



INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

X.780

(01/2001)

SERIES X: DATA NETWORKS AND OPEN SYSTEM
COMMUNICATIONS

OSI management – Management functions and ODMA
functions

TMN guidelines for defining CORBA managed objects

ITU-T Recommendation X.780

(Formerly CCITT Recommendation)

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

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.399
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.899
OPEN DISTRIBUTED PROCESSING	X.900–X.999

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

TMN guidelines for defining CORBA managed objects

Summary

This Recommendation specifies guidelines for defining CORBA-based interfaces to software objects representing manageable resources in a TMN. It covers information modelling guidelines, rules for translating models from GDMO, and IDL style conventions. It also provides an IDL module defining data types, superclasses, and notifications to be used in CORBA-based information model specifications.

Source

ITU-T Recommendation X.780 was prepared by ITU-T Study Group 4 (2001-2004) and approved under the WTSA Resolution 1 procedure on 19 January 2001.

Keywords

Abstract Syntax Notation One (ASN.1), Common Object Request Broker Architecture (CORBA), Distributed Processing, Guidelines for the Definition of Managed Objects (GDMO), Interface Definition Language (IDL), Managed Objects, TMN Interfaces.

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.

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, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

© ITU 2001

All rights reserved. No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from ITU.

CONTENTS

Page

1	Scope.....	1
1.1	Purpose.....	1
1.2	Application.....	2
1.3	Roadmap	3
1.4	Conventions	4
1.5	Compiling the IDL.....	4
2	References.....	5
2.1	Normative References.....	5
3	Definitions and Abbreviations	5
3.1	Definitions from ITU-T X.701	5
3.2	Definitions from ITU-T X.703	5
3.3	Abbreviations.....	6
4	CORBA Modelling Goals and Requirements.....	7
4.1	Goals	7
4.1.1	Application Interoperability	7
4.1.2	Common Usage of CORBA Common Object Services	7
4.1.3	Information Model Transparency	7
4.2	Entities	8
4.2.1	Access Granularity	8
4.3	Principles of Containment and Naming.....	8
4.3.1	Naming	9
4.3.2	Entity Identification	9
4.4	Managed Object Classes	10
4.5	Packages.....	10
4.6	Attributes	10
4.6.1	GET and SET.....	10
4.6.2	Generic Attribute Get	10
4.6.3	Set-valued Attributes	10
4.7	Creation and Deletion of Managed Objects.....	10
4.7.1	Creation	11
4.7.2	Deletion	12
4.8	Inheritance	12
5	The Object Model IDL Module	12
5.1	The Base (Top) Managed Object Interface.....	13
5.1.1	The <i>nameGet()</i> Operation.....	14

	Page
5.1.2 The <i>objectClassGet()</i> Operation	14
5.1.3 The <i>packagesGet()</i> Operation	14
5.1.4 The <i>creationSourceGet()</i> Operation	14
5.1.5 The <i>deletePolicyGet()</i> Operation	14
5.1.6 The <i>attributesGet()</i> Operation	15
5.1.7 The <i>destroy()</i> Operation	15
5.2 The Managed Object Factory	16
5.3 The Notifications Interface	16
5.4 The Data Type Definitions	18
5.5 Exceptions	18
5.5.1 The <i>ApplicationError</i> Exception	18
5.5.2 The <i>CreateError</i> Exception	19
5.5.3 The <i>DeleteError</i> Exception	20
5.6 Macro Definitions	20
5.7 The Constant Definitions	21
6 Information Modelling Guidelines	21
6.1 Modules	21
6.2 Interfaces	21
6.3 Attributes	22
6.3.1 Readable Attributes	22
6.3.2 Settable Attributes	23
6.3.3 Set-valued Attributes	23
6.3.4 Exceptions	23
6.3.5 Standard Attributes	23
6.4 Actions	24
6.5 Notifications	25
6.6 Conditional Packages	25
6.7 Behaviour	26
6.8 Name Binding Information	26
6.9 Factories	28
6.9.1 Create Operations	28
6.9.2 Factory Finder	30
6.10 Managed Object Class Value Types	30
6.11 Constants	31
6.12 Registration	32
6.13 Versioning of CORBA/IDL Specifications	33
7 GDMO Translation	34

	Page
7.1 Managed Object Classes	34
7.2 Packages.....	34
7.3 Attributes	35
7.4 Attribute Groups	36
7.5 Actions	36
7.6 Notifications.....	36
7.7 Behaviours	37
7.8 Name Bindings	37
7.9 Parameters.....	38
7.9.1 ACTION-INFO and ACTION-REPLY.....	38
7.9.2 EVENT-INFO and EVENT-REPLY.....	39
7.9.3 Context-Keyword	40
7.9.4 SPECIFIC-ERROR	40
7.10 ASN.1 Data Types	41
7.10.1 Basic Types.....	41
7.10.2 Sequence.....	41
7.10.3 Sequence of.....	41
7.10.4 Set of.....	41
7.10.5 Choice.....	42
7.10.6 Object Identifier (OID).....	42
7.10.7 Object Instance	42
7.10.8 BIT STRING	42
8 Style Idioms for CORBA IDL Specifications	44
8.1 Use Consistent Indentation	44
8.2 Use Consistent Case for Identifiers	45
8.3 Follow JIDM Approach for IMPORT	45
8.4 Use JIDM Approach for OPTIONAL and CHOICE.....	45
8.5 Use a Consistent Type Suffix	46
8.6 Use a Consistent Suffix for Sequence Types.....	46
8.7 Use a Consistent Suffix for Set Types	46
8.8 Use a Consistent Suffix for Optional Types	46
8.9 Arrange Operation Parameters in a Consistent Manner	46
8.10 Assume No Global Identifier Spaces.....	46
8.11 Module Level Definitions	47
8.12 Use of Exceptions and Return Codes	47
8.13 Explicit vs. Implicit Operations	47
8.14 Don't Create a Large Number of Exceptions.....	47

	Page
9 Compliance and Conformance.....	47
9.1 Standards Document Compliance.....	47
9.2 System Conformance	48
9.3 Conformance Statement Guidelines	48
Annex A – The Object Model CORBA IDL Module.....	48
Annex B – Network Management Constant Definitions	70
Appendix I – Bibliography	73

TMN guidelines for defining CORBA managed 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. The initial TMN interface specifications for intra- and inter-TMN interfaces were developed using the Guidelines for the Definition of Managed objects (GDMO) notation from OSI Systems Management with Common Management Information Protocol (CMIP) as the protocol. The inter-TMN interface (X) included both CMIP and CORBA GIOP/IOP as possible choices at the application layer.

