

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

X.780.2

(03/2007)

SERIES X: DATA NETWORKS, OPEN SYSTEM
COMMUNICATIONS AND SECURITY

OSI management – Management functions and ODMA
functions

TMN guidelines for defining service-oriented CORBA managed objects and façade objects

ITU-T Recommendation X.780.2

ITU-T X-SERIES RECOMMENDATIONS
DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY

PUBLIC DATA NETWORKS	
Services and facilities	X.1–X.19
Interfaces	X.20–X.49
Transmission, signalling and switching	X.50–X.89
Network aspects	X.90–X.149
Maintenance	X.150–X.179
Administrative arrangements	X.180–X.199
OPEN SYSTEMS INTERCONNECTION	
Model and notation	X.200–X.209
Service definitions	X.210–X.219
Connection-mode protocol specifications	X.220–X.229
Connectionless-mode protocol specifications	X.230–X.239
PICS proformas	X.240–X.259
Protocol Identification	X.260–X.269
Security Protocols	X.270–X.279
Layer Managed Objects	X.280–X.289
Conformance testing	X.290–X.299
INTERWORKING BETWEEN NETWORKS	
General	X.300–X.349
Satellite data transmission systems	X.350–X.369
IP-based networks	X.370–X.379
MESSAGE HANDLING SYSTEMS	X.400–X.499
DIRECTORY	X.500–X.599
OSI NETWORKING AND SYSTEM ASPECTS	
Networking	X.600–X.629
Efficiency	X.630–X.639
Quality of service	X.640–X.649
Naming, Addressing and Registration	X.650–X.679
Abstract Syntax Notation One (ASN.1)	X.680–X.699
OSI MANAGEMENT	
Systems Management framework and architecture	X.700–X.709
Management Communication Service and Protocol	X.710–X.719
Structure of Management Information	X.720–X.729
Management functions and ODMA functions	X.730–X.799
SECURITY	X.800–X.849
OSI APPLICATIONS	
Commitment, Concurrency and Recovery	X.850–X.859
Transaction processing	X.860–X.879
Remote operations	X.880–X.889
Generic applications of ASN.1	X.890–X.899
OPEN DISTRIBUTED PROCESSING	X.900–X.999
TELECOMMUNICATION SECURITY	X.1000–

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

ITU-T Recommendation X.780.2

TMN guidelines for defining service-oriented CORBA managed objects and façade objects

Summary

This Recommendation defines a set of TMN CORBA managed object and façade object modelling guidelines required to support service-oriented interfaces. It specifies how service-oriented CORBA TMN interfaces are to be defined. It covers IDL repertoire, information modelling in IDL and IDL style conventions. It also provides guidelines on redesigning fine-grained and coarse-grained interfaces to service-oriented interfaces. An IDL module is provided defining basic data types, exceptions, notification constructs, a base CORBA struct to be included by every service-oriented managed object and a base CORBA interface to be inherited by every service-oriented CORBA interface. This Recommendation and ITU-T Recommendation Q.816.2 together compose a framework for CORBA-based service-oriented TMN interfaces with a wide range of applications.

Source

ITU-T Recommendation X.780.2 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), lightweight object, 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/>.

© ITU 2008

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.

CONTENTS

Page

1	Scope	1
1.1	Purpose	2
1.2	Application	2
2	References.....	2
3	Definitions	4
3.1	Terms defined elsewhere	4
4	Abbreviations and acronyms	5
5	Conventions	8
6	Service-oriented interface design considerations	8
6.1	Flexible use of façade design pattern	9
6.2	Lightweight use of ORB.....	10
6.3	Naming of managed objects and service-oriented façade objects.....	11
6.4	Creation and deletion of objects	11
6.5	Inheritance and containment.....	12
6.6	Exceptions	12
6.7	Retrieval and modification of objects and attributes	12
6.8	Notifications	12
6.9	Conditional capabilities and supported capabilities	12
6.10	Additional service-oriented interface design considerations.....	13
7	Service-oriented framework and requirements overview.....	13
7.1	Framework overview.....	13
7.2	Framework constituents overview.....	13
8	Service-oriented object model IDL	16
8.1	IDL repertoire and foundation IDL	17
8.2	Use of name/value pairs with string values	22
8.3	Single-valued, multi-valued and tagged parameters	22
8.4	Modelling of transmission technologies.....	24
8.5	Common attributes <code>Common_T</code> of service-oriented managed objects	25
8.6	Exceptions	27
8.7	Notifications	28
9	Providing service-oriented façade interfaces.....	49
9.1	Façade instantiation.....	49
9.2	Getting and setting objects and attributes.....	50
9.3	Service-oriented façade interface base class <code>Common_I</code>	52
9.4	Iterator interfaces.....	55



Documents

ITU-T Rec. X.780.2 (03/2007)

Document Sharing Hub

iii

	Page
10 Service-oriented CORBA modelling guidelines	59
10.1 Rules and conventions	61
10.2 Superclasses of service-oriented managed objects and façade objects	62
10.3 Naming conventions for managed objects and façade objects	62
10.4 Service-oriented modelling of transmission technologies	69
10.5 Modelling of IDL extensions	80
10.6 Additional CORBA modelling guidelines and principles	88
11 Style guide for lightweight CORBA IDL modelling	90
12 Guidelines for redesigning fine-grained and coarse-grained models to service-oriented	90
13 Service-oriented compliance and conformance	90
13.1 Standards document compliance	91
13.2 System conformance	91
13.3 Conformance statement guidelines	97
Annex A (Normative) – Service-oriented modelling IDL	98
A.1 Module <code>globaldefs</code>	100
A.2 Module <code>common</code>	105
A.3 Module <code>transmissionParameters</code>	110
A.4 Module <code>notifications</code>	112
Bibliography	132

ITU-T Recommendation X.780.2

TMN guidelines for defining service-oriented CORBA managed objects and façade objects

1 Scope

The TMN architecture defined in [ITU-T M.3010] introduces concepts from distributed processing and includes the use of multiple management protocols. [ITU-T Q.816] and [ITU-T 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. [ITU-T Q.816] and [ITU-T X.780] use an approach where each manageable resource is addressable with a unique interoperable object reference (IOR) and have become known as "fine-grained" interfaces.

[ITU-T Q.816.1] and [ITU-T X.780.1] extend the CORBA-based TMN interface framework so that not every manageable resource needs to have an IOR. The extension has become known as "coarse-grained" interfaces. 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. All managed objects that are accessible through a particular coarse-grained façade are required to be of the same class and, for each managed object class, (at least) one façade interface shall be provided. Managed objects accessed through a façade still have a CORBA name even though they may not (but nevertheless could) have an individual CORBA interface. To support the ability to determine a managed object's façade based on only its name, the CORBA names of managed objects accessible through a coarse-grained façade are extended slightly beyond the names of managed objects not accessible through a façade.

This Recommendation, along with [ITU-T Q.816.2], further extends the framework to enable it to support a different style of interaction between managing systems and managed systems. This style of interaction is termed "service-oriented". The "service-oriented" style introduces the so-called "service-oriented façade objects" in the managed systems and relieves a managing system from having to retrieve the location information for each manageable resource or type of manageable resource it wishes to access. This Recommendation provides the IDL-related service-oriented modelling guidelines, including the superclasses of the service-oriented managed objects and façade objects, rules and conventions, naming of managed objects and service-oriented façade objects, IDL repertoire and style guide for use by IDL modellers, modelling of transmission technologies according to [ITU-T Q.805] and [ITU-T G.809], and standardized modelling of IDL extensions. The companion Recommendation [ITU-T Q.816.2] specifies the lightweight use of built-in CORBA system services of ORB products and of certain OMG common object services, and introduces a new ITU-T TMN support service, the session service, which allows for the implementation of a very high degree of lightweight object.

The scope of this Recommendation is the same as the original TMN CORBA framework in the Q.816-series and X.780-series of ITU-T Recommendations. The framework and these extensions cover all interfaces in the TMN where CORBA may be used. These interfaces are OS-OS interfaces according to [ITU-T 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 modelling 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 Recommendation [ITU-T Q.816.2] 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 Q.816.2] accompanies this Recommendation and extends the framework ORB usage, the OMG common services usage and the ITU-T TMN framework support services defined in [ITU-T Q.816] and [ITU-T Q.816.1]. Collectively, [ITU-T Q.816.2] and this Recommendation define a *framework* for CORBA-based service-oriented TMN interfaces. The fine-grained, coarse-grained and service-oriented choices of the CORBA-based TMN interface framework are used in Amendment 1 to [ITU-T M.3010] to provide conformance definitions for TMN interfaces between operations systems. Refer to [ITU-T Q.816.2] for details.

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.

[ITU-T G.805]	ITU-T Recommendation G.805 (2000), <i>Generic functional architecture of transport networks</i> .
[ITU-T G.809]	ITU-T Recommendation G.809 (2003), <i>Functional architecture of connectionless layer networks</i> .
[ITU-T G.852.2]	ITU-T Recommendation G.852.2 (1999), <i>Enterprise viewpoint description of transport network resource model</i> .
[ITU-T G.8010]	ITU-T Recommendation G.8010/Y.1306 (2004), <i>Architecture of Ethernet layer networks</i> .
[ITU-T M.3010]	ITU-T Recommendation M.3010 (2000), <i>Principles for a telecommunications management network</i> .
[ITU-T M.3017]	ITU-T Recommendation M.3017 (2003), <i>Framework for the integrated management of hybrid circuit/packet networks</i> .
[ITU-T M.3100]	ITU-T Recommendation M.3100 (2005), <i>Generic network information model</i> .
[ITU-T Q.816]	ITU-T Recommendation Q.816 (2001), <i>CORBA-based TMN services</i> .
[ITU-T Q.816.1]	ITU-T Recommendation Q.816.1 (2001), <i>CORBA-based TMN services: Extensions to support coarse-grained interfaces</i> .
[ITU-T Q.816.2]	ITU-T Recommendation Q.816.2 (2007), <i>CORBA-based TMN services: Extensions to support service-oriented interfaces</i> .

