shall support:

1) the basic ORB requirements (see conformance point 1);
2) the naming service requirements (see conformance point 3);
3) the notification service requirements (see conformance point 4).

An OS claiming conformance to the *Q.816 Basic Profile* (fine-grained) shall support:

1) the Q.816 core profile;
2) the factory finder service requirements (see conformance point 8);
3) the channel finder service requirements (see conformance point 9);
4) the terminator service requirements (see conformance point 10);
5) the basic MOO Service requirements (see conformance point 11).

An OS claiming conformance to the *Q.816.1 Core Profile* (coarse-grained) shall support:

1) the Q.816 core profile (coarse-grained);
2) the containment service requirements (see conformance point 14).

An OS claiming conformance to the *Q.816.1 Basic Profile* (coarse-grained) shall support:

1) the Q.816 basic profile (coarse-grained);
2) the containment service requirements (see conformance point 14).

An OS claiming conformance to the *Q.816.2 Core Profile* (service-oriented) shall support:

1) the basic ORB requirements (see conformance point 1);
2) the naming Service requirements (see conformance point 3);
3) the notification service requirements (see conformance point 4).

An OS claiming conformance to the *Q.816.2 Basic Profile* (service-oriented) shall support:

1) the Q.816.2 core profile;
2) the session service requirements (see conformance point 15).

The definition of advanced conformance profiles with regard to the support of OMG ORB capabilities, OMG common object services, and ITU-T TMN support services (including the definition of new support services, if required) is for further study. For example, fine-grained and coarse-grained interfaces could also support the session service requirements and become gradually more service-oriented along the lines outlined in clauses 6.7 and 7.3, and all three choices for TMN interface design could support the optional notification service requirements (see clause 8.3) or some of the telecom log service requirements (see conformance point 5).

10.3 Conformance statement guidelines

The users of this framework must be careful when writing conformance statements. Because IDL modules are being used as name spaces, they may, as allowed by OMG IDL rules, be split across files. Thus, when an IDL module is extended its name will not change. Instead, a new IDL file will simply be added. Simply stating the name of an IDL module in a conformance statement, therefore, will not suffice to identify a set of IDL interfaces. The conformance statement shall identify a document and month of publication to make sure the right version of IDL is identified.

A standards document claiming compliance to the service-oriented framework may specify a lightweight IDL file structure where modules are not allowed to be split into multiple files (and therefore updates of IDL modules always result in updates of IDL files).

```
/**
```

Annex A

Service-oriented framework support services IDL

(This annex forms an integral part of this Recommendation)

```
*/

/* This IDL code is intended to be stored in a file named "itut_q816_2.idl"
located in the search path used by IDL compilers on your system. Most comments
are formatted to be parsed by an IDL-to-HTML converter. */

#ifndef ITUT_Q816_2_IDL
#define ITUT_Q816_2_IDL


// ********************************
// *                              *
// * itut_q816_2.idl              *
// *                              *
// ********************************


/*

This file defines an extensible, service-oriented NML-EML interface
in CORBA IDL, more generally an OS-OS interface according to Rec. M.3010,
where one OS takes a client/manager role (e.g., an NMS) and the other OS
takes a server/agent role (e.g., an EMS). The OS in a client role is called
managing system, and the OS in a server role is called managed system.
The IDL uses the modules "globaldefs" and "common" of the service-oriented
modelling IDL of Recommendation X.780.2 and defines a Session Service
according to Recommendation Q.816.2.

The IDL is organised into the following modules, interfaces,
operations, exceptions, attributes, and data types.

module             | interface                  | operation
-------------------+----------------------------+-----------------------------
idlVersion         | Version_I                  | getVersion()
session            | Session_I                  | ping()
                   |                            | endSession()
nmsSession         | NmsSession_I               | eventLossOccurred()
                   |                            | eventLossCleared()
                   |                            | alarmLossOccurred()
emsSession         | EmsSession_I               | getSupportedManagers()
                   |                            | getManager()
                   |                            | getEventChannel()
emsSessionFactory  | EmsSessionFactory_I        | getEmsSession()

module             | exception, attribute       | data type
-------------------+----------------------------+-----------------------------
idlVersion         | -                          | -
session            | associatedSession          | -
nmsSession         | -                          | -
emsSession         | -                          | managerNames_T
emsSessionFactory  | -                          | -

*/

// Include list
#include "itut_x780_2.idl"
#include "OMGidl/CosNotifyChannelAdmin.idl"

#pragma prefix "itu.int"

/**
```