CORBA, a distributed processing technology, is being considered for use in the TMN communication architecture primarily due to its acceptance by the Information Technology industry. This acceptance is expected to enhance the availability of CORBA-based interfaces due to better development tools and wide-spread expertise in developing CORBA-based interfaces. This technology, developed by the Object Management Group (OMG), is also being considered by multiple industries. Specifications using this technology provide support for standard application programming interfaces (APIs) and language bindings to programming languages, and they also facilitate software portability. The interoperability solutions offered by the object request broker combined with the Inter-ORB protocol address interoperability between client and server. While CMIP and information models provide solutions for interoperability between manager and agent systems, CORBA defines inter-object interactions where the objects may be distributed.

1.1 Purpose

Several groups are developing network management specifications that use CORBA modelling techniques with IDL as the notation along with CORBA services. The scope of this Recommendation is to define guidelines suitable for use in the specification of interoperable CORBA-based network management interfaces. The demands placed on "X" interfaces are different from those used "inside" an administration, "Q" interfaces. The scope of this Recommendation covers all interfaces in the TMN where CORBA may be used. It is expected that not all capabilities and models 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) as well as between management systems and network elements.

ITU-T Q.816 [1] defines a set of services that are required for CORBA-based TMN interfaces. This Recommendation defines guidelines for specifying information models written in CORBA IDL to which the services are applicable. It also provides rules for translating existing GDMO models to IDL. Finally, it defines some base IDL code for use by all CORBA-based TMN information models. The combination of this Recommendation and ITU-T Q.816 form a *framework* for defining and implementing CORBA-based TMN interfaces.

Use of a common framework on telecommunications management interfaces has several advantages. Some examples are: facilitating reuse of models that are developed to meet the generic requirements of telecommunications; profiling CORBA services for use by the telecommunications industry; easing the definition of new services for TMN; reusing the semantics of the existing rich set of models; and harmonizing the modelling approach across groups using a single source similar to ITU-T X.720, X.721 and X.722 for CMIP. Reusing a common approach to modelling resources and reusing a generic information model for a variety of network technologies and network management applications will speed the introduction of new network services while keeping network management system development costs down.

The telecommunications industry has invested a great deal of time and energy in the development of information models for the CMIP network management protocol. A primary goal of the TMN CORBA framework is the reuse of these information models by enabling their translation to CORBA Interface Definition Language (IDL) with little change in semantics. As a result, initial IDL information models are expected to be derived from CMIP models.

1.2 Application

ITU-T M.3020 defines three phases in the development of a TMN specification. The three phases are Requirements, Analysis and Design. Figure 1 shows this process and the scope of this Recommendation for developing CORBA-based interface specification relative to this process.

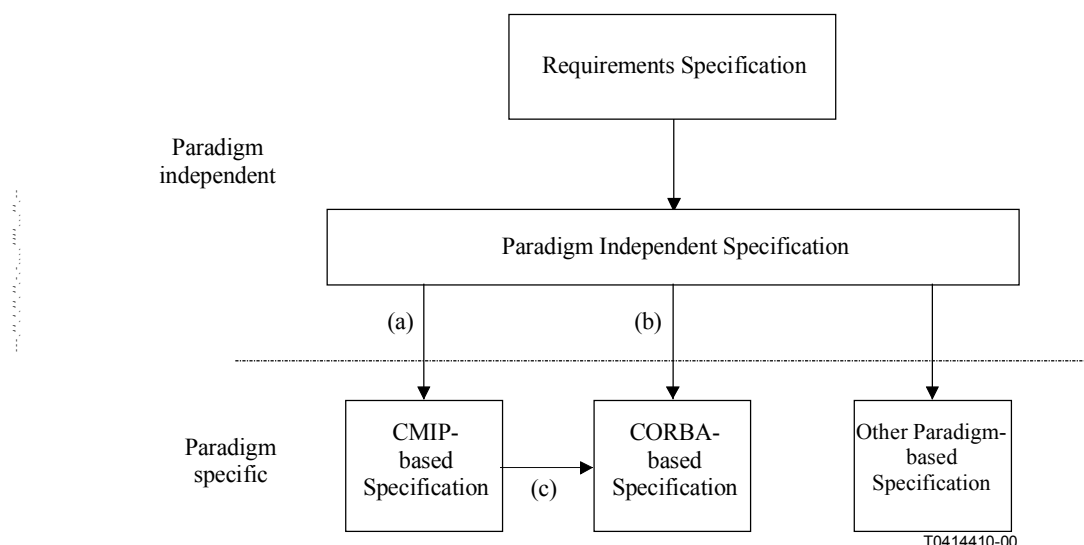


Figure 1/X.780 – CORBA-based Specification

The requirements and analysis are specified using an approach that is not specific to a network management technology paradigm. The output from the analysis phase, the paradigm independent specification, is used as input to the paradigm specific design phase.

In the design phase, network management paradigm specific features are used to define information models. These paradigm specific specifications incorporate both behaviour (normally in natural language) and formal interface signatures (e.g. GDMO, IDL).

The arrows marked as (a) and (b) show that the analysis output is mapped to GDMO/ASN.1 based model to use with CMIP or IDL models to use with CORBA/IIOP, respectively. There are no prescriptive rules available at this time to generate these models. It may be possible to develop such rules in the future in ITU-T M.3020.

This Recommendation addresses the reuse of existing models developed in the CMIP paradigm if CORBA/IIOP is to be used instead of CMIP. The arrow shown as (c) is addressed by this Recommendation.