[ITU-T Q.821.1]	ITU-T Recommendation Q.821.1 (2001), <i>CORBA-based TMN alarm surveillance service</i> .
[ITU-T Q.822]	ITU-T Recommendation Q.822 (1994), <i>Stage 1, stage 2 and stage 3 description for the Q3 interface – Performance management</i> .
[ITU-T X.200]	ITU-T Recommendation X.200 (1994) ISO/IEC 7498-1:1994, <i>Information technology – Open Systems Interconnection – Basic Reference Model: The basic model</i> .
[ITU-T X.721]	ITU-T Recommendation X.721 (1992) ISO/IEC 10165-2:1992, <i>Information technology – Open Systems Interconnection – Structure of management information: Definition of management information</i> .
[ITU-T X.733]	ITU-T Recommendation X.733 (1992) ISO/IEC 10164-4:1992, <i>Information technology – Open Systems Interconnection – Systems Management: Alarm reporting function</i> .
[ITU-T X.780]	ITU-T Recommendation X.780 (2001), <i>TMN guidelines for defining CORBA managed objects</i> .
[ITU-T X.780.1]	ITU-T Recommendation X.780.1 (2001), <i>TMN guidelines for defining coarse-grained CORBA managed object interfaces</i> .
[ITU-T Y.1311]	ITU-T Recommendation Y.1311 (2002), <i>Network-based VPNs – Generic architecture and service requirements</i> .
[ITU-T Y.1314]	ITU-T Recommendation Y.1314 (2005), <i>Virtual private network functional decomposition</i> .
[ETSI TS 132 150]	ETSI TS 132 150 V6.5.0 (2006), <i>Digital cellular telecommunications system (Phase 2+); UMTS; Telecommunication management; Integration Reference Point (IRP) Concept and definitions</i> .
[OASIS]	OASIS Standard soa-rm (2006), <i>Reference Model for Service Oriented Architecture 1.0</i> , OASIS Standard.
[OMG 98-07-01]	OMG Document formal/98-07-01, <i>The Common Object Request Broker: Architecture and Specification</i> , Revision 2.2.
[OMG 99-10-07]	OMG Document formal/99-10-07, <i>The Common Object Request Broker: Architecture and Specification</i> , Revision 2.3.1.
[OMG 00-06-22]	OMG Document formal/00-06-22, <i>Property Service</i> , Version 1.0.
[OMG 01-02-33]	OMG Document formal/01-02-33, <i>The Common Object Request Broker: Architecture and Specification</i> , Revision 2.4.2.
[OMG 04-10-02]	OMG Document formal/04-10-02, <i>Event Service Specification</i> , Version 1.2.
[OMG 04-10-03]	OMG Document formal/04-10-03, <i>Naming Service</i> , Version 1.3.
[OMG 04-10-13]	OMG Document formal/04-10-13, <i>Notification Service Specification</i> , Version 1.1.
[TMF versioning]	TM Forum TMF814 Version 3.0 (2004), <i>Multi-Technology Network Management (MTNM) NML-EML Interface: CORBA</i>

	<i>IDL Solution Set, Supporting Document "Programmatic Versioning", file "versioning.pdf".</i>
[TMF objectNaming]	TM Forum TMF814 Version 3.0 (2004), <i>Multi-Technology Network Management (MTNM) NML-EML Interface: CORBA IDL Solution Set, Supporting Document "Multi-Technology Network Management support for a Naming Convention", file "objectNaming.pdf".</i>
[TMF servicesUsages]	TM Forum TMF814 Version 3.0 (2004), <i>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".</i>
[TMF LayerRates]	TM Forum TMF814 Version 3.0 (2004), <i>Multi-Technology Network Management (MTNM) NML-EML Interface: CORBA IDL Solution Set, Supporting Document "Layer Rates", file "LayerRates.pdf".</i>
[TMF probableCause]	TM Forum TMF814 Version 3.0 (2004), <i>Multi-Technology Network Management (MTNM) NML-EML Interface: CORBA IDL Solution Set, Supporting Document "Specification of probableCause strings", file "ProbableCauses.pdf".</i>
[TMF PerformanceParameters]	TM Forum TMF814 Version 3.0 (2004), <i>Multi-Technology Network Management (MTNM) NML-EML Interface: CORBA IDL Solution Set, Supporting Document "Performance Parameters", file "PerformanceParameters.pdf".</i>

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses terms defined in other Recommendations. Notes, if any, following the terms describe how the terms are being used in the context of this Recommendation.

3.1.1 agent: See ITU-T Rec. X.701.

3.1.2 event channel: See [ITU-T Q.816] and [OMG 04-10-02].

3.1.3 façade: See [ITU-T X.780.1].

3.1.4 interface: See [ITU-T M.3010].

3.1.5 managed object: See ITU-T Rec. X.700.

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.

3.1.6 managed object class: See ITU-T Rec. X.701.

3.1.7 manager: See ITU-T Rec. X.701.

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

3.1.8 notification: See ITU-T Rec. X.703.

NOTE – ITU-T Rec. X.703 "Information technology – Open Distributed Management Architecture" provides the ODMA definition, or ODP-RM definition, of notification. Amendment 1 to ITU-T Rec. X.703 "Support using Common Object Request Broker Architecture (CORBA)" relates this definition to the OMG event definition [OMG 04-10-02] (generic or typed event message), and hence to the OMG notification definition [OMG 04-10-13] (generic or typed or structured event message).

3.1.9 notification channel: See [OMG 04-10-13].

3.1.10 operations system: See [ITU-T M.3010].

3.1.11 service-oriented approach/architecture (SOA): See [ITU-T Q.816.2] in harmony with [OASIS].

3.1.12 service-oriented façade: See [ITU-T Q.816.2] in harmony with [OASIS].

3.1.13 service-oriented façade interface: See [ITU-T Q.816.2] in harmony with [OASIS].

3.1.14 service-oriented façade object: See [ITU-T Q.816.2] in harmony with [OASIS].

3.1.15 service-oriented managed object: See [ITU-T Q.816.2] in harmony with [OASIS].

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

3GPP	3rd Generation Partnership Project
AI	Adapted Information
AID	Alarm Identifier
AIS	Alarm Indication Signal
ALM	Alarm notification
AP	Access Point
ARC	Alarm Reporting Control
ASA	Alarm Severity Assignment
ASAP	Alarm Severity Assignment Profile
ASN.1	Abstract Syntax Notation One (see X.680-series ITU-T Recommendations)
AVC	Attribute Value Change notification
BSE	Backup Status Event notification
cFD	component Flow Domain
CI	Characteristic Information
CL	ConnectionLess
cLink	component Link
CO	Connection-Oriented
CORBA	Common Object Request Broker Architecture
COS	Current Operational Status
CP	Connection Point
CS	Circuit-Switched

cSN	component SubNetwork
CTP	Connection Termination Point
CTP	Connectivity Termination Point
DIOA	Distributed Interface-Oriented Architecture
DMI	Definition of Management Information
EML	Element Management Layer
EMS	Element Management System
FD	Flow Domain
FDF	Flow Domain Flow
FDFr	Flow Domain Fragment
FDN	Full Distinguished Name
fLink	fibre Link
FM	Fault Management
FP	Flow Point
FPP	Flow Point Pool
FT	File Transfer
FTS	File Transfer Status notification
GDMO	Guidelines for the Definition of Managed Objects (ITU-T Recs X.721 and X.722)
HBE	HeartBeat Event notification
HCPN	Hybrid Circuit/Packet Network
ID	Identifier
IDL	Interface Definition Language
IOR	Interoperable Object Reference
IRP	Integration Reference Point
LC	Link Connection
LF	Link Flow
MFD	Matrix Flow Domain
MO	Managed Object
MOC	Managed Object Class
MTN	Multi-Technology Network
MTNM	Multi-Technology Network Management
MTTM	Multi-Technology Telecommunications Management
NC	Network Connection
NE	Network Element
NF	Network Flow
NML	Network Management Layer
NMS	Network Management System

NSA	Non-Service Affecting
NV pair	Name/Value pair with name being a string and with a "name=value" semantics
NVL pair	Name/Value pair with value being a list of strings
NVS pair	Name/Value pair with value being a string
OCN	Object Creation Notification
ODN	Object Deletion Notification
OMG	Object Management Group
ORB	Object Request Broker
OS	Operations System
PM	Performance Management
PMP	Performance Monitoring Point
PS	Packet-Switched
PTP	Physical Termination Point
RCAI	Root Cause Alarm Indication
RDN	Relative Distinguished Name
RP	Reference Point
Rx	Receive side of unidirectional transmission (sink)
SA	Service Affecting
SCN	State Change Notification
SN	Subnetwork
SNC	Subnetwork Connection
SNFr	Subnetwork Fragment
SO	Service-Oriented
SOA	Service-Oriented Approach/Architecture
SOF	Service-Oriented Façade
TCA	Threshold Crossing Alert notification
TCAPP	Threshold Crossing Alert Parameter Profile
TCP	Termination Connection Point
TFP	Termination Flow Point
TMN	Telecommunications Management Network
TP	Termination Point
TS	Technical Specification
TSN	Technology-Specific Network
TTP	Trail Termination Point
TWM	Threshold Watermark
Tx	Transmit side of unidirectional transmission (source)
UFATN	Unified Functional Approach/Architecture for Transport Networks



UML	Unified Modelling Language
UMTS	Universal Mobile Telecommunication System
UTC	Coordinated Universal Time
VC	Virtual Connection/Circuit
VPN	Virtual Private Network

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 data types, base classes and other service-oriented modelling constructs of the framework (e.g., iterators), is included in Annex A. The IDL is written in a 9-point courier typeface:

```
// Example IDL
void setAdditionalInfo(
    in globaldefs::NamingAttributes_T objectName,
    inout globaldefs::NVSLList_T additionalInfo)
    raises (globaldefs::ProcessingFailureException);
```

Annex A of this Recommendation 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 of [ITU-T X.780.1] 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_x780_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".

6 Service-oriented interface design considerations

This clause, a companion to clause 6 of [ITU-T Q.816.2], identifies several design considerations that should be addressed by the framework as support for lightweight use of CORBA through service-oriented interfaces is added. While clause 6 of [ITU-T Q.816.2] identifies design considerations for lightweight use of CORBA services, this clause identifies design considerations for lightweight use of IDL repertoire and derived lightweight IDL modelling of service-oriented managed objects and façade objects. Clauses 6.4 to 6.10 are for further study but provide, in the current version of this Recommendation, a brief summary of their intended contents.

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 a "fine-grained" approach. A general "coarse-grained" approach uses the façade design pattern¹ 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 is organized at a separate upper level which results in coarser "grains" being accessed and controlled.

[ITU-T Q.816.1] and [ITU-T 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 Q.816.2], defines a lightweight generic use of the façade design pattern which, together with other lightweight design considerations, generalizes 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 (M.3020-series of ITU-T Recommendations). 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 of [ITU-T X.780]). Refer to clause 12 below and clause 7.3 of [ITU-T Q.816.2] for a concise description of the relationships between the three approaches. Additional information on CORBA IDL modelling is available from [b-Siemens AG].

The service-oriented design considerations related to CORBA IDL repertoire and modelling concern superclasses, naming of managed objects and service-oriented façades, lightweight definition of exceptions, operations and notifications, modelling of multiple telecom transmission technologies according to [ITU-T G.805] and [ITU-T G.809], and standardized modelling of extensions.

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. A coarse-grained interface is created by first defining a fine-grained interface according to [ITU-T X.780]. Once a fine-grained interface is defined, façade interfaces for each of the managed object interfaces are developed according to [ITU-T X.780.1]. Refer to clause 6.1 of [ITU-T Q.816.2] for further details.

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 (preferably exclusively) through service-oriented façades². 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 clauses 7.2.1.1 and 10.3) leaving detailed relationships to the

¹ The façade design pattern [b-GAMMA] 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 lets one vary the components and interfaces of the subsystem without affecting its clients.