A.1 Module `idlVersion`

```
*/

  /**
   * <p>This module contains the definition of the Version interface
   * of the NML-EML interface.</p>
   **/

module idlVersion
{
  /**
   * <p>The interface Version_I allows the NMS to query the current
   * version of the IDL interface (IDL version) implemented by the EMS.
   * In order to use this CORBA interface, the NMS needs to invoke the
   * getVersion() service to figure out which version of the NML-EML interface
   * the EMS is providing. getVersion() should be called by any NMS
   * before other communications with an EMS. The NMS can determine
   * from the response string which IDL version of the EMS is available.</p>
   *
   * <p>For details on how to support multiple versions of the IDL see
   * clause 9.1.2.5/Q.816.2 "Server Session Factory interface" and clause
   * 10.5.4/X.780.2 "Versioning of CORBA IDL specifications".</p>
   **/
   interface Version_I
   {
  /**
   * <p>This service returns the version of the IDL interface (IDL version)
   * that the corresponding EMS supports.</p>
   *
   * <p>The format of the return string is as follows:<br>
   * <i>Release</i>.<i>Major</i>[.<i>Minor</i>],
   * where <i>Release</i>, <i>Major</i> and <i>Minor</i>
   * are strings that contain only digits.</p>
   *
   * <p>For example, "2.1" indicates release 2 and major release 1,
   * "1.3" indicates release 1 and major release 3, and so on.
   * Note that "x.y" has the same meaning as "x.y.0".
   * The minor digit is used for bug fixing of the major release
   * (e.g., "1.2.1" is a minor release on "1.2").</p>
   *
   * @returns string: The IDL version of the NML-EML interface.
   **/
   string getVersion();

   };  // end of interface

};  // end of module


/**
```

A.2 Module `session`

```
*/

  /**
   * <p>This module contains the definition of the Session interface
   * of the NML-EML interface.</p>
   **/

module session
{

  /**
   * <p>The Session_I interface provides capabilities to manage a
   * client/server connection, which is called a session. Its main
   * purpose is to enable either a client or a server to detect the
   * loss of communication with the associated party.</p>
   *
```

```
   * <p>For a single communication session between an NMS and an EMS, there
   * are two Session_I objects. One is maintained on the NMS; the other one
   * is maintained on the EMS. The Session_I object maintained on the EMS is
   * actually an EmsSession_I while the Session_I object maintained on the
   * NMS is actually an NmsSession_I (both inherit from Session_I).</p>
   *
   * <p>Each Session_I object is responsible to "ping" the other Session_I
   * object periodically to detect communication failures. Exactly when this
   * is done is up to the implementation.</p>
   *
   * <p>When a Session_I object detects a communication failure, or when
   * the endSession() operation is called on it, all resources allocated
   * with that communication session must be freed and the Session_I object
   * must be deleted.</p>
   **/

   interface Session_I
   {
  /**
   * <p>This attribute contains a reference to the Session_I object on the
   * other side (NMS/EMS) to which the object is associated. It is readonly.
   * The attribute can (and should) be checked to make sure the
   * NmsSession_I/EmsSession_I association is still valid
   * (in particular in case of communication failures).</p>
   **/
   readonly attribute Session_I associatedSession;

  /**
   * <p>Allows for the detection of loss of communication.
   * It is implementation-specific to differentiate intermittent
   * problems from loss of connection.</p>
   **/
   void ping();

  /**
   * <p>Allows for a controlled disconnect between associated parties.
   * All resources allocated for this session are deleted by the
   * operation. Best-effort semantics are expected of invocations of
   * this operation (which does not guarantee delivery of the call);
   * the default semantics are exactly-once if the operation successfully
   * returns or at-most-once if a (standard) exception is returned.</p>
   **/
   oneway void endSession();

   };  // end of interface

};  // end of module


/**
```