In developing the transformation from GDMO/ASN.1 definitions to CORBA/IDL, two approaches are possible.

- In the first approach, every element of the syntax is translated to CORBA/IDL using a well-specified algorithm or a prescriptive definition. This approach is the one taken with Joint Inter-Domain Management (JIDM) where a gateway can be used to support interoperability.

- The second approach (used in this Recommendation) does not prescriptively translate every element of the syntax. Rather, the elements are translated from the existing GDMO in a way that preserves the semantics and also uses the features of CORBA. This approach is not used for inter-working via gateways but to preserve the requirements and semantics of the models developed to meet the telecommunication context. It is applied when the managing and managed systems are designed to communicate using CORBA/IIOP.

In addition to the recommendations for translating from GDMO information models defined here, ITU-T Q.816 defines recommendations for CORBA services to be used for managing telecommunications networks. Q.816 aspects of the framework are applicable irrespective of how CORBA based specifications are developed (i.e., using the path designated as (b) or (c) in Figure 1).

In addition to taking advantage of CMIP information models, another purpose of the guidelines is to take advantage of CORBA. The framework leverages the functions defined in the CORBA specifications, including a set of Common Object Services. Also, these guidelines reuse CORBA approaches and design patterns wherever they are appropriate. Finally, while reusing existing models is important, it is equally important that the framework support the development of new models. These guidelines do not require a GDMO model to be developed prior to the development of an IDL model. In fact, developing a new IDL information model for use within this framework is straightforward and guidelines for doing so are provided.

ITU-T M.3120 [11] provides a CORBA IDL version of the generic network information model originally defined in ITU-T M.3100. The IDL version follows the object modelling guidelines defined here and is designed to use CORBA-based TMN services defined in ITU-T Q.816.

1.3 Roadmap

This Recommendation has the following structure:

- Clause 1 Introduction, document roadmap, and updates.
- Clause 2 References.
- Clause 3 Definitions of abbreviations used throughout the rest of the Recommendation.
- Clause 4 Requirements for the object modelling guidelines. These are the design goals the guidelines must meet.
- Clause 5 Description of the CORBA IDL module that defines interfaces to be used and subclassed in network management interface specifications. The actual IDL is in Annexes A and B.
- Clause 6 Guidelines for defining CORBA-based TMN information models. These guidelines are specifically designed for IDL objects using the TMN CORBA-based services in ITU-T Q.816.
- Clause 7 Guidelines for translating GDMO information models to IDL models suitable for use with the TMN CORBA-based services in ITU-T Q.816.
- Clause 8 Style idioms for CORBA IDL network management interface specifications.
- Clause 9 Compliance and conformance guidelines.
- Annex A The IDL module for the modelling guidelines specification. This annex is normative.
- Annex B Additional IDL defining constants used by the modelling guidelines. This annex is normative.

1.4 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 likewise preceded by an "O" instead of an "R." For example:

(O) OPTION-1 An example optional requirement.

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

Many examples of CORBA IDL are included in this Recommendation, and IDL specifying the data types and base classes are included in normative annexes. The IDL is presented in a 9-point courier typeface:

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

Instructions for extracting the IDL from an electronic version of this Recommendation and compiling it are presented in clause 1.5.

1.5 Compiling the IDL

An advantage of using IDL to specify network management interfaces is that IDL can be "compiled" into programming code by tools that accompany an ORB. This actually automates the development of some of the code necessary to enable network management applications to interoperate. This Recommendation has two annexes that contain code that implementors will want to extract and compile. Both Annex A and Annex B are normative and should be used by developers implementing systems that conform with this Recommendation. The IDL in this Recommendation has been checked with two compilers to ensure its correctness. A compiler supporting the CORBA 2.3 specification must be used.

The annexes have been formatted to make it simple to cut and paste them into plain text files that may then be compiled. Below are tips on how to do this.

- 1) Cutting and pasting seems to work better from the Microsoft® Word® version of this Recommendation. Cutting and pasting from the Adobe® Acrobat® file format seems to include page headers and footers, which cannot be compiled.
- 2) All of Annex A, beginning with the line "/* This IDL code..." through the end should be stored in a file named "itut_x780.idl" in a directory where it will be found by the IDL compiler.
- 3) All of Annex B, beginning with the line "/* This IDL code..." through the end should be stored in a file named "itut_x780Const.idl" in the same directory as the file containing Annex A.
- 4) The headings embedded in these annexes need not be removed. They have been encapsulated in IDL comments and will be ignored by the compiler.

- 5) Comments that begin with the special sequence `/**` are recognized by compilers that convert IDL to HTML. These comments often have special formatting instructions for these compilers. Those that will be working with the IDL may want to generate HTML as the resulting HTML files have links that make for quick navigation through the files.

The annexes have been formatted with tab spaces at 8-space intervals and hard line feeds that should enable almost any text editor to work with the text.

2 References

2.1 Normative 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; all 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.

- [1] ITU-T Q.816 (2001), *CORBA-based TMN services*.
- [2] OMG Document formal/99-10-07, *The Common Object Request Broker: Architecture and Specification*, Revision 2.3.1.
- [3] OMG Document formal/2000-08-01, *CORBA/TMN Interworking, Version 1*, Edition 4.31.
- [4] ITU-T X.701 (1997) | ISO/IEC 10040:1998, *Information technology – Open Systems Interconnection – Systems management overview*.
- [5] ITU-T X.703 (1997) | ISO/IEC 13244:1998, *Information technology – Open Distributed Management Architecture*.
- [6] ITU-T X.721 (1992) | ISO/IEC 10165-2:1992, *Information Technology – Open Systems Interconnections – Structure of management information: Definition of management information*.
- [7] ITU-T X.722 (1992) | ISO/IEC 10165-4:1992, *Information Technology – Open Systems Interconnection – Structure of management information: Guidelines for the definition of managed objects*.

3 Definitions and Abbreviations

3.1 Definitions from ITU-T X.701

The following terms used in this Recommendation are defined in the Systems Management Overview (ITU-T X.701):

- managed object class;
- manager;
- agent.