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.

discretion of the managed system, which may also support dynamic binding of managing systems to one or more service-oriented façades (see clause 7.2.1.2 of [ITU-T Q.816.2]).

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 X.780.1] and [ITU-T X.780]. The service-oriented approach mainly promotes co-existence of fine-grained, coarse-grained and (information model-dependent) service-oriented interfaces, however, and initially does not aim at transition of one TMN interface kind to another.

6.2 Lightweight use of ORB

The OMG CORBA 2 specifications [OMG 98-07-01] [OMG 99-10-07] [OMG 01-02-33] consist of several parts for several aspects of CORBA that shall or should be offered by compliant ORB vendors. Most important for use by the TMN frameworks are the following two aspects of CORBA:

- IDL modelling repertoire of ORB products: These are mainly chapters 3 "OMG IDL Syntax and Semantics" and 5 "Value Type Semantics" of the CORBA 2 specifications.
- Built-in basic CORBA system services of ORB products: These are mainly chapters 4 "ORB Interface" and 11 "The Portable Object Adapter" of the CORBA 2 specifications.

The lightweight use of the first aspect, the IDL repertoire and modelling, is specified in this Recommendation, whilst the lightweight use of the second aspect, the basic CORBA system services, is specified in the companion Recommendation [ITU-T Q.816.2] together with OMG services usage.

6.2.1 IDL repertoire and modelling

The CORBA IDL-related constituents of the service-oriented framework are described in this Recommendation, including the minimum IDL repertoire that should be used, based on the service-oriented object model IDL provided in Annex A (see clauses 8 and 9) and the service-oriented CORBA modelling guidelines for using and extending the object model IDL (see clause 10). As a result, a number of requirements on service-oriented IDL repertoire usage and derived modelling of interfaces, operations, exceptions, attributes, data types and notifications are defined.

A particularly important part is the modelling of managed object classes and service-oriented façade interfaces, which inherit from a set of common attributes `Common_T` (a struct that is often virtual) or from a root service-oriented façade interface called `Common_I` (which is often virtual). The `Common_I` façade interface specifies among other things the setter/mutator functions for the settable common attributes and some getter/accessor functions (see clauses 9.2 and 9.3.2).

6.2.2 Attributes, use of value types, and managed object class specification

A CORBA interface can have constants and attributes which are inherited. Inherited constants can be redefined and attributes are mapped to programming language-specific system accessor/getter and mutator/setter functions with system exceptions; an accessor to retrieve (i.e., get) the value of the attribute and a mutator to modify (i.e., set) the value of the attribute if it is not read-only. Better control of attribute access can be achieved by modelling attributes as explicit IDL operations that are used to read and write the attribute value and have well-defined exceptions (i.e., IDL-specific user accessor and mutator functions). The names of the operations should then indicate the attribute name as well as the type of operation (get or set) and the attribute type should be used as return value and input parameter type. For example, an operation name convention could be "<attribute name>Get" and "<attribute name>Set". Set-valued (i.e., multi-valued) attributes would be modelled as CORBA sequences and should have additional operations to add and remove values to and from

the value set. The fine-grained and coarse-grained frameworks adopt this approach for attribute modelling. Within the service-oriented framework, such accessor and mutator functions are optional and more lightweight modelling rules are recommended.

The collection of all attributes of an interface defines its state³. All three frameworks group the state of natively fine-grained managed object interfaces 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:

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

The `struct` and `valuetype` options can be considered lightweight. The `sequence` option is generic but very heavyweight since it requires the specification of all `name` fields at the interface and the use of type code techniques to properly treat the contents of the `value` fields (or else very detailed implementation agreements between the parties using the interface).

A managed system claiming conformance to the service-oriented framework may choose one of these alternatives to specify MOCs but, for the purpose of transitioning to a harmonized way of MOC definition, with the goal to have only a single alternative for all MOCs in the long term.

6.3 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 (see clause 6.2.2) 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 (see clauses 8.5 and 10.3). 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 7.2.1.1 and 10.3 for more details.

6.4 Creation and deletion of objects

This clause first summarizes creation and deletion of fine-grained and coarse-grained managed objects and coarse-grained façades including OCNs and ODNs. Service-oriented façades cannot be created or deleted by a managing system but are created by the managed system without OCNs during start up and restart. The available façades can be determined by a managing system at any time using the session service or the naming service. Service-oriented managed objects can be created or deleted by the managed system or the managing system or both, and OCNs or ODNs are

³ In the context of state definition, interface constants are considered as initialized read-only attributes. Interface attributes are something akin to public class members though they do not define storage.

generated or not – the actual choices depend on the individual MOC. Creation and deletion of managed objects will modify the set of containment relationships and hence the naming graph (see clauses 6.5 and 10.3.2). The service-oriented framework does not require definition of managed object factories (per MOC). Instead, for each MOC that can be created and deleted by managing systems, the responsible SO façade has to expose create and delete operations.

The completion of this clause is for further study.

6.5 Inheritance and containment

This clause first summarizes the inheritance and containment principles of fine-grained and coarse-grained interfaces. In case of service-oriented interfaces, more lightweight principles are followed. Inheritance is determined by the root classes `Common_T` and `Common_I` (see clauses 8.5 and 9.3) and containment by the service-oriented naming conventions (see clause 10.3.2). The service-oriented framework initially does not require definition of IDL name binding submodules per clause 6.8 of [ITU-T X.780] (with examples from ITU-T Rec. M.3120) to identify the allowable containment relationships but proposes consideration of simplified such modules when progressing the framework.

The completion of this clause is for further study.

6.6 Exceptions

This clause sets out the principles of exceptions as a lead into clause 8.6: the invocation of an operation may result in raising a previously defined exception as listed in the `raises` expression of the operation declaration, or a standard exception defined for the supported ORB version (and not listed under `raises`). An overview is provided of the exceptions defined and used by the fine-grained, coarse-grained and service-oriented frameworks and their different flavours.

The completion of this clause is for further study.

6.7 Retrieval and modification of objects and attributes

This clause leads into clause 9 by identifying, based on clause 6.2.2, the generalizable natures of retrieval and modification (e.g., dealing with individual attributes and attribute selections, dealing with attributes that are lists of parameters such as `additionalInfo` and `transmissionParams`, use of iterators, scoping with regard to contained objects, use of a filtering constraint language, unsolicited modifications through state changes) including AVCs and SCNs. It summarizes the principles of clauses 9.2 and 8.7.5.4 and refers to clause 10.1 for more general guidelines.

The completion of this clause is for further study.

6.8 Notifications

This clause leads into clause 8.7 by introducing the principles of notifications arising from CORBA managed objects. It describes the relationship between façades and notifications, provides a concise and consolidated overview of the events of the fine-grained, coarse-grained and service-oriented frameworks including ways for harmonisation/co-existence based on clause 8.3 of [ITU-T Q.816.2], and summarizes service-oriented fault management (with ASAPs and event acknowledgement) and relevant parts of service-oriented performance management (including PMPs and TCAPPs).

The completion of this clause is for further study.

6.9 Conditional capabilities and supported capabilities

The fine-grained and coarse-grained frameworks support, in the context of GDMO translation, a generic concept of conditional support for groups of capabilities called conditional packages and a corresponding `packagesGet()` operation. The service-oriented framework offers a more lightweight

concept of a two-level capability model with a corresponding getCapabilities() operation to identify the supported operations of a given service-oriented façade (and other capabilities), and with a session service operation for on-demand determination of the supported façades themselves.

The completion of this clause is for further study.

6.10 Additional service-oriented interface design considerations

This clause serves for alignment with the fine-grained and coarse-grained design considerations (see clauses 4 of [ITU-T X.780], 6 of [ITU-T X.780.1], 4 of [ITU-T Q.816] and 6 of [ITU-T Q.816.1]) and oversees further key service-oriented interface design considerations contained in [ITU-T X.780.2] (and [ITU-T Q.816.2]) such as universal parameter list attributes with single-valued (NVS pair), multi-valued (NVL pair) or tagged parameters.

The completion of this clause is for further study.

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 defines guidelines for developing or adopting/endorsing information models for service-oriented interfaces including lightweight generic façades to access and control managed objects, while [ITU-T Q.816.2] focuses on the framework support services for service-oriented interfaces. First, an overview of the service-oriented framework is presented, then an overview of its constituents.

7.1 Framework overview

The service-oriented framework consists of two superclasses, three lightweight OMG services, one lightweight ITU-T support service and a small number of conventions (standard data types, notification specifications, etc.). Refer to clause 7.1 of the companion Recommendation [ITU-T Q.816.2] for details. The superclasses and conventions are defined in this Recommendation. Since the service-oriented CORBA framework is capable of supporting any protocol-neutral information model, the potential range of application-specific objects (managed objects and façade objects) that are supported by the framework through inheritance from the superclasses is very large.

7.2 Framework constituents overview

The CORBA services-related constituents of the framework (ORB, OMG services, ITU-T services) are described in the companion Recommendation [ITU-T Q.816.2] and its CORBA IDL-related constituents are described in this Recommendation (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 G.805] and [ITU-T G.809], standardized modelling of extensions, etc.).

7.2.1 Façade design pattern and service-oriented façade objects

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 its own behaviour (e.g., present its capabilities). All façade objects are instantiated by the managed system during startup, 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 1 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 1 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.

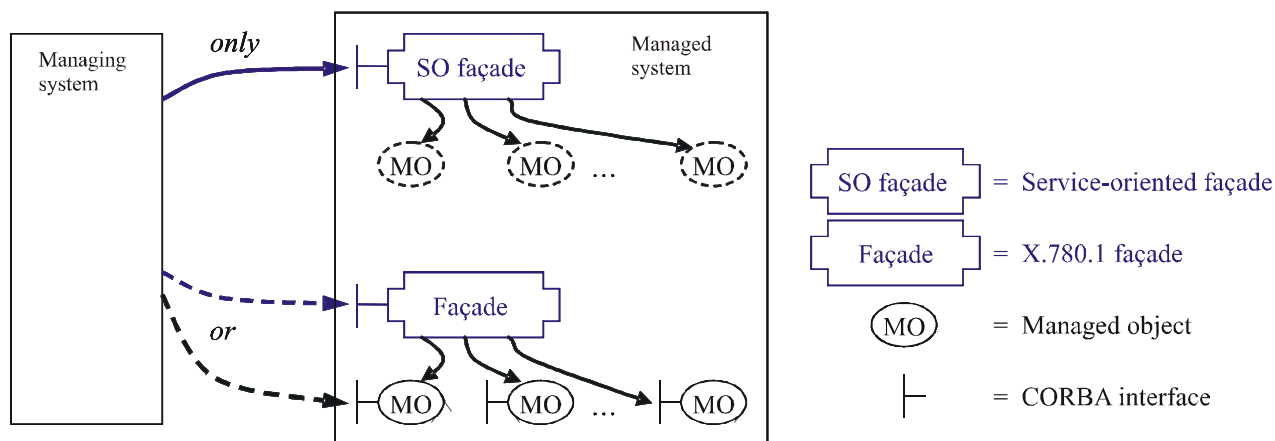


Figure 1 – SO façade and façade of a managed system

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 operations of the `Common_I` interface in Annex A.