A.3 Module `nmsSession`

```
*/

  /**
   * <p>This module contains the definition of the NmsSession interface
   * of the NML-EML interface.</p>
   *
   * <p>The nmsSession module provides a means for the server to inform the
   * NMS in case of notification losses and termination of a loss period.</p>
   **/

module nmsSession
{
  /**
   * <p>This interface is instantiated at the NMS. The NMS passes a handle
   * to an instance of this interface in the client parameter of the
   * getEmsSession() operation of EmsSessionFactory_I.</p>
   */
```

```
 interface NmsSession_I : session::Session_I
 {
/**
 * <p>When an EMS fails to push an event, it can notify all connected
 * NMSes by invoking this method on every active NmsSession_I.
 * This method should also be invoked on any new NmsSession_I set up
 * during the event loss period.</p>
 *
 * <p>Once the EMS invokes this method on the NmsSession_Is, it sets
 * an internal flag to indicate that it has already informed NMSes of
 * event loss. As long as this flag is set, the EMS will not invoke
 * this method again. It however may invoke alarmLossOccurred()
 * if it discards a non-lifecycle event.</p>
 *
 * <p>When this method is invoked on an NmsSession_I, the NMS comes to
 * know that the EMS has failed to push one or more events that may be
 * of interest to it. The NMS should consider itself to be potentially
 * out-of-sync with the EMS. It should wait until the EMS calls
 * eventLossCleared() before resynchronizing with the EMS.</p>
 *
 * @parm globaldefs::Time_T <b>startTime</b>: The time of detection of
 * the first notification loss.
 *
 * @parm string <b>notificationId</b>: The notificationId of the first
 * notification lost.
 **/
 void eventLossOccurred(
       in globaldefs::Time_T startTime,
       in string notificationId);

/**
 * <p>The EMS invokes this method to indicate that the event (or
 * alarm etc.) loss period is over, and that it is now capable of
 * providing all relevant notifications.</p>
 *
 * <p>After invoking this method on the NmsSession_Is, the EMS clears
 * the internal flag set by alarmLossOccured() or/and eventLossOccurred().
 * If an event or alarm etc. loss occurs again, alarmLossOccurred() or
 * eventLossOccurred() will be called again.</p>
 *
 * <p>How and when the EMS decides to invoke eventLossCleared() is
 * an EMS implementation detail.</p>
 *
 * @parm globaldefs::Time_T <b>endTime</b>: The time of the end of
 * the event loss period, as determined by the EMS.
 **/
 void eventLossCleared(
       in globaldefs::Time_T endTime);

/**
 * <p>When an EMS discards an alarm, a TCA, a file transfer status,
 * or another non-lifecycle event (with regard to the lifecycle of
 * the event source), it can notify all connected NMSes by
 * invoking this method on every active NmsSession_I. This service
 * should also be invoked on any new NmsSession_I set up during
 * the event loss period.</p>
 *
 * <p>Once the EMS invokes this method on the NmsSession_Is, it sets
 * an internal flag to indicate that it has already informed NMSes
 * of alarm etc. loss. As long as this flag is set, the EMS will not
 * invoke this method again. It however may invoke eventLossOccurred()
 * if it fails to push a different type of event.</p>
 *
 * <p>When this method is invoked on an NmsSession_I, the NMS comes to
 * know that the EMS has discarded one or more alarms, TCAs, file
 * transfer statuses, or notifications of other types that may be
 * of interest to it. The NMS should consider itself to be potentially
 * out-of-sync with the EMS with respect to these notification types.
 * It should wait until the EMS calls eventLossCleared() before
 * resynchronizing with the EMS on alarms, TCAs, file transfer statuses,
 * and the other non-lifecycle notification types.</p>
```

```
   *
   * @parm globaldefs::Time_T <b>startTime</b>: The time of the first
   * notification discard.
   *
   * @parm string <b>notificationId</b>: The notificationId of the first
   * notification discarded.
   **/
   void alarmLossOccurred(
         in globaldefs::Time_T startTime,
         in string notificationId);

   };  // end of interface

};  // end of module


/**
```