3.2 Definitions from ITU-T X.703

The following term used in this Recommendation is defined in the Open Distributed Management Architecture (ITU-T X.703)

- notification.

3.3 Abbreviations

This Recommendation uses the following abbreviations:

ASN.1	Abstract Syntax Notation No. 1
ATM	Asynchronous Transfer Mode
CMIP	Common Management Information Protocol
CORBA	Common Object Request Broker Architecture
COS	Common Object Services
DN	Distinguished Name
EMS	Element Management System
GDMO	Guidelines for the Definition of Managed Objects
GIOP	General Interoperability Protocol
HTML	Hypertext Markup Language
ID	Identifier
IDL	Interface Definition Language
IIOP	Internet Interoperability Protocol
IOR	Interoperable Object Reference
ITU-T	International Telecommunication Union – Telecommunication Standardization Sector
JIDM	Joint Inter-Domain Management
MO	Managed Object
NE	Network Element
NMS	Network Management System
OAM&P	Operations, Administration, Maintenance, and Provisioning
OID	Object Identifier
OMG	Object Management Group
ORB	Object Request Broker
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
QoS	Quality of Service
RDN	Relative Distinguished Name
TMN	Telecommunications Management Network
TTP	Trail Termination Point
UID	Universal Identifier
UML	Unified Modelling Language
UTC	Universal Time Code

4 CORBA Modelling Goals and Requirements

This clause describes the key goals for modelling TMN resources using CORBA, and the requirements that the modelling guidelines must meet to support these goals. Clause 4.1 introduces the goals of the modelling guidelines. Subsequent subclauses then provide terminology and requirements. The requirements in clause 4 are requirements that the framework must satisfy. They are based on the telecommunications management needs. Clauses 5, 6, 7, and 8 then describe modelling guidelines that meet these needs and define how to achieve the requirements of clause 4 by using CORBA in a certain way. The rules in clauses 5, 6, 7, and 8 on how to use CORBA also are referred to as requirements.

4.1 Goals

This Recommendation specifies guidelines for defining CORBA managed objects for use on interfaces supported by telecommunications network management systems and network elements. Some key goals of the modelling guidelines are:

- Application Interoperability;
- Common Usage of CORBA Common Object Services;
- Information Model Transparency

This clause elaborates on these three goals.

4.1.1 Application Interoperability

A key goal of the TMN architecture, and in particular the information architecture, is to promote a standard framework for providing interoperability and information exchange between systems from a diverse set of network management system suppliers. Interoperability between systems involves many aspects of development. At its lowest layer, a common communication mechanism must be in place to support a common syntax, the establishment of connectivity and the exchange of operation requests/replies between systems. This aspect of interoperability is inherently supported by the CORBA specification.

For TMN, there is the need to provide application interoperability. That is, management systems from diverse suppliers will be utilized within a single administration's TMN to support different functions necessary to support management of its networks. To simplify integration of these various suppliers' systems, they must agree on the semantics of the information being exchanged. This is accomplished with the specification of an information model. This clause specifies the rules for defining these information models.

4.1.2 Common Usage of CORBA Common Object Services

A second aspect of these guidelines is the reliance upon a common usage and profiling of the distributed processing environment of choice. Rather than re-defining the interface capabilities needed to support common network management functions such as object naming and notification filtering with each information model, these guidelines rely upon a set of support services. These support services enable the information models to be simpler, and also enhance interoperability. The support services required for CORBA-based interfaces are specified in ITU-T Q.816.

4.1.3 Information Model Transparency

If CORBA is used in places within the TMN architecture where existing information models (e.g. GDMO) are well established, then the framework must support the reuse of those models without any major changes.

A single standard way to map these GDMO information models to OMG IDL is needed so that the same models are always presented by the application protocol to the application with the same set of services (capabilities).

4.2 Entities

An **entity type** describes a kind of "thing" in the real world. Each **entity type** has particular properties, called attributes.

An **entity instance** (or **entity**) (e.g. Circuit Pack #1) is of an **entity type** (e.g. Circuit Pack). Each **entity's** attributes have values that represent the state of that instance. In addition, each **entity** must be uniquely identifiable.

In CORBA, an **entity** may be accessed by different methods. For example, an **entity** may be accessed by an IDL data structure, a value type, or an interface type. This Recommendation presents how CORBA is utilized to define **entity types**.

4.2.1 Access Granularity

In the context of TMN operations, granularity defines the level of abstraction that is exposed between systems. Access Granularity identifies the level at which **entities** may be accessed (i.e. how information is exposed via an interface). For CORBA, each CORBA object is provided a unique address known as an Interoperable Object Reference (IOR). The IOR provides an address to the client system identifying which server system to connect to for communication with the server side CORBA object.

In CORBA, it is possible to define different access abstractions (i.e. access granularity) to the Entities defined for TMN (e.g. ITU-T M.3100). Two different access abstractions are defined here:

- 1) **Instance granularity:** Each entity has its own IOR. For the creation of new Entities, this implies the instantiation of a new CORBA object.

-1 IOR / entity instance

For example, an **entity type** in the ATM domain is an *atmLink*. In the Instance Grained approach, a CORBA object is defined that supports the same attributes as the entity type which it represents. For each instance of the *atmLink*, an independent CORBA object is created. Thus each *atmLink* can be uniquely addressed by its IOR.

- 2) **Application-specific granularity:** Instances of a well-defined set of entity types are accessed via a single IOR (a single interface).

-1 IOR / Family (set of) entity types

Bulk operations are defined in application-specific CORBA IDL interfaces, which pass identities and states of managed entities using operation parameters employing lists of IDL structured types.

The CORBA object modelling guidelines defined in this specification are applicable to the specification of managed object interfaces that support instance-grained access granularity. TMN standards may also be defined using application-specific access granularity. Such interface specifications, however, are outside the scope of this Recommendation.

4.3 Principles of Containment and Naming

Containment is a logical representation of how entities of one type contain entities of another type. A Containment Tree defines the relationship between the entity instances. An entity instance is contained by one and only one containing entity instance. Containing entity instances may themselves be contained in another entity instance forming a directed graph. The directed graph forms what is called the Naming (or Containment) Tree.