7.2.1.1 Relating managed objects to their service-oriented façade

While managed objects may no longer have unique IORs, they will still have unique names and can still be thought of as individual entities representing manageable resources even though they may not have an individual CORBA IDL interface. It is important that a managing system be able to determine which service-oriented façade to use based only on the managed object's name (see clause 6.2 of [ITU-T Q.816.1]). If it cannot, it will have to query the managed system or persistently associate a façade IOR with every managed object's name. To support the ability to determine a managed object's façade based on only its name, the coarse-grained framework extends the names of managed objects accessible through a façade slightly beyond the names of managed objects not accessible through a façade (see clause 7.2.2 of [ITU-T X.780.1] and clause 8.1.3 of [ITU-T Q.816.1] for details, and clause 10.3.2 for a concise overview).

For reasons of compatibility the service-oriented framework keeps this name extension as an option but also adds a lightweight alternative that allows for simple determination of a managed object's responsible SO façade without the need to change the naming conventions. This option is based on the following rules for service-oriented façade and managed object modelling:

- Each managed object class (MOC) is managed by exactly one service-oriented façade interface (managing object class) but a given service-oriented façade may manage more than one MOC.
- Managed object names and containment relationships are stored, possibly redundantly, by the service-oriented façade that is responsible for the management of the respective managed object (i.e., owns the managed object in a steward role) (see clauses 7.2.2 and 10.2).
- Each service-oriented façade interface is instantiated only once (singleton design pattern⁴).
- From the (full distinguished) name of a managed object, its MOC and its façade steward can be determined unambiguously (deterministic naming – see clause 10.3.2).

As a consequence, the name of any managed object instance will identify uniquely the service-oriented façade instance which is managing that managed object. Refer to clause 10.3 for details on naming conventions for service-oriented CORBA managed objects and façade objects.

7.2.1.2 Relationship to SOA concepts

It is believed that the novel service-oriented approach to CORBA-based TMN interface design can pave the way for introducing SOA principles to ITU-T's management interface specification methodology (M.3020-series of ITU-T Recommendations). Therefore, the most basic SOA principles and artefacts are defined and described in considerable detail, and a mapping between SOA and CORBA concepts is provided. Refer to clause 7.2.1.2 of the companion Recommendation [ITU-T Q.816.2] for details.

⁴ The singleton design pattern [b-GAMMA] is a creational pattern with the intent to ensure a class only has one instance, and provide a global point of access to it. A singleton class is made itself responsible for keeping track of its sole instance. The class can ensure that no other instance can be created (by intercepting requests to create new objects), and it can provide a way to access its unique instance.

7.2.2 Storage of names and object containment relationships

In the coarse-grained framework, all façade objects have names that are stored in an OMG naming service and the ITU-T containment service is introduced as a central place to store managed object names and containment relationship information for façade-accessible managed objects. The alternative to store managed object names in the façade objects is mentioned but excluded.

In the service-oriented framework, façade objects may be accessed without the OMG naming service if the session service is available, and the façade objects are storage locations of managed object names and their containment relationships if the containment service is not available. These options are a federated and more lightweight means of object name administration that eliminates the threat for performance issues with the containment service similar to those known with the naming service in case of fine-grained interfaces with a gigantic number of objects.

Service-oriented façades provide for the managed objects they own (see clause 7.2.1.1) standard accessor functions to retrieve all names, individual objects by name and, where applicable, contained objects and containing objects. Refer to clauses 9.2, 9.3 and 10.3 for details.

8 Service-oriented object model IDL

Before specifying in clauses 10 and 11 the rules and style idioms for defining service-oriented TMN managed objects and façade objects using CORBA IDL [OMG 98-07-01] [OMG 99-10-07] [OMG 01-02-33], this and the next clause present network management modules containing a set of object interfaces and supporting data structures specified in CORBA IDL that comply with the modelling guidelines and style guide. These IDL modules provide the basic set of IDL definitions on which service-oriented IDL information models are then built. This IDL is included in Annex A.

Clause 7 outlined the constituents of the framework for the service-oriented paradigm covering some of the issues stated in clause 6. This and the next clause continue the description of the extensions of the CORBA framework to support service-oriented interfaces by setting out the considerations covered by the service-oriented object model in IDL provided in Annex A. This clause describes the supporting data types, managed object classes, exceptions and notifications, similar to clause 5 of [ITU-T X.780] for fine-grained interfaces, whilst the next clause describes the façade interfaces, similar to clause 8 of [ITU-T X.780.1] for coarse-grained interfaces.

This clause specifically covers the following key aspects of the service-oriented object model:

- An overview of the very small IDL repertoire required for service-oriented modelling.
- Use of name/value pairs with string values as a principle, and the specific cases:
 - capability support;
 - additional info parameter;
 - transmission parameter;
 - managed object name component (RDN).
- Use of compact list and tabular structures:
 - list of name/value pairs with string values, and the specific cases:
 - list of capabilities with indication of support;
 - list of additional info parameters ("additionalInfo" attribute);
 - list of transmission parameters (untagged "transmissionParams" attribute);
 - name/value pair with list of string values (multi-valued parameter);
 - list of multi-valued parameters;
 - tagged list of name/value pairs with string values (tagged parameters);
 - list of tagged parameters.

- Approach taken to dealing with the representation of parameters for multi-layered objects:
 - layered transmission parameters (i.e., tagged list with transmission Layer rate as tag) (single-layer "transmissionParams" attribute);
 - list of layered transmission parameters (multi-layer "transmissionParams" attribute).
- Approach taken to naming of the objects used in service-oriented interfaces.
- The attributes common to all service-oriented managed object classes:
 - the "name" of the object;
 - a "userLabel" for the object;
 - a "nativeEMSName" for the object;
 - an "owner" attribute;
 - the "additionalInfo" attribute (list of additional info parameters).
- Lightweight means to inherit the common attributes and to add MOC-specific attributes.
- The MOC-specific attributes that characterize single- and multi-layered objects.
- Lightweight concept of operation exception.
- Lightweight approach to notification definition.

This clause provides details on each of the above including IDL fragments taken from the CORBA modules `globaldefs`, `common`, `transmissionParameters`, and `notifications` (see Annex A).

8.1 IDL repertoire and foundation IDL

The foundation IDL of the service-oriented framework (i.e., the SO modelling IDL in Annex A and the SO framework support services IDL in Annex A of [ITU-T Q.816.2]) uses only the following OMG IDL grammatical constructs of CORBA 2 (see chapter 3 of [OMG 98-07-01] [OMG 99-10-07] [OMG 01-02-33]):

- module, interface
- typedef, short, unsigned long, unsigned long long, boolean, any, struct, union, switch, case, enum, sequence, string
- exception, void, oneway, in, out, inout, raises

Figure 2 – Service-oriented IDL repertoire

Using this IDL repertoire, a set of IDL data types, managed object classes, exceptions, notifications, and interfaces have been defined in Annex A and Annex A of [ITU-T Q.816.2] to support service-oriented modelling. Table 1 provides a summary of these IDL specifications.

Table 1 – Service-oriented foundation IDL

IDL	Brief description
Defined in module "globaldefs" in itut_x780_2.idl of Annex A	
NameAndStringValue_T	A structure for representing a name and value pair which uses the string type for the value field (NVS pair)
NVSList_T	A list of name and string value pairs (list of NVS pairs)
NamingAttributes_T	A hierarchical name structure (NVSList_T) representing the full distinguished name of a managed entity that is not instantiated as a first class CORBA object (i.e., does not possess an IOR)
NamingAttributesList_T	A list of naming attribute lists (i.e., a list of lists)
NameAndStringList_T	A structure for representing multi-valued parameters with string values (NVL pair)
NLSList_T	A list of NameAndStringList_T (list of NVL pairs)
TaggedParameters_T	A tagged list of name and string value pairs
TaggedParameterList_T	A list of tagged lists of name and string value pairs
Time_T	A string representing a generalized time (absolute time)
Duration_T	An integer representing a duration of time (relative time)
ExceptionType_T	An enumerated list of possible exception codes
ProcessingFailureException	An exception appended with a string (errorReason) indicating further details about the exception
NamingAttributesIterator_I	A CORBA interface representing an iterator for bulk retrieval of managed object names
IterableManagedEntity_T	A structure representing a generic managed object type which is iterable for bulk retrieval of managed objects
IterableManagedEntityList_T	A list of iterable managed objects
ManagedEntityIterator_I	A CORBA interface representing an iterator for bulk retrieval of managed objects
Defined in module "common" in itut_x780_2.idl of Annex A	
Capability_T	A name and string value pair representing the functionality supported by the EMS across an NMS-EMS interface
CapabilityList_T	A list of capabilities presenting the functionalities supported by the EMS across an EMS-NMS interface
Common_T	A structure defining the mandatory common attributes of each managed object class (including the full distinguished name, user label, native EMS name, owner and additional info parameters) and allowing for extensions by object class-specific attributes
ManagedObjectList_T	A list of generic managed objects (i.e., which have only common attributes)
ManagedObjectIterator_I	A CORBA interface representing an iterator for bulk retrieval of managed objects

Table 1 – Service-oriented foundation IDL

IDL	Brief description
Common_I	A CORBA interface representing a set of services and utilities that is inherited by every service-oriented façade interface, including retrieval of all managed object names, retrieval of all managed objects, retrieval of a specific managed object, setting the native EMS name of a specific managed object, setting the user label of a specific managed object, setting the owner of a specific managed object, getting the capabilities of the manager object (a first class CORBA object) and setting additional information of a specific managed object
Defined in module "transmissionParameters" in itut_x780_2.idl of Annex A	
LayerRate_T	An integer representing the layer of a transport entity
LayerRateList_T	A list of Layer rates
LayeredParameters_T	A structure representing the Layer rate along with the applicable list of transmission parameters
LayeredParameterList_T	A list of Layer rates along with the applicable list of transmission parameters of each of the Layer rates
Defined in module "notifications" in itut_x780_2.idl of Annex A	
NT_OBJECT_CREATION	A constant identifying the OCN notification type
NT_OBJECT_DELETION	A constant identifying the ODN notification type
NT_ATTRIBUTE_VALUE_CHANGE	A constant identifying the AVC notification type
NT_STATE_CHANGE	A constant identifying the SCN notification type
NT_FILE_TRANSFER_STATUS	A constant identifying the FTS notification type
NT_BACKUP_STATUS	A constant identifying the BSE notification type
NT_HEARTBEAT	A constant identifying the HBE notification type
NT_ALARM	A constant identifying the ALM notification type
NT_TCA	A constant identifying the TCA notification type
Event_T	A struct specifying the notification structure for all notification types – this is the OMG structured event
EventList_T	A list of notifications (i.e., OMG structured events)
EventType_T	A struct specifying the event type of a notification – this is the OMG event type consisting of domain_name and type_name
EventName_T	A string holding the OMG event name of the event
NotificationId_T	A string holding the notification ID of the event
ObjectName_T	A name identifying the managed object reporting the event
ObjectType_T	An enum identifying the type of object that is a filterable field attribute of all notifications
ObjectTypeQualifier_T	A string for identifying object types of future releases
EmsTime_T	A Time_T holding the event report time provided by the EMS
NeTime_T	A Time_T holding the event report time provided by the NE
EdgePointRelated_T	A boolean indicating whether an event is related to an edge TP
OCN_T	A struct specifying the OCN notification type