A.4 Module `emsSession`

```
*/

  /**
   * <p>This module contains the definition of the EmsSession interface
   * of the NML-EML interface.</p>
   *
   * <p>The emsSession module provides a means for the client to
   * interrogate the EMS to determine which manager CORBA interfaces
   * (i.e., service-oriented façades) it actually supports. The NMS can
   * then retrieve an instance of each of the manager interfaces it
   * requires. This capability is achieved using generic IDL so that
   * new manager interfaces can be easily added.</p>
   *
   * <p>The module also provides access to the unique OMG event channel
   * to be used within the session.</p>
   **/

module emsSession
{
  /**
   * <p>Sequence of manager names, i.e., a list of names of service-oriented
   * façades, as defined by NML-EML interface specifications that extend
   * this framework and preferably derived from the IDL names of the manager
   * CORBA interfaces that inherit from common::Common_I.</p>
   **/
   typedef sequence<string> managerNames_T;

  /**
   * <p>A handle to an instance of this interface is gained via the
   * emsSessionInterface parameter of the getEmsSession() operation
   * of EmsSessionFactory_I.</p>
   */
   interface EmsSession_I : session::Session_I
   {
  /**
   * <p>This allows an NMS to request the manager CORBA interfaces that
   * the EMS implements.</p>
   *
   * @parm managerNames_T <b>supportedManagerList</b>: The list of manager names
   * supported by the EMS (see managerNames_T type).
   *
   * @raises globaldefs::ProcessingFailureException<dir>
   * EXCPT_INTERNAL_ERROR - Raised in case of non-specific EMS internal
   *  failure<br>
   * EXCPT_ACCESS_DENIED - Raised in case of security violation<br>
   * </dir>
   **/
   void getSupportedManagers(
         out managerNames_T supportedManagerList)
        raises(globaldefs::ProcessingFailureException);
```

```
  /**
   * <p>This operation allows an NMS to gain access to an instance of
   * the specified manager CORBA interface (i.e., service-oriented façade)
   * without using the OMG Naming Service.</p>
   *
   * @parm string <b>managerName</b>: The class or type name of the manager
   * object that the client wants to access (see getSupportedManagers()
   * operation).
   *
   * @parm common::Common_I <b>managerInterface</b>: A CORBA IOR for the manager
   * object. The actual object returned will implement the specified manager
   * interface. However it is returned as a Common_I object (see module
   * common) so that this operation can be generic.  The client should
   * narrow the returned object reference to the correct object type.
   *
   * @raises globaldefs::ProcessingFailureException<dir>
   * EXCPT_NOT_IMPLEMENTED - Raised if the EMS does not support the manager<br>
   * EXCPT_INTERNAL_ERROR - Raised in case of non-specific EMS internal
   *  failure<br>
   * EXCPT_ACCESS_DENIED  - Raised in case of security violation<br>
   * </dir>
   **/
   void getManager(
         in string managerName,
         out common::Common_I managerInterface)
        raises(globaldefs::ProcessingFailureException);

  /**
   * <p>This operation allows an NMS to gain access to the OMG event
   * channel to receive notifications. It returns a reference to a
   * NotifyChannel interface instance (which is an EventChannel) as
   * defined by the OMG Notification Service. When the EMS supports
   * the OMG Telecom Log Service, this operation will return a
   * reference to a NotifyLog interface instance (which is a
   * NotifyChannel and an EventLog).</p>
   *
   * @parm CosNotifyChannelAdmin::EventChannel <b>eventChannel</b>:
   * The event channel (NotifyChannel or NotifyLog) to be used
   * by the NMS in this session.
   *
   * @raises globaldefs::ProcessingFailureException<dir>
   * EXCPT_INTERNAL_ERROR - Raised in case of non-specific EMS internal
   *  failure<br>
   * EXCPT_ACCESS_DENIED - Raised in case of security violation<br>
   * </dir>
   **/
   void getEventChannel(
         out CosNotifyChannelAdmin::EventChannel eventChannel)
        raises(globaldefs::ProcessingFailureException);

   };  // end of interface

};  // end of module


/**
```