The containment relationship can be used to model real-world hierarchies of parts (e.g. assembly, sub-assemblies and components) or real-world organizational hierarchies (e.g. company name, org. name).

An example of a possible containment tree is shown in Figure 2 below.

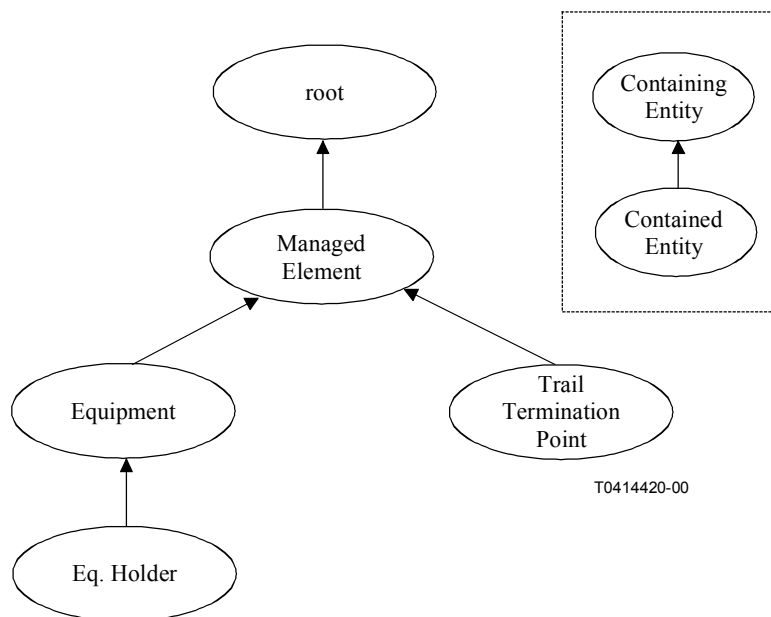


Figure 2/X.780 – Example of Containment

4.3.1 Naming

One purpose of containment relationships is for naming entities. Names are designed to be unique in a specified context; for TMN, this context is determined by the containing entity instance.

An entity that is named in the context of another entity is termed a "Subordinate Entity". The entity that establishes the naming context (this term is used in general and should not have the direct connotation of a COS Naming Service Naming Context) for other entities is called the "Superior Entity".

A "Subordinate Entity" is named by the combination of:

- The name of its "Superior Entity".
- Information uniquely identifying this "Subordinate Entity" within the scope of its superior entity.

The name of an entity that is unique in a local naming context may not be so in some larger naming context. However, if the local naming context is unique in the larger context, a local name can be made unique by qualifying it by its naming context; the name of the naming context is used as the qualifier. This arrangement can be visualized as a directed graph with each edge (or arrow) pointing from a named object to a naming context.

The naming context can itself be recursively qualified by another naming context, so the complete naming structure can be visualized as a single-rooted hierarchy. This hierarchy is called the naming tree. Thus "Superior Entities" become the naming contexts and their names become the names of the contexts. An object name need only be unique within the context of its superior Entities; within a wider context its name is always qualified by names of its superior Entities.

4.3.2 Entity Identification

Because a "Superior Entity" may contain multiple "Subordinate Entities" of the same type, each of these contained entities of the same type must be distinguishable relative to their containing entity. The relative name of an entity within its containing entity is called an entity's **Relative Distinguished Name** (RDN). For example, there may be several equipment holders within a managed element. To uniquely identify each equipment holder within the managed element, the equipment holders must be provided an RDN. The RDN should identify the name of the entity type

(e.g. equipment holder, which is an entity type) and a unique value within the scope of the containing entity.

An RDN is a basic element of a **Distinguished Name** (DN), as specified in ITU-T X.720. A DN is defined by a sequence of RDNs starting from a specific context. The DN yields a unique name relative to this context.

4.4 Managed Object Classes

These modelling guidelines specify that each entity type maps one-to-one with a CORBA operational interface. When an entity type is mapped in this manner, the CORBA object representing the entity type is called a *Managed Object Class*. A Managed Object Class must also exhibit the ability to emit notifications (see ITU-T X.703).

The term "Managed Object Class" is defined in ITU-T X.720. As explained in ITU-T X.703, managed object classes and sub-classes map to interfaces and derived interfaces.

4.5 Packages

It is necessary to capture the notion of packages in CORBA IDL. Packages are groups of capabilities (attributes, actions, or notifications) that may be conditionally supported by a Managed Object Instance. A managing system must have the capability to determine which packages are supported by a Managed Object Instance. If any operations are performed on a Managed Object, and those operations are contained by a Conditional Package that is not instantiated for that Managed Object, then the Managed Object must indicate an error.

4.6 Attributes

The guidelines must support the definition of attributes (i.e. visible properties) on Managed Object Classes.

4.6.1 GET and SET

The value of an attribute may be observable or modifiable across a standard interface. If observable, the information modeler must define a "get" method for that attribute. If modifiable, the information modeler must define a "set" method for that attribute.

4.6.2 Generic Attribute Get

CORBA-based TMN information models should allow a managing system the ability to read arbitrary groups of attributes from a single managed object with a single operation. This service allows many management tasks to be performed with a single operation. Support of the Generic Attribute Get is required.

4.6.3 Set-valued Attributes

For attributes containing lists of values, a modeler should have the capability to allow managing systems to add or remove individual values to/from lists without resending all the information in the original list.

4.7 Creation and Deletion of Managed Objects

The existence of Managed Objects (MOs) is closely related to the containment relationship between the MOs. A MO's existence is tied to the existence of that MO's superior MO Instance. If the specified "Superior MO" does not exist for a "Subordinate MO", then that "Subordinate MO" cannot be created. Similarly, if a MO's "Superior MO" is deleted, then that "Subordinate MO" (and the "Subordinate MO's" subordinates) can no longer exist. Given this, there are creation and deletion semantics that must be enforced by the TMN CORBA framework.