Table 1 – Service-oriented foundation IDL

IDL	Brief description
ODN_T	A struct specifying the ODN notification type
NameAndAnyValue_T	A structure for representing a name and value pair which uses the any type for the value field (NV pair)
NVList_T	A list of name and value pairs in which the values of the value fields are of any type (list of NV pairs)
AVC_T	A struct specifying the AVC notification type
SCN_T	A struct specifying the SCN notification type
FileTransferStatus_T	An enum describing the status of a file transferring process
FTS_T	A struct specifying the FTS notification type
Current_OperationStatus_T	An enum identifying the status of an NE with respect to a database backup operation
BackupStatus_T	A struct specifying the status of a backup operation for an NE
BSE_T	A struct specifying the BSE notification type
HBE_T	A struct specifying the HBE notification type
NativeEMSName_T	A string representing the name of the managed object that reports the event as presented at the EMS GUI
PerceivedSeverity_T	An enum indicating the perceived severity of an alarm
PerceivedSeverityList_T	A list of perceived severities
ServiceAffecting_T	An enum describing the impact of a fault on the monitored entity
AffectedTPLList_T	A list of names of TPs affected by an alarm
ProbableCause_T	A string representing the probable cause of an alarm
ProbableCauseList_T	A list of probable causes
ProbableCauseQualifier_T	A string used to fully qualify the probable cause, if needed
NativeProbableCause_T	A string representing the value of a probable cause as portrayed on the EMS user interface
AssignedSeverity_T	An enum indicating an assigned severity of a managed object
AlarmSeverityAssignment_T	A struct specifying the alarm severity assignment (ASA)
AlarmSeverityAssignmentList_T	A list of alarm severity assignment
AdditionalText_T	A string holding additional textual information
AdditionalInfo_T	An NVSList_T that holds additional information
RcaiIndicator_T	A boolean indicating whether an alarm is a raw alarm or a root cause alarm indication (RCAI)
IsClearable_T	A boolean indicating clearability of an event
AcknowledgeIndication_T	An enum indicating the acknowledge state of an event
X733_EventType_T	A string holding the definition of the event type of an alarm according to [ITU-T X.733]
X733_EventTime_T	A Time_T holding the time of generation of an alarm
SpecificProblem_T	A string qualifying a probable cause
SpecificProblemList_T	A list of specific problems
X733_BackupStatus_T	A string specifying whether the managed object reporting an alarm has been backed-up or not

Table 1 – Service-oriented foundation IDL

IDL	Brief description
X733_BackupObject_T	The name of the managed object providing back-up services for the managed object about which an alarm pertains
X733_TrendIndication_T	A string specifying the current perceived severity trend of the managed object reporting an alarm
NotifIDList_T	A list of notification IDs
CorrelatedNotifications_T	A structure identifying the related notifications by pointing to an object and the notifications emitted by the object
CorrelatedNotificationList_T	A list of correlated notification lists
X733_MonitoredAttributes_T	An NVList_T providing attributes of the monitored object
ProposedRepairAction_T	A string describing the proposed repair action
ProposedRepairActionList_T	A list of proposed repair actions
X733_StateChange_T	An NVList_T providing X.731 state transition information
X733_AdditionalInfo_T	An NVList_T providing additional information in alarms
AlarmId_T	A structure uniquely identifying an alarm ID
ALM_T	A struct specifying the ALM notification type
PMPParameterName_T	A string holding a PM parameter name
PMLocation_T	A string specifying a PM location
Granularity_T	A string specifying a count period for PM data collection
PMThresholdType_T	An enum describing the threshold types for TCA parameters
TCAId_T	A structure uniquely identifying a threshold crossing alert ID
Value_T	A float holding a PM threshold value
Unit_T	A string holding the unit of measurement of a threshold value
TCA_T	A struct specifying the TCA notification type
AlarmTypeQualifier_T	An enum used to distinguish threshold crossing alert from alarm
AlarmOrTCAIdentifier_T	A switch between an alarm ID or a TCA ID
AlarmAndTCAIDList_T	A list of alarm IDs and TCA IDs (could be a mixed list)
EventIterator_I	A CORBA interface representing an iterator for bulk retrieval of notifications (events, alarms, alerts)
Defined in module "idlVersion" in itut_q816_2.idl of Annex A of [ITU-T Q.816.2]	
Version_I	A CORBA interface that allows the NMS (managing system) to query the current version of the IDL interface (IDL version) implemented by the EMS (managed system)
Defined in module "session" in itut_q816_2.idl of Annex A of [ITU-T Q.816.2]	
Session_I	A CORBA interface that provides capabilities to enable either a client or a server to detect the loss of communication with the associated party and to enable either party to end the association
Defined in module "nmsSession" in itut_q816_2.idl of Annex A of [ITU-T Q.816.2]	
NmsSession_I	A CORBA interface that allows an EMS to notify all connected NMSes about failures to push an event, about the event/alarm loss period being over, about discarding of alarms, TCAs, file transfer statuses, or notifications of other specified types

Table 1 – Service-oriented foundation IDL

IDL	Brief description
Defined in module "emsSession" in itut_q816_2.idl of Annex A of [ITU-T Q.816.2]	
ManagerNames_T	A list of names of service-oriented façades
EmsSession_I	A CORBA interface that allows an NMS to request the manager interfaces that the EMS implements to gain access to a specified manager interface without using the OMG naming service and to gain access to an event channel for receiving notifications
Defined in module "emsSessionFactory" in itut_q816_2.idl of Annex A of [ITU-T Q.816.2]	
EmsSessionFactory_I	A CORBA interface that allows an NMS to obtain the EmsSession_I object from which all managers (i.e., service-oriented façade objects) of the EMS can be obtained

Refer to clause 10 for additional IDL repertoire that is recommended for optional use when extending the foundation IDL for domain-specific applications.

8.2 Use of name/value pairs with string values

As a rule, OMG specifications define properties of entities as name/value pairs with `string` names and `any` values [OMG 98-07-01] [OMG 99-10-07] [OMG 01-02-33] [OMG 04-10-13] [OMG 00-06-22] but also introduce such name/value pairs with `string` values [OMG 04-10-03]. 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` (NVS pairs):

```
module globaldefs
{
...
    struct NameAndStringValue_T
    {
        string name;
        string value;
    };
...
}
```

The rationale for such usage and the "name=value" semantics are set out in clause 10.1. NVS pair structures are available in the IDL where extensibility is recognized as necessary and use of the IDL type `any` is unsuitable for good reasons. There are several cases of use in the service-oriented framework which are detailed below, such as "capability support", "additional info parameter", "transmission parameter", and the "name component" of managed objects.

8.3 Single-valued, multi-valued and tagged parameters

List and tabular structures are used extensively to provide a compact and efficient representation of potentially complex data that is to be conveyed as a single unit. In many cases these structures are built from name/value pairs. IDL describing such cases is provided below (without comments).

```
module globaldefs
{
...
    typedef sequence<NameAndStringValue_T> NVSList_T;
...
    typedef NVSList_T NamingAttributes_T;
...
    typedef sequence<NamingAttributes_T> NamingAttributesList_T;
...
}
```

```

module common
{
...
    typedef globaldefs::NameAndStringValue_T Capability_T;
...
    typedef sequence<Capability_T> CapabilityList_T;
...
    ...
    globaldefs::NVList_T additionalInfo;
    ...
...
}

module transmissionParameters
{
...
    ...
    globaldefs::NVList_T transmissionParams;
    ...
...
}

module globaldefs
{
...
    struct NameAndStringList_T
    {
        string name;
        sequence<string> value;
    };
...
    typedef sequence<NameAndStringList_T> NLSList_T;
...
    struct TaggedParameters_T
    {
        unsigned long parameterTag;
        NVList_T taggedParams;
    };
...
    typedef sequence<TaggedParameters_T> TaggedParameterList_T;
...
}

```

NVList_T provides a list of name/value pairs with string values (NVS pairs) that may be used to encode multi-valued attributes as a list of single-valued parameters or components. Examples for such lists are NamingAttributes_T (see clause 8.5), CapabilityList_T (see clause 9.3.2.7), the additionalInfo attribute (see clauses 8.5, 9.2 and 9.3.2.8), and the transmissionParams attribute (see clauses 8.4, 9.2 and 10.4).

String values can be used quite universally to encode list and tabular structures when their use is documented in sufficient detail in a standardized or vendor-specific way. This usage may lead to rather unstructured and complicated contents of the value fields of the NV pairs, however.

More structured and nevertheless lightweight approaches are multi-valued parameters and tagged parameters. NameAndStringList_T generalizes NameAndStringValue_T to name/value pairs where the value is a list of strings (sequence<string>) and NLSList_T generalizes NVList_T to a list of name/value pairs with list of string values (NVL pairs). NLSList_T can be substituted wherever NVList_T is used in order to provide proper tabular structures: the elements of the sequence of NLSList_T can represent table rows while the elements of the sequence of NameAndStringList_T can represent table columns.

TaggedParameters_T generalizes NVList_T to a list of name/value pairs which is qualified by a parameter tag, and TaggedParameterList_T is a list of such tagged parameters. The tag type unsigned long used here could also be another appropriate IDL type such as short or string. A

third generalization of NVSList_T would be tagged parameters of type NLSList_T, i.e., the combination of the first and second generalization of NVSList_T (see clause 10.6).

Examples of use of these structures are highlighted throughout clauses 8 and 9.

8.4 Modelling of transmission technologies

Whilst the framework can be applied for the modelling of any OS-OS interface in a service-oriented way, a particularly important application is the case of an NMS-EMS interface where the EMS is in contact with a physical telecom network that may encompass a number of transport, access and aggregation technologies. The framework therefore provides a generic and flexible means to model any of these network technologies down to any required detail and in the context of protocol stacks and multiplexing hierarchies. This approach is called multi-technology capability for short and is based on the layered transport network functional architecture of [ITU-T G.805] and the corresponding management abstractions provided in [ITU-T G.852.2].