A.5 Module `emsSessionFactory`

```
*/

  /**
   * <p>This module contains the definition of the EmsSessionFactory
   * interface of the NML-EML interface.</p>
   **/

module emsSessionFactory
{
  /**
   * <p>There is a single instance of the EmsSessionFactory_I (per
   * supported IDL version). It is the entry point to the server/EMS.
```

```
    * This instance provides the object reference that the client/NMS
    * uses to connect to the server (if the IDL version fits).</p>
    *
    * <p>This CORBA interface implements the Version interface and will
    * return the server's IDL version (i.e., the version of the NML-EML
    * interface) when getVersion() is called on it.</p>
    **/

    interface EmsSessionFactory_I : idlVersion::Version_I
    {
   /**
    * <p>This operation allows the client/NMS to obtain the EmsSession_I
    * object from which all managers of the server/EMS can be obtained
    * (i.e., all service-oriented façade objects actually implemented
    * by the server/EMS).</p>
    *
    * @parm string <b>user</b>: The (registered) user or application that is
    * trying to access the EMS. The user name can be the empty string
    * to indicate that no authentication mechanism is implemented by the
    * EMS. The format is defined by the interface specifications
    * that extend this framework.
    *
    * @parm string <b>password</b>: The password/passphrase of the user; it
    * can be the empty string. The format and possible encryption are
    * defined by the interface specifications that extend this framework.
    *
    * @parm nmsSession::NmsSession_I <b>client</b>: A handle to the NmsSession_I
    * object, instantiated at the NMS, to which the returned EmsSession_I
    * object will be associated.
    *
    * @parm emsSession::EmsSession_I <b>emsSessionInterface</b>:
    * A CORBA IOR for the EmsSession_I interface object instantiated
    * for the (authenticated) user.
    *
    * @raises globaldefs::ProcessingFailureException<dir>
    * EXCPT_INTERNAL_ERROR - Raised in case of non-specific EMS internal
    *  failure<br>
    * EXCPT_INVALID_INPUT - Raised when client is invalid (see errorReason
    *  for details provided by the EMS if applicable)<br>
    * EXCPT_ACCESS_DENIED - Raised in case of security violation
    *  (e.g., when user or password is invalid) (see errorReason
    *  for details provided by the EMS if applicable)<br>
    * </dir>
    **/
    void getEmsSession(
          in string user,
          in string password,
          in nmsSession::NmsSession_I client,
          out emsSession::EmsSession_I emsSessionInterface)
         raises(globaldefs::ProcessingFailureException);

    };  // end of interface

};  // end of module


#endif  // end of #ifndef ITUT_Q816_2_IDL
```

Bibliography

The following references contain information that was used in the development of the framework for CORBA-based service-oriented TMN interfaces.

[30] OMG TC Documents orbos/97-05-15 and orbos/97-05-16, *ORB Portability Joint Submission (Final)*.

[31] OMG TC Document orbos/98-01-18, *Objects By Value Joint Revised Submission (with errata)*. See also OMG TC Document ptc/98-07-06, *OBV 2.3 RTF Report*.

[32] OMG TC Document orbos/98-08-04, *Minimum CORBA*. See also: OMG Document formal/01-02-59, *Chapter 23* of [14]; OMG Document formal/02-08-01, *Minimum CORBA Specification*.

[33] OMG TC Document orbos/98-05-05, *CORBA Messaging*. See also: OMG Document formal/01-02-58, *Chapter 22* of [14], February 2001; OMG Document formal/04-03-20, *Chapter 22* of [21], March 2004.

[34] OMG Document formal/98-12-09, *CORBAservices: Common Object Services Specification*. See also OMG Document formal/97-02-24, November 1996.

[35] OMG Document formal/05-02-02, *Lightweight Log Service Specification*, Version 1.1.

[36] OMG Document formal/03-09-02, *Concurrency Control Service (CCS) Specification*, Version 1.0, April 2000. See also *Chapter 7* of [34], March 1995.

[37] OMG Document formal/03-09-02, *Transaction Service Specification*, Version 1.4. See also *Chapter 10* of [34], November 1997.

[38] OMG Document formal/02-03-11, *Security Service Specification*, Version 1.8. See also: OMG TC Document 1994/94-07-01, *Object Services RFP3: Security and Time Services*, July 1994; OMG TC Document 1995/95-12-01, *CORBA Security Joint Revised Submission*, December 1995; [39]; [40]; OMG Document formal/98-12-17, *Chapter 15* of [34], December 1998.

[39] OMG TC Document orbos/96-06-20, *Common Secure Interoperability Joint Submission*, July 1996. See also OMG TC Document orb/96-01-03, *Common Secure IIOP RFP*, February 1996.