The following subclauses define the high-level requirements that must be supported for object creation and deletion. ITU-T Q.816 describes the generic services used to carry out creation (i.e. the factory) and deletion (i.e. the factory in coordination with the terminator service). Clause 6 defines modelling guidelines for how the requirements defined in this clause are supported.

4.7.1 Creation

When creating a Managed Object, three aspects of the MO's existence must be identified:

- The MO's name;
- The MO's attribute values;
- The conditional packages of the MO that are to be instantiated with the creation of the new MO.

Note that definition of these aspects in the create request may be either explicit or implicit. Options for identifying these aspects of a MO's existence are defined in the following three clauses.

4.7.1.1 Identification of the MO Name

The name of the MO to be created can be determined in one of two ways:

- 1) The manager may specify, as a parameter of the create operation, a reference to an existing MO which is to be the superior of the new MO and may specify the RDN of the new MO in the create operation's attribute list. This results in the complete specification of the MO name being supplied by the manager.
- 2) The manager may specify, as a parameter of the creation operation, a reference to an existing MO which is to be the superior of the new MO and may omit specifying the RDN of the new MO. In this case, the RDN of the new MO is assigned by the managed system.

If the associated information is not correct, or, for some other reason the create operation cannot be performed, then the factory attempting to perform the operation shall indicate an error.

4.7.1.2 Identification of the MO Attributes

When a MO is created, its attributes are assigned values that are valid for the type of attribute. These values are derived from information in the Create operation and the MO class definition in one of the two manners listed below:

- 1) The create request is permitted to specify an explicit value for each individual attribute. When the MO is created, explicit values are assigned to attributes as required by the MO class definition.
- 2) The MO class definition is permitted to specify how default values are assigned to attributes that are not set by the create operation.

If default values are not specified for an attribute, then the managing system must supply a value for that attribute in the create request. If no value is specified for that attribute, then an error should occur.

If an explicit value is defined for a particular attribute in the create request, then the MO will take that value for the specified attribute over any potential default value that may be specified for that attribute.

4.7.1.3 Identification of MO Packages for Instantiation

To ensure that underlying resources can be instantiated with required capabilities, the manager must be able to specify the capabilities (i.e. the conditional packages) that the managed object should have instantiated.

Instantiation of a conditional package will occur if an associated condition is satisfied for the managed object being instantiated. The manager may also request the instantiation of a conditional package as part of the create request, by including it in the packages attribute of the create request.

4.7.2 Deletion

For deletion, deletion semantics may support the deletion of all contained entities while in other cases, the delete method immediately fails if there are contained subordinate entities. These semantics must be maintained for each entity type.

4.8 Inheritance

One "Managed Object Class" may be defined as a specialization of another "Managed Object Class" by utilizing inheritance. Specialization of a "Managed Object Class" implies that all methods and attributes defined on the superclass will also be supported by the subclass.

In CORBA IDL, an attribute or operation cannot be inherited from more than one interface, nor can an inherited operation or attribute be redefined by a subclass. (Note that, in general, it is not expected that a CORBA information model would define a method or attribute in a class, where that same method or attribute may also be defined in the superclass. However, there are cases in the mapping from GDMO to IDL where this may occur. For example, because GDMO attributes specify permitted and required values, a subclass in GDMO may sometimes redefine the same attribute. Care must be taken when mapping to IDL that the same attribute is not redefined.)

A subclass in CORBA cannot inherit the same attribute or method (with the same name) from more than one superclass (unless they in turn inherited it from the same base class). Also, a subclass cannot redefine the same attribute or method (with the same name) defined in one of its superclasses.

These guidelines place no constraints over CORBA inheritance.

5 The Object Model IDL Module

Before describing the rules for defining TMN managed objects using CORBA Interface Definition Language (IDL), [2] this clause presents a network management module containing a set of object interfaces and supporting data structures specified in CORBA IDL. This IDL module is intended to play a role in CORBA-based network management similar to that played by the GDMO and ASN.1 definitions in ITU-T X.721 [6] for CMIP. It provides the basic set of IDL definitions on which information models are then built.

The IDL is included in Annexes A and B of this Recommendation. Annex A contains the base classes (interfaces), data structures, and notifications. Annex B is a separate file containing just constant definitions. Both of these are based on the GDMO and ASN.1 definitions found in ITU-T X.721.

ITU-T X.721 is a convenient source for capabilities that must be provided in network management information models. ITU-T X.721 defines the following managed object classes using GDMO:

- 9 types of records (Log Record, Event Log Record, Alarm Record, Attribute Value Change Record, Object Creation Record, Object Deletion Record, Relationship Record, Security Alarm Report Record, State Change Record);
- Discriminator and Event Forwarding Discriminator;
- Log;
- System;
- Top.

Each of these has attributes, actions, and supporting data types and parameters. In addition, ITU-T X.721 defines 15 notifications.

Looking at the managed object classes listed above, it is clear that many of these are covered by the CORBA Common Object Services already included in the framework (see ITU-T Q.816 for details on the TMN CORBA Based TMN Services):

- The CORBA Telecom Event Log service defines a structure for holding log records, so the record classes need not be redefined. (Note that by specifying the use of the CORBA Telecom Event Log Service the TMN CORBA framework treats log records as data structures, not objects.)
- The CORBA Notification Service defines a filtering capability, so the discriminator and event forwarding discriminator need not be redefined.
- The CORBA Telecom Event Log Service defines the equivalent of X.721's Log.

That leaves just System and Top, along with the notifications. System is not really a framework class and belongs instead in a generic information model (if it is needed). The IDL in Annex A, therefore, defines a "top" managed object interface, called "Managed Object," that is intended to be subclassed by all other managed object interfaces similar to the way the managed object class named "Top" is subclassed by all CMIP managed object classes. Also included is a generic "factory" object. Managed object factories are used for object creation. (The CORBA based TMN services defined in ITU-T Q.816 includes a *Terminator* service that handles object deletions independent of object type, but object creation is handled by class-specific factories so that object creation operations may be strongly typed.) The notifications are defined on a third IDL interface. In addition, a number of IDL data types are defined. Finally, some IDL pre-compiler macros are defined to ease managed object interface specification. Each of these is discussed below.