The structures LayeredParameters_T and LayeredParameterList_T provide support for layering of the network protocols according to [ITU-T G.805]. A transmission layer per [ITU-T G.805] is determined by a characteristic information and an adapted information which can be identified by a standard tag. This tag is called **Layer rate** and encoded as LayerRate_T. The IDL describing the multi-technology structures as tagged parameters is provided below (without comments).

```
module transmissionParameters
{
...
    typedef short LayerRate_T;
...
    typedef sequence<LayerRate_T> LayerRateList_T;
...
    struct LayeredParameters_T
    {
        LayerRate_T layer;
        globaldefs::NVSList_T transmissionParams;
    };
...
    typedef sequence<LayeredParameters_T> LayeredParameterList_T;
...
}
```

The structures convey both the name or tag of the layer and for each layer a list of attributes. Each attribute is conveyed by the transmissionParams structure as a part of LayeredParameters_T. The transmissionParams structure is a list of name/value pairs that provides a capability to extend definitions of layered managed objects both for proprietary purposes and for standard definitions where inter-release backward compatibility is a concern. In a majority of cases, properties conveyed by LayeredParameters_T can be considered as optional. This structure hence covers all attributes that relate to specific transmission protocol layers as specified by the characteristic information and adapted information concepts of [ITU-T G.805]. The encoding and modelling of such specific layers is out of scope of this generic framework for OS-OS interface design but is left to domain-specific applications and extensions. The structure LayeredParameterList_T encompasses multiple layers and their transmission parameters according to client/server relationships between adjacent layers as specified in [ITU-T G.805]. Refer to clause 10.4 regarding guidelines for the modelling of layered transmission technologies in IDL. In the interest of interoperability, users of the service-oriented CORBA framework for the TMN must cooperate in reaching normative usage of standardized Layer rates (and associated transmission parameters). Details of such cooperation are for further study. The value 0 is not used, and the value 1 is identified as LR_Not_Applicable and used in cases where a Layer rate attribute is not applicable. A Layer rate in the range 0 to 9999 is considered as a standardized Layer rate, while the range of numbers 10 000 and above is allocated for vendor-specific usage and a Layer rate in this range is considered as a proprietary Layer rate. The standardized Layer rates specified by MTNM v3.0 [TMF LayerRates] shall be used

as a starting point when an interface specification extends this framework, i.e., the extension shall only define new Layer rates when no value specified by [TMF LayerRates] fits the underlying requirements (see also Table 3).

NOTE – [ITU-T G.805] is concerned with connection-oriented network technologies. The multi-technology capability of the service-oriented framework applies equally well to connectionless network technologies as defined in [ITU-T G.809]. See clause 10.4 for some details.

8.5 Common attributes `Common_T` of service-oriented managed objects

Service-oriented managed object classes and service-oriented façade interfaces shall inherit from a set of common attributes `Common_T`, or from a root service-oriented façade interface called `Common_I` (which is often virtual). The `Common_I` "managing object" class is described in clause 9, and the IDL describing the `Common_T` MOC is provided below (without comments).

```
module common
{
...
    struct Common_T
    {
        globaldefs::NamingAttributes_T name;
        string userLabel;
        string nativeEMSName;
        string owner;
        // add any MOC-specific attributes here
        globaldefs::NVSList_T additionalInfo;
    };
...
}
```

The `Common_T` managed object class has the following attributes:

- **name:** The full distinguished name (FDN) of the managed object (`NamingAttributes_T`). The name is structured as an ordered list of one or more relative distinguished names (RDNs) or name components. The order of the RDNs reflects the containment relationship of the managed object and each RDN is an NVS pair (`NameAndStringValue_T`) where the value may itself be substructured (see clause 10.3). The FDN is thus a list of name/value pairs (`NVSList_T`). This FDN structure is equivalent to the CORBA name structure (see [OMG 04-10-03] and clause 6.4 of [ITU-T Q.816.2]). The FDN is unique within the scope of the EMS and is unchanging through the lifecycle of the managed object, i.e., the name cannot be changed during the life of the object. The name may be chosen by the EMS or NMS depending upon the specific managed object class behaviour. The name is the primary reference used when acquiring or identifying the object.
- **userLabel:** A friendly/familiar name by which the human operator recognizes the object. The `userLabel` is not guaranteed to be unique but the EMS may support enforcement of uniqueness.
- **nativeEMSName:** A representation of how the managed object is referenced on the EMS displays. The aim is to provide a "nomenclature bridge" to aid the relating of information presented on NMS displays to that on the EMS displays. A mechanism to allow setting of this attribute is available to the NMS. The EMS may reject changes to the value of this attribute. When an object is created by the EMS, the EMS selects the `nativeEMSName` for the object.
- **owner:** The owner of the managed object as provisioned by the NMS, for example, in the sense of a "used by" relationship. There is no specified behaviour related to this attribute, and it is considered simply as a label.
- **additionalInfo:** A list of name/value pairs that provides a capability to extend the object definition both for proprietary purposes and for standard definitions, in particular where inter-release backward compatibility is a concern. In a majority of cases, property conveyed

by `additionalInfo` can be considered as optional. Additional information hence covers all attributes that are not specific to a layer instance (see clause 8.4 above) and which are not explicitly modelled at the NML-EML interface.

The `Common_T` structure represents a service-oriented managed object class without explicitly modelled attributes except the common attributes. It is extended to specific managed object classes by simply adding further record members, preferably between owner and `additionalInfo`, and renaming. This procedure is called lightweight inheritance. It represents the second alternative way of MOC modelling introduced in clause 6.2.2, which is particularly useful for applications that deliberately define very few MOCs. Capabilities of these lightweight managed objects are defined through operations of service-oriented façade instances (see clause 9).

(R) OBJECT-1 Service-oriented resources on a managed system shall be modelled as IDL structs that inherit lightweightly (directly or indirectly) from the `Common_T` struct described above and defined in the CORBA IDL in Annex A. The capabilities described in clause 9 shall be supported.

When `Common_T` is extended to model specific managed object classes, the service-oriented framework recommends to use attributes of type `string` or `NVSLIST_T` as far as possible and to avoid certain standard IDL types such as `any` and `enum` (refer to clause 10.1 for details). When attributes of type `string` are used, their contents can be standardized in any desired way by appropriate documentation though this weakly typed, lightweight approach does not exploit the type checking capabilities of IDL compilers and ORB runtime environments.

For example, it is recommended to encode an absolute time value (calendar date and time of day) as a string with the convention that the contents are encoded according to clause 42 of ITU-T Recommendation X.680 (generalized time type). The IDL describing this useful type is provided below (without comments).

```
module globaldefs
{
...
    typedef string Time_T;
...
}
```

The X.680 string format is "yyyyMMddhhmmss.s[Z|{+|-}HHMm]" (see IDL comments).

It is further recommended to adopt the OMG definition `TimeBase::TimeT` to encode a relative time value (duration). The IDL describing this useful type is provided below (without comments).

```
module globaldefs
{
...
    typedef unsigned long long Duration_T;
...
}
```

NOTE – The OMG time service specification includes the CORBA module `TimeBase` with basic data structures pertaining to time-related CORBA modules (e.g., OMG's `CosNotification`). Relative time values (duration) are encoded as "typedef unsigned long long TimeT;" with unit 100 nanoseconds, and absolute time values by the 16-byte struct `UtcT`, which uses `TimeT` with base "15 October 1582 00:00" of the Gregorian calendar. But `UtcT` may also represent relative times depending on context.

The ITU-T module `itut_x780` adopts `TimeBase::UtcT` for the encoding of absolute and relative time values. The above definitions of the service-oriented framework are more lightweight and straightforward, and are independent of OMG's time service specification.

(R) OBJECT-2 When the `Common_T` struct is extended to model specific managed object classes, the types defined in the modules of Annex A shall be used wherever possible and reasonable. The admissible contents of strings shall be specified in as much detail as possible.

8.6 Exceptions

The principle of the use of exceptions is set out in clause 10.1. The specific exception types are listed below. `ExceptionType_T` which is an `enum`. The unique exception is augmented using a free format `errorReason` which is a `string`. The IDL describing the unique exception to be used for all operations of the service-oriented framework is provided below (without comments).

```
module globaldefs
{
...
    enum ExceptionType_T
    {
        // see below for the standard values
    };
...
    exception ProcessingFailureException
    {
        ExceptionType_T exceptionType;
        string errorReason;
    };
...
}
```

Where an operation needs to be rejected, the following values are specified for the `exceptionType` `enum` and should be used in the response that rejects the operation:

- **EXCPT_NOT_IMPLEMENTED:**
Used if some IDL operations are optional or not implemented in the specific implementation. If the operation itself is not supported, then `errorReason` shall be an empty string. If this exception is raised because of the values of specific parameters, then the names of these parameters shall be supplied in `errorReason` (separated by commas) unless otherwise specified in the operation description.
- **EXCPT_INTERNAL_ERROR:**
Used to indicate an EMS internal error. Applies to all operations. The `errorReason` can be used to provide further clarification.
- **EXCPT_INVALID_INPUT:**
Used if the format of an input parameter is incorrect, for example, if a name which is a 3-level naming attribute is passed as a single level name, then this value may be used in the response that rejects the operation. Also, if a parameter is out of range, this type will be used. The `errorReason` field will be filled with the parameter that was incorrect.
- **EXCPT_OBJECT_IN_USE:**
Used to indicate an object already in use where the request to use the object would conflict with the current use.
- **EXCPT_ENTITY_NOT_FOUND:**
Used if the NMS supplies an object name as a parameter to an operation and the EMS cannot find the object with the given name, then an exception of this type is returned. The `errorReason` field in the exception will be filled with the name that was passed in as parameter.
- **EXCPT_NOT_IN_VALID_STATE:**
Used if the client tries to delete an object that is in a state that prevents its deletion.
- **EXCPT_UNABLE_TO_COMPLY:**
Used as a generic value when a server (EMS) cannot respond to the request and no other exception provides the appropriate clarification. The `errorReason` string is used to provide further information.

- `EXCPT_NE_COMM_LOSS`:
Used as a generic value when a server (EMS) cannot communicate with the NE (network element) to which the operation should apply and that prevents the successful completion of the operation. Note that network element here applies to any device to which the server (EMS) will have to communicate to fulfil the operation.
- `EXCPT_CAPACITY_EXCEEDED`:
Used when an operation will result in resources being created or activated beyond the capacity supported by the NE/EMS.
- `EXCPT_ACCESS_DENIED`:
Used when an operation results in a security violation.
- `EXCPT_TOO_MANY_OPEN_ITERATORS`:
Used when an EMS exceeds its internal limit of the number of iterators it can support.
- `EXCPT_USERLABEL_IN_USE`:
Raised when the userLabel uniqueness constraint cannot be met.

This standard framework exception list may be extended with more focused exceptions that relate to the problem space to which the respective interface using the framework is applied. For example, an exception `EXCPT_UNSUPPORTED_ROUTING_CONSTRAINTS` may be specified where the framework has been used to develop an interface that relates to the problem of routing across a network. Standard OMG exceptions (see clauses 3.15 of [OMG 98-07-01], 3.17 of [OMG 99-10-07] and 4.11 of [OMG 01-02-33]) may also occur, e.g., `NO_IMPLEMENT` if an operation implementation is unavailable, but they should be documented with IDL comments similar to the standard ITU-T exceptions.