[40] IETF TLS Working Group Internet-Draft, *The SSL Protocol Version 3.0*, November 1996, draft-freier-ssl-version3-02.txt (Alan O. Freier, Philip Karlton, Paul C. Kocher). See also http://home.netscape.com/eng/ssl3/index.html.

[41] OMG TC Document orbos/97-02-04, *Secure Sockets Layer/CORBA Security Joint Revised Submission*, February 1997. See also OMG TC Document orbos/96-08-02, *Secure Sockets Layer/CORBA Security RFP*, August 1996.

[42] OMG TC Document orbos/98-05-04, *CORBA/Firewall Security Joint Revised Submission*, May 1998. See also: OMG TC Document cf/97-06-07, *CORBA/Firewall Security RFP*, June 1997; OMG TC Document orbos/98-07-03, *CORBA/Firewall Security Joint Revised Submission + Errata*, July 1998.

[43] IETF RFC 4346 (2006), *The Transport Layer Security (TLS) Protocol Version 1.1*. See also IETF RFC 2246 (1999), *The TLS Protocol Version 1.0*.

[44] OMG TC Document formal/02-06-17, *ORB Interoperability Architecture*, Revision 3.0, changebar version, July 2002. See also: *Chapter 13* of [21], May 2002; *Chapter 13* of [22], March 2004; previous changebar versions: *2.6* (formal/01-12-17, December 2001), *2.5* (formal/01-09-50, September 2001), *2.4.2* (formal/01-02-17, February 2001), *2.4.1* (formal/00-11-05, November 2000), *2.4* (formal/00-10-17, October 2000), *2.3* (formal/99-07-17, June 1999).

[45] OMG TC Document formal/02-06-19, *General Inter-ORB Protocol*, Revision 3.0, changebar version, July 2002. See also: OMG TC Document ptc/04-04-06, *Firewall Traversal FTF – revised GIOP chapter*, April 2004; *Chapter 15* of [21], May 2002; *Chapter 15* of [22], March 2004; previous changebar versions: *2.6* (formal/01-12-19, December 2001), *2.5* (formal/01-09-52, September 2001), *2.4.2* (formal/01-02-19, February 2001), *2.4* (formal/00-10-19, October 2000), *2.3.1* (formal/99-10-11, October 1999).

[46] OMG TC Document formal/01-12-64, *Secure Interoperability*, December 2001. See also: OMG TC Document orbos/99-01-10, *Common Secure Interoperability Version 2 RFP*, January 1999; OMG TC Document orbos/00-08-04, *CSIv2 Joint Revised Submission*, July 2000; *Chapter 26* of [21], May 2002; *Chapter 24* of [22], March 2004; OMG TC Document ptc/04-04-09, *Firewall Traversal FTF – revised Secure Interoperability chapter*, April 2004.

[47] OMG TC Document pas/04-08-01, ISO/IEC 19500-1:200x, *CORBA Specification, Version 3.1, Part 1: CORBA Interfaces*, Draft 1, August 2004.

[48] OMG TC Document pas/04-08-02, ISO/IEC 19500-2:200x, *CORBA Specification, Version 3.1, Part 2: CORBA Interoperability*, Draft 1, August 2004.

[49] OMG TC Document ptc/04-04-05, *CORBA Firewall Traversal Convenience Document*, April 2004. See also: OMG TC Document orbos/00-09-20, *CORBA Firewall Traversal RFP*, September 2000; OMG TC Document orbos/01-10-10, *CORBA Firewall Traversal Joint Revised Submission with Errata Applied*, October 2001.

[50] OMG TC Document ptc/05-02-02, *SECurity Protocol (SECP) Specification*, February 2005. See also OMG TC Document security/02-09-02, *SECP RFP*, September 2002.

[51] OMG TC Document ab/98-06-03, *OMG IDL Style Guide*, June 1998.

[52] GAMMA (Erich), HELM (Richard), JOHNSON (Ralph), VLISSIDES (John): Design Patterns – Elements of Reusable Object-Oriented Software, *Addison-Wesley*, August 1994, ISBN 0-201-63361-2.

[53] Siemens AG, Using CORBA for Multi-Technology Network Management (MTNM), *TM Forum MTNM Contribution*, May 2003, http://www.tmforum.org/browse.asp?catID=2014.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	**Switching and signalling**
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2008