5.1 The Base (Top) Managed Object Interface

The first interface defined in Annex A is the *ManagedObject* interface, found after all the data type definitions. It is intended to be the base managed object interface from which all other interfaces inherit. It defines a set of capabilities that all managed object instances must support. These capabilities are:

- A method that returns the name of the object.
- A method that returns the interface (actual class) name of the object.
- A method that returns the conditional packages supported by the object instance.
- A method that returns the creation source of the object (whether it was created autonomously by the managed resource, in response to a management operation, or unknown).
- A method that returns the delete policy for the instance. This is an enumerated value and indicates if the object is not deletable, if it is deletable only if it contains no objects, or if all contained objects will be deleted when it is deleted.
- A method that returns a CORBA value type object containing all of the readable attributes for the object.
- A destroy operation.

The IDL describing the *ManagedObject* interface (without comments) is:

```
interface ManagedObject {
    NameType nameGet()
        raises (ApplicationError);
    ObjectClassType objectClassGet()
        raises (ApplicationError);
    StringSetType packagesGet()
        raises (ApplicationError);
    SourceIndicatorType creationSourceGet()
        raises (ApplicationError);
}
```

```

DeletePolicyType deletePolicyGet()
    raises (ApplicationError);
ManagedObjectValueType attributesGet (
    inout StringSetType attributeNames)
    raises (ApplicationError);
void destroy()
    raises (ApplicationError, DeleteError);
}; // end of ManagedObject interface

```

5.1.1 The *nameGet()* Operation

The first operation, *nameGet()*, returns the CORBA name of the object. *NameType* is a type definition for the CORBA Naming Service *Name* type. *NameType* is used to conform to the IDL conventions defined later in this Recommendation. This method returns the compound name of the object, beginning with the name assigned to the local root naming context under which the object is contained. That is, the method returns the "globally unique" name for the object. See ITU-T Q.816 for details on assigning a unique name to the root naming context of a managed system. The *ApplicationError* exception is defined to be raised by any managed object operation if the operation cannot be completed due to some resource problem. See clause 5.5 for details on this and all the other exceptions.

5.1.2 The *objectClassGet()* Operation

The *objectClassGet()* operation returns the scoped interface name (actual class name) of the object. Scoped interface names include the name(s) of the module(s) in which the interface is defined. The return value type, *ObjectClassType*, is a type definition for string. If the object's class is a minor extension of another class (e.g. an "R1" class), the string returned is the name of the actual class (with the "R1"). For example, "EquipmentR1".

5.1.3 The *packagesGet()* Operation

The *packagesGet()* operation returns the list of conditional packages supported by an object instance. The notion of conditional packages, each with a string name, is supported by these guidelines. See clause 6.6 for details. *StringSetType* is a type definition for a list of strings.

Note that this differs slightly from the *packages* attribute on CMIP objects because this framework does not support the definition of mandatory packages, only conditional. In CMIP it is possible for the *packages* attribute to list mandatory packages. Obviously, since the definition of mandatory packages is not supported by this framework, they can't be listed in the *packages* attribute of a managed object.

5.1.4 The *creationSourceGet()* Operation

The *creationSourceGet()* operation returns a value indicating the system that caused the object to be created. *SourceIndicatorType* is an enumerated type with three values: *resourceOperation*, *managementOperation*, and *unknown*. It indicates if the object was created autonomously by the resource in response to a management operation or, if it is unknown, why the object was created.

5.1.5 The *deletePolicyGet()* Operation

The *deletePolicyGet()* operation returns the delete policy for this object instance. This is an enumerated value that indicates if the object is not deletable, if it is deletable only if it contains no objects, or if all contained objects will be deleted when it is deleted. (Deleting an object but not its contained objects is not allowed.) This policy is set when the object is created by its factory based on the name binding information identified in the create operation.

5.1.6 The *attributesGet()* Operation

The *attributesGet()* method is used to return all, or any subset, of an object's attribute values in one operation. For each managed object interface in an information model, a CORBA *valuetype* containing data members for each of the readable attributes on that interface will be defined. (Readable attributes are those with an *<attribute name>Get()* operation.) This method may be used to retrieve this value type for any managed object. The value types will be defined following the inheritance hierarchy of the managed object interfaces (except that value types cannot support multiple inheritance), and each will ultimately be derived from the *ManagedObjectValueType* defined for the *ManagedObject* interface. The managed object must return a value type defined for its interface in response to this method. Thus, when a client invokes the *attributesGet()* operation on any managed object, it will receive back a reference to a *ManagedObjectValueType* which it may then narrow (cast) to the value type defined for the interface on which the operation was invoked.

Complicating this somewhat are the concerns that a client may not want to retrieve all of the attribute values from an instance, and an instance may not support all of the attributes that are in conditional packages. (The value types include attributes in conditional packages.) This is accommodated through the use of the in/out *attributeNames* parameter. On invocation, the client may submit a list of the names of the attributes in which it is interested, with a null list having the special meaning that all supported attributes should be returned. Any names on the list that are not valid attribute names should be ignored by the managed object. In its response, the object will return the actual list of attributes for which values are supplied. Note that this list may not match the submitted list. The object must always return an accurate list, even if the submitted list was null or had invalid names. If all the names on the submitted list are invalid, the object should return a null list and an empty value type.

Because the structure of the value type is pre-defined, the object must fill in some value for the attributes not requested or not supported. Basically, the object may return any values for these attributes, but the values should be as short as possible for efficiency. Thus, null values should be returned for strings, references, and lists of any kind. Any value may be returned for integers and enumerated types. The client must consider any value for an attribute not named in the list returned by the object to be invalid.

The base interface *ManagedObject* currently only has a method that returns a CORBA value type containing all of the readable attributes for the object. It does not contain a similar method for setting the attributes because not all attributes are settable.