Note that modifications of enumerated types such as `ExceptionType_T` are not backward compatible. Refer to clause 10.5.2.4 for a definition of extensibility for such types and a mechanism how to extend an extensible enum. `ExceptionType_T` becomes extensible when `EXCPT_INTERNAL_ERROR` is considered to be the escape value and `errorReason` is used to convey new exception values in accordance with the convention "`EXCPT_<new exception type>;<errorReason>`" similar to the requirement **OBJECT-2** when extending `Common_T`.

8.7 Notifications

The structure and behaviour of notifications of the TMN CORBA framework, no matter whether fine-grained or coarse-grained or service-oriented interfaces are considered, are based on the concepts of the OMG notification service (see clauses 6.5 and 8.3 of [ITU-T Q.816.2]). The fine-grained and coarse-grained frameworks take the 15 event types specified by [ITU-T X.721] and the X.730-series of ITU-T Recommendations as the notification types and define an IDL operation for each type, thereby specifying each type as an OMG typed event (see interface `Notifications` of `itut_x780.idl`). They require use of structured or typed events, with typed events being the preferred choice. In case of service-oriented interfaces, structured events are the preferred choice and typed events are an option. The service-oriented framework defines nine event types and an IDL struct for each type, thereby fixing all details of the associated structured event in a compilable way (see module `notifications` of `itut_x780_2.idl`). This clause summarizes the lightweight approach to notification definition.

8.7.1 Notification types and common event fields

The supported standard notification types are defined by constants and, for each type, a three-letter acronym is used as a shortcut. The IDL describing these notification types is provided below.

```
module notifications
{
...
    // Object Creation Notification (OCN)
    const string NT_OBJECT_CREATION = "NT_OBJECT_CREATION";
    // Object Deletion Notification (ODN)
    const string NT_OBJECT_DELETION = "NT_OBJECT_DELETION";
    // Attribute Value Change notification (AVC)
    const string NT_ATTRIBUTE_VALUE_CHANGE = "NT_ATTRIBUTE_VALUE_CHANGE";
    // State Change Notification (SCN)
    const string NT_STATE_CHANGE = "NT_STATE_CHANGE";
    // File Transfer Status notification (FTS)
    const string NT_FILE_TRANSFER_STATUS = "NT_FILE_TRANSFER_STATUS";
    // Backup Status Event notification (BSE)
    const string NT_BACKUP_STATUS = "NT_BACKUP_STATUS";
    // Heartbeat Event notification (HBE)
    const string NT_HEARTBEAT = "NT_HEARTBEAT";
    // Alarm notification (ALM)
    const string NT_ALARM = "NT_ALARM";
    // Threshold Crossing Alert notification (TCA)
    const string NT_TCA = "NT_TCA";
...
}
```

An interface specification that extends this framework may define additional event types according to the rule `const string NT_<notification type> = "NT_<notification type>;`. The encoding of event types as defined by these constants shall be used by the managed system in the common field `type_name` of all notifications (see below). To guarantee forward compatibility, the managing system should ignore unrecognized notification types.

The attributes of each standard notification type are defined through an appropriate IDL struct in order to fix all standard details in a compilable way (OCN_T, ODN_T, AVC_T, SCN_T, FTS_T, BSE_T, HBE_T, ALM_T, TCA_T). All definitions are specializations of OMG structured events as defined by the OMG notification service specification [OMG 04-10-13], and lists of events (of usually mixed types) are therefore OMG event batches (see clause 8.7.5). These event definitions shall be used for all transport mechanisms such as push/pull model and structured/typed events (see clauses 6.5, 8.3 and 10.2.1 of [ITU-T Q.816.2]). The corresponding IDL is provided below.

```
module notifications
{
...
    typedef CosNotification::StructuredEvent Event_T;
...
    typedef sequence<CosNotification::StructuredEvent> EventList_T;
...
}
```

An interface specification that extends this framework may refine the standard notification types with additional record members. In this context it is also allowed to refactor definitions from the `notifications` module to other CORBA modules (e.g., when adding further PM details to TCA notifications). When name/value pairs are used to define the fields of the filterable event body of structured events, the field names shall be the names of the record members of the struct defining the notification type, and the actual field types shall be the types of these record members, except possibly when such a type is itself a struct, in which case the record members of the nested struct shall be considered directly (e.g., in case of `AlarmId_T` for `ALM_T`).

The common fields of most notification types are `eventType`, consisting of `domain_name` and `type_name`, `eventName`, `notificationId`, `objectName`, `objectType`, `objectTypeQualifier`, `emsTime`, `neTime` and `edgePointRelated`. Their IDL data types are either `string` or `boolean` or defined in the

module `globaldefs` (see above) or newly defined. The IDL describing the new data types for common event fields is provided below (without the tagged comments).

```
module notifications
{
...
    typedef CosNotification::_EventType EventType_T;
...
    enum ObjectType_T
    {
        OT_EMS,
        // OT_<object type>, for each object type supported by a standardized
        //                               interface specification that extends this framework
        OT_AID
    };
...
    typedef string ObjectTypeQualifier_T;
...
};
```

The `eventType` field of type `EventType_T` is part of the fixed event header. It consists of the `domain_name` and the `type_name` (see clause 8.3 of [ITU-T Q.816.2]), which are both of type `string`. The values of `domain_name` are defined by the interface specification that extends this framework. The values of `type_name` are the constants `NT_<notification type>` (standard, as defined above, or implementation-specific) that specify the notification types used at the interface.

To comply with the OMG definition of the fixed event header of structured events, each notification has a record member `eventName` of type `string`, which is meant to name a specific instance of an OMG structured event where the semantics of the names is defined by the end users of the notification service (i.e., the managed and managing systems). For reasons of interoperability, it is recommended to ignore this field or to have it always set to `" "`.

Each notification has a record member `notificationId` of type `string`, which maps to a field of the filterable event body of OMG structured events and contains an implementation-defined free format notification identifier whose semantics is specified in clause 8.1.2.8 of [ITU-T X.733]. The uniqueness and the sequence of the notification ID are not guaranteed. When correlation of notifications according to [ITU-T X.733] is supported (see clause 8.7.3.3), notification IDs must be chosen (by the managed system) to be unique across all notifications from a particular managed object throughout the time the considered correlations are significant. Therefore notification IDs may be reused only when no requirements exist for correlating prior notifications and should be chosen to ensure uniqueness over as long a time as is feasible for the managed system.

The record member `objectName` of type `globaldefs::NamingAttributes_T` of an event uniquely identifies the managed object that reports the event (i.e., the event source).

The record member `objectType` of type `ObjectType_T` belongs to all notification types that are reported by a managed object. Standardized interface specifications that extend this framework shall define an `objectType` value for each managed object class that is modelled at the interface (by extending the `Common_T` structure – see clause 8.5). All standard notification types, except FTS and BSE, have this attribute (see below). The `enum` avoids any uncertainty in the type of object and allows simple filtering. It cannot be extended in a backward compatible way, however.

A notification must be reported against the correct object type, if this MOC is modelled at the interface. The standard `objectType` `OT_EMS` represents the managed system (e.g., EMS). For alarms, the `objectType` value `OT_AID` denotes an alarm identifier (AID) and is used to represent the managed system's object types that are not modelled (at the interface) but can emit alarms. Other notification types should not be reported against AIDs. But this `objectType` value shall also be used for potential new object types in future releases of an interface implementation in order to guarantee backward compatibility of the interface. This convention works as follows.

The `objectTypeQualifier` record member of type `ObjectTypeQualifier_T` is used to identify managed object types of future releases of standardized interface specifications that extend this framework. It is needed because the `ObjectType_T` enum, once specified for a particular release of an interface specification, cannot be extended for backward compatibility reasons. So when backward compatibility is required (as usual – see clause 10.5), the enum `ObjectType_T` cannot be extended to include future new object types. Therefore, `OT_AID` is used as an "escape value" for the field `objectType` (see also clause 10.5.2.4), i.e., `OT_AID` may also represent new object types. To identify which of the new object types applies, the filterable field attribute `objectTypeQualifier` is introduced, which is of type `string` and whose values are as follows:

- " " – indicates an AID;
- "OT_<object type>" – Where <object type> identifies a new object type.

If `objectTypeQualifier` is not present but `objectType` has the value `OT_AID`, we have a proper AID.

Most notification types have a record member `emstime` of type `globaldefs::Time_T`, which holds the time at which the event was reported by the managed system, and a record member `neTime` of type `globaldefs::Time_T`, which holds the time provided by the NE if the NE reports time and is optional or " " otherwise. Most notification types have a record member `edgePointRelated` of type `boolean`, which is `TRUE` if the event relates to a TP that is an edge point or to a managed object that is related to an edge TP, and is `FALSE` or not present otherwise.

8.7.2 OCN, ODN, AVC, SCN, FTS, BSE and HBE notifications

An OCN or ODN holds details of the managed object that is being created or deleted. An AVC or SCN, respectively, holds details of the attribute values or state attribute values of a managed object that have changed. An FTS holds details of a file transfer process. A BSE holds details of a change to a network element's backup status. An HBE holds details of an EMS heartbeat notification. The IDL describing these notification types is provided below (without the tagged comments).

```
module notifications
{
...
    struct OCN_T
    {
        EventType_T eventType;
        string eventName;
        string notificationId;
        globaldefs::NamingAttributes_T objectName;
        ObjectType_T objectType;
        ObjectTypeQualifier_T objectTypeQualifier;
        globaldefs::Time_T emsTime;
        globaldefs::Time_T neTime;
        boolean edgePointRelated;
        any remainderOfBody;
        // this record member is mapped to the remaining event body of the
        // OMG structured event and holds a copy of the object which has been
        // created; its actual type depends on the value of objectType and is
        // then the struct defining the specified objectType (see clause 8.5)
    };
...
    struct ODN_T
    {
        EventType_T eventType;
        string eventName;
        string notificationId;
        globaldefs::NamingAttributes_T objectName;
        ObjectType_T objectType;
        ObjectTypeQualifier_T objectTypeQualifier;
        globaldefs::Time_T emsTime;
        globaldefs::Time_T neTime;
        boolean edgePointRelated;
    };
...
}
```

```

struct AVC_T
{
    EventType_T eventType;
    string eventName;
    string notificationId;
    globaldefs::NamingAttributes_T objectName;
    ObjectType_T objectType;
    ObjectTypeQualifier_T objectTypeQualifier;
    globaldefs::Time_T emsTime;
    globaldefs::Time_T neTime;
    boolean edgePointRelated;
    NVList_T attributeList;
};

...
struct SCN_T
{
    EventType_T eventType;
    string eventName;
    string notificationId;
    globaldefs::NamingAttributes_T objectName;
    ObjectType_T objectType;
    ObjectTypeQualifier_T objectTypeQualifier;
    globaldefs::Time_T emsTime;
    globaldefs::Time_T neTime;
    boolean edgePointRelated;
    NVList_T attributeList;
};

...
struct FTS_T
{
    EventType_T eventType;
    string eventName;
    string notificationId;
    string fileName;
    FileTransferStatus_T transferStatus;
    short percentComplete;
    string failureReason;
};

...
struct BSE_T
{
    EventType_T eventType;
    string eventName;
    string notificationId;
    globaldefs::NamingAttributes_T meName;
    globaldefs::Time_T emsTime;
    globaldefs::Time_T neTime;
    BackupStatus_T backupStatus;
};

...
struct HBE_T
{
    EventType_T eventType;
    string eventName;
    string notificationId;
    globaldefs::NamingAttributes_T objectName;
    ObjectType_T objectType;
    // the value of this record member is always OT_EMS
    globaldefs::Time_T emsTime;
};

...
}

```

AVC, SCN and ALM notifications have attributes whose type is a list of name/value pairs with any values. The IDL describing this type is provided below (without comments).

```
module notifications
{
...
    struct NameAndAnyValue_T
    {
        string name;
        any value;
    };
...
    typedef sequence<NameAndAnyValue_T> NVList_T;
...
}
```

NVList_T is equivalent to the NVList standard defined by the OMG to represent (lists of) general properties (see clause 8.2). Refer to the IDL comments in Annex A for a description of usage.