5.1.7 The *destroy()* Operation

The final operation on the object, the *destroy()* operation, is used to release any resources associated with the managed object and to delete it. The *DeleteError* exception is raised by the object if it has a delete policy of *NotDeletable*. The *DeleteError* exception is also an extensible means of reporting problems destroying an object that are model-dependent. For example, trying to delete a Trail Termination Point object before the Trail is deleted might result in a *DeleteError*. ITU-T Q.816 defines a service called the "Terminator Service," however, to implement the logic needed to enforce delete policies and to maintain the integrity of the naming tree. The destroy operation is actually intended to be used by this service, and should not be directly invoked by a managing system. See ITU-T Q.816 for details on the Terminator Service.

(R) OBJECT-1. The interfaces used to model resources on a managed system shall inherit (directly or indirectly) from the *ManagedObject* interface described above and defined in the CORBA IDL in Annex A. The capabilities described above shall be supported.

5.2 The Managed Object Factory

Sometimes managed objects are created automatically by the managed system, sometimes they are created as a result of an action on another object (such as a cross-connection object created in response to a connect action on a fabric), and sometimes they are created in response to a request from a manager to create an object. In this last case, on CMIP systems, the create operation is typically handled by the CMIP agent framework. It can't be handled by the object itself because it hasn't been created yet. In CORBA implementations there is no agent framework, so something needs to be present on the managed system to enable the managing system to create objects. In CORBA systems this is often handled by "factory" objects. The *ManagedObjectFactory* interface is intended to be the base interface from which other factory interfaces inherit. It will define capabilities that all managed object factories are expected to support. Currently, no such capabilities have been identified, so the interface is null (inherits from nothing and has no attributes or methods). It is a placeholder in which capabilities may be placed in the future if needed. It also serves as a common superclass for all factories.

CORBA IDL information models are expected to include a factory interface per managed object interface (unless the managed object class is not instantiable). The factories will contain operations for creating managed objects. These operations will take a number of parameters, such as the new object's superior object, the new object's name, and values for each of the writeable or set-by-create attributes, etc. Upon successful creation of the new object, the factory will return a reference to it.

In addition to creating objects, it is expected that factories will also create name bindings in the CORBA Naming Service for the new objects. Though this functionality could be implemented elsewhere, it is believed that implementing it in the factories will simplify implementations by relieving the managed object implementation from this task, leaving them to focus on representing resources. See ITU-T Q.816 for details on how the TMN CORBA framework makes use of the CORBA Naming Service.

To help clients find factories, ITU-T Q.816 defines a Factory Finder Service. This service acts as a broker between clients and factories. Basically, factories register themselves with the service, then clients query the well-known service to find a factory of a particular type. See ITU-T Q.816 for details on the Factory Finder Service.

(R) FACTORY-1. The factory objects used to create managed objects on a managed system shall inherit (either directly or indirectly) from the *ManagedObjectFactory* interface described above and defined in the CORBA IDL in Annex A.

(R) FACTORY-2. All factories shall be registered in the Factory Finder object(s) instantiated on that system.

5.3 The Notifications Interface

The third interface defined in Annex A is the notifications interface. Each of the notifications in ITU-T X.721 has a corresponding operation on this interface. The notifications are defined as typed method calls as required by ITU-T Q.816. The OMG Notification Service is used to filter and broadcast notifications. The typed notification methods can be used directly with a notification service that supports typed notifications. Mappings between these typed event methods and structured events are provided in ITU-T Q.816.

All of the notification operations defined in this interface pass a number of parameters, some of which are common to all of the notifications. Several of the notifications have identical parameters, but are used for slightly different reasons. The notifications interface IDL looks like this:

```
interface Notifications {
    void equipmentAlarm (
        in ExternalTimeType    eventTime,
        in NameType             source,
        in ObjectClassType      sourceClass,
```

```

        in NotifIDType
        in CorrelatedNotificationSetType
        in AdditionalTextType
        in AdditionalInformationSetType
        in ProbableCauseType
        in SpecificProblemSetType
        in PerceivedSeverityType
        in BooleanTypeOpt
        in NameType
        in TrendIndicationTypeOpt
        in ThresholdInfoType
        in AttributeChangeSetType
        in AttributeSetType
        in ProposedRepairActionSetType
        in BooleanTypeOpt
        in BooleanTypeOpt
        in SuspectObjectSetType
    );

    ...
}; // end of Notifications interface

```

The other fourteen notification operations are similar to the one above. The names of the 15 notifications defined are:

- | | |
|--------------------------|----------------------------|
| • Attribute Value Change | • Physical Violation |
| • Communications Alarm | • Processing Error Alarm |
| • Environmental Alarm | • Quality of Service Alarm |
| • Equipment Alarm | • Relationship Change |
| • Integrity Violation | • Security Violation |
| • Object Creation | • State Change |
| • Object Deletion | • Time Domain Violation |
| • Operational Violation | |

This CORBA Framework requires the use of notification identifiers where they may not be required in other interfaces (they are not required in ITU-T X.733). To illustrate, below are four possible cases where the mapping of alarm notification identifiers from the network element/EMS interface to the EMS/NMS interface must be done:

- 1) The network element always uses notification identifiers and the managed object is represented in both interfaces. In this case, the EMS passes the alarm (with its notification identifier) on to the NMS.
- 2) The network element never uses notification identifiers and the managed object is represented in both interfaces. In this case, the EMS uses an internal counter, includes this value as the notification identifier and passes the alarm onto the NMS.
- 3) The network element sometimes uses notification identifiers and the managed object is represented in both interfaces. Because the notification identifier is required, the EMS must define a value when one is not provided. It may be difficult to define a value at the EMS because notification identifier values must be unique across all notifications of a particular managed object instance throughout the time that correlation is significant [1]. Thus, the EMS must choose a value that is not being used in current alarms and will not be used in subsequent alarms. Extra care must be taken when doing this, since the algorithm for choosing notification identifier values is owned by the producing system (in this case, the network element).

In one possible solution, the EMS could supply its own value for notification identifier for all alarms. This would also require the updating of each alarm's correlated notification lists, resulting in the EMS maintaining a complete mapping of network element Notification Identifier values to EMS Notification Identifier values.