FTS notifications convey the current value of the status associated with a file transfer process. The IDL describing the type of the file transfer status is provided below (without comments).

```
module notifications
{
...
    enum FileTransferStatus_T
    {
        FT_IN_PROGRESS,
        FT_FAILED,
        FT_COMPLETED
    };
...
}
```

Refer to the IDL comments in Annex A for a description of these values and the usage together with the percentComplete and failureReason attributes of FTS notifications.

BSE notifications convey the current value of the operational status associated with a database backup process of a network element (which may well be more sophisticated than a file transfer process). The IDL describing the type of this status is provided below (without comments).

```
module notifications
{
...
    enum Current_OperationStatus_T
    {
        COS_Idle,
        COS_Pending,
        COS_InProgress,
        COS_Completed,
        COS_Aborted
    };
...
    struct BackupStatus_T
    {
        Current_OperationStatus_T opStatus;
        string failureReason;
    };
...
}
```

Refer to the IDL comments in Annex A for a description of the values and record members.

HBE notifications are emitted by the EMS and convey an "I'm alive" absolute time value. Refer to clauses 7.2.5 and 9.1.1 of [ITU-T Q.816.2] and clause 7.5 of [ITU-T Q.816] for information on how this interface may be used or extended to offer a heartbeat service (Q.816.2 session service or Q.816 heartbeat service).

8.7.3 Service-oriented fault management

The lightweight approach to alarm definition covers the mandatory attributes of the alarm record definition of [ITU-T X.721] and [ITU-T X.733], namely managed object class, managed object instance, event type, probable cause and perceived severity, and defines further attributes including all conditional X.733/X.721 attributes. Alarms are specific types of notifications emitted from managed objects concerning detected faults or abnormal conditions. Managed object class (MOC) specifications should include causes of potentially abnormal situations which can then be used to convey with each alarm of the MOC such a cause, called the probable cause of the alarm. Probable causes are used to classify alarm types. When available, additional diagnostic information should be notified (e.g., information related to side effects such as current and past values of state attributes, further qualification of the probable cause). Refer to clause 8.7.3.1 for an overview of the probable causes defined in [ITU-T M.3100], [ITU-T X.721], [ITU-T X.733] and this Recommendation.

[ITU-T X.721] and [ITU-T X.733] introduce five event types for alarms which can also be conveyed with the lightweight alarm structure (see clause 8.7.3.3). Additionally, the service-oriented framework calls the X.733/X.721 alarm with probable cause thresholdCrossed (and event type qualityofService) a threshold crossing alert (TCA) and defines its structure independently of the alarm structure. The TCA structure and behaviour depend on the concepts of service-oriented performance management (PM) outlined in clause 8.7.4. A TCA is an event used across the OS-OS interface to indicate that a PM parameter threshold, associated with a managed object, has been crossed. TCAs have specific PM-related attributes not found in (other) alarms and do not need a probable cause and related attributes, and ALM notifications do not provide any threshold information. TCAs address the requirement of early detection of faults before significant effects have been felt by the user. Degradation of service may be detected by monitoring of error rates. Threshold mechanisms on counters and gauges are a method of detecting such trends and providing a warning to managing systems when the rate becomes, or tends to become, unacceptable (see clause 8.7.4).

An important criterion by which failures of managed objects are to be notified is the level to which the fault degrades the quality of the service that was originally requested by (or promised to) the service user. Malfunctions will range in severity from warning, where there is no impact upon the quality of service offered to the user, to critical, where it is no longer possible to provide the service requested by (or promised to) the service user. The level of severity can be described generically and criteria specified based upon the level of degradation that the fault causes to the service: critical, major, minor, warning, or indeterminate. When conveyed in an alarm, this information is called perceived severity. When used in an MOC specification (on a per probable cause basis), this information is called assigned severity. Refer to clause 8.7.3.2 for details.

A further perceived severity level is cleared, which indicates the clearing of one or more previously reported, clearable alarms or TCAs. A raise/trigger event is called clearable if a clear event can be generated for it after its cause has disappeared. Therefore, the perceived severity attribute allows to distinguish raise/trigger events from clear events. ALM and TCA notifications have a mandatory attribute isClearable of type `boolean` which is an indication as to whether a raise/trigger event will have an associated clear event (or the event is a clear itself). An alarm or TCA is called active if it is not clearable or is clearable but no associated clear notification has been received yet.

NOTE – This framework specifies reportable alarms (i.e., alarm reports). There may well be pending alarms associated with managed objects whose existence can be recognized through an alarm status attribute (see clauses 8.7.3.2 and 8.7.5.3). In the presence of alarm reporting control (ARC) (see [ITU-T M.3100]), i.e., when alarm reporting is inhibited, all alarms are pending. Pending alarms are also considered active.

Alarms have an attribute serviceAffecting of type `ServiceAffecting_T` which specifies the impact of a fault on monitored managed entities on further providing their service, as determined by the managed system. The IDL describing this data type is provided below (without comments).

```

module notifications
{
...
    enum ServiceAffecting_T
    {
        SA_UNKNOWN,
        SA_SERVICE_AFFECTING,
        SA_NON_SERVICE_AFFECTING
    };
...
}

```

Refer to the IDL comments in Annex A for a description of these values.

8.7.3.1 Probable causes

Alarms have a mandatory record member probableCause of type `string` which defines further qualification as to the probable cause of the alarm in the sense of [ITU-T X.733] as stated above. [ITU-T X.733] and [ITU-T X.721] define 57 probable causes, [ITU-T M.3100] defines or reserves 207 probable causes that partly overlap with the X.733/X.721 definitions, and TM Forum's MTNM v3.0 defines 96 probable cause strings [TMF probableCause]. The service-oriented framework does not encode specific probable causes but any interface specification that extends this framework shall specify all probable causes which may occur for any managed object according to the following requirement (e.g., "AIS" for alarm indication signal, "BER_SD" for bit error rate signal degrade).

(R) NOTIF-1 Probable causes shall be specified as human-readable strings which contain only capital letters, underscores, hyphens (not dashes) and digits. The standard probable cause strings specified by MTNM v3.0 [TMF probableCause] shall be used as a starting point when an interface specification extends this framework, i.e., the extension shall only define new probable causes when no value specified by [TMF probableCause] fits the underlying requirements. When probable causes from [ITU-T X.733], [ITU-T X.721] or [ITU-T M.3100] are being used, they shall be translated into syntactically correct strings in the obvious fashion (e.g., leak detected/leakDetected into "LEAK_DETECTED").

Since probable causes are defined as strings, sets of probable causes are defined as lists of strings:

```

module notifications
{
...
    typedef sequence<string> ProbableCauseList_T;
...
}

```

This type may be used, for example, to include or exclude alarms with specified probable causes in or from result lists of on-demand pull event operations (see clause 8.7.5.4).

8.7.3.2 Perceived and assigned severities

The perceived severity levels provide an indication of how it is perceived that the capability of a managed object reporting an alarm or a TCA has been affected. Four levels represent service affecting (SA) or non-service affecting (NSA) conditions according to the determination by the managed system (see also the serviceAffecting attribute): as stated by [ITU-T X.733], usually critical and major are SA and require immediate or urgent corrective action, while usually minor and warning are NSA or potentially SA and require corrective or diagnostic action.

ALM and TCA notifications have a mandatory attribute perceivedSeverity of type `PerceivedSeverity_T` which holds the X.733-defined perceived severity level. In case of a TCA, this attribute mainly serves as the trigger flag, i.e., allows to distinguish between a trigger/raise TCA (value `PS_INDETERMINATE` or, optionally, some SA or NSA level) and a clear TCA (value `PS_CLEARED`). The IDL describing this data type is provided below (without comments).

```

module notifications
{
...
    enum PerceivedSeverity_T
    {
        PS_INDETERMINATE,
        PS_CRITICAL,
        PS_MAJOR,
        PS_MINOR,
        PS_WARNING,
        PS_CLEARED
    };
...
}

```

Refer to clause 8.1.2.3 of [ITU-T X.733] and Corrigendum 2 to [ITU-T X.733] for the semantics of the values.

Sets of perceived severities are defined as lists of `PerceivedSeverity_T`:

```

module notifications
{
...
    typedef sequence<PerceivedSeverity_T> PerceivedSeverityList_T;
...
}

```

This type may be used, for example, to include or exclude alarms and TCAs with specified perceived severities in or from result lists of on-demand pull event operations (see clause 8.7.5.4).

NOTE 1 – The perceived severity levels are the starting point for [b-ITU-T X.731], [ITU-T X.721] and [ITU-T M.3100] to define the alarm status of managed objects. This status is of interest when considering lifecycles of alarms and TCAs in the context of event acknowledgement. Refer to clause 8.7.5.3 for some details.

The relationship between probable causes and perceived severities (and service affection), as reported by alarms, is fundamental for (service-oriented) alarm management. Therefore, the advance assignment of severities to probable causes per managed object (having regard to service affection) is of interest. For this purpose, the concepts of assigned severity and alarm severity assignment (ASA) are introduced for SA, NSA and service-independent usage (as determined by the managed system). An ASA list provides a listing of all abnormal conditions that may exist in instances of a managed object class. The IDL describing these types is provided below (without comments).

```

module notifications
{
...
    enum AssignedSeverity_T
    {
        AS_INDETERMINATE,
        AS_CRITICAL,
        AS_MAJOR,
        AS_MINOR,
        AS_WARNING,
        AS_NONALARMED,
        AS_FREE_CHOICE
    };
...
    struct AlarmSeverityAssignment_T
    {
        string probableCause;
        string probableCauseQualifier;
        string nativeProbableCause;
        AssignedSeverity_T serviceAffecting;
        AssignedSeverity_T serviceNonAffecting;
        AssignedSeverity_T serviceIndependentOrUnknown;
    };
...
}

```