INTERNATIONAL TELECOMMUNICATION UNION

ITU-T X.780.1

(08/2001)

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

SERIES X: DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS

OSI management – Management functions and ODMA functions

TMN guidelines for defining coarse-grained CORBA managed object interfaces

ITU-T Recommendation X.780.1

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.

ITU-T Recommendation X.780.1

TMN guidelines for defining coarse-grained CORBA managed object interfaces

Summary

This Recommendation defines extensions to the set of TMN CORBA managed object modelling guidelines required to support coarse-grained interfaces. It specifies how coarse-grained CORBA TMN interfaces are to be defined. It also provides guidelines on converting fine-grained interfaces to coarse-grained. A CORBA IDL module defining the base interface types to be extended is provided.

Source

ITU-T Recommendation X.780.1 was prepared by ITU-T Study Group 4 (2001-2004) and approved under the WTSA Resolution 1 procedure on 13 August 2001.

Keywords

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.

CONTENTS

ITU-T Recommendation X.780.1

TMN guidelines for defining coarse-grained CORBA managed object interfaces

1 Scope

The TMN architecture defined in ITU-T M.3010 (2000) introduces concepts from distributed processing and includes the use of multiple management protocols. ITU-T Q.816 and X.780 subsequently define within this architecture a framework for applying the Common Object Request Broker Architecture (CORBA) as one of the TMN management protocols.

This Recommendation, along with ITU-T Q.816.1, adds specifications to the framework to enable it to support a slightly different style of interaction between managing systems and managed systems than that specified in the original framework documents. This style of interaction has certain benefits, the main one being that it relieves a managing system from having to retrieve an object-oriented software address for each manageable resource it wishes to access. These software addresses could number in the millions on large systems. It also changes somewhat the way software is structured on the managed systems, which some managed system suppliers may prefer.

The scope of this Recommendation is the same as the original TMN CORBA framework. The framework and these extensions cover all interfaces in the TMN where CORBA may be used. 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) as well as between management systems and network elements.

1.1 Purpose

The purpose of this Recommendation is to extend the TMN CORBA framework to enable it to be used in a wider range of applications. The extensions enable a slightly different mode of interaction between the managing and managed systems which may be preferred in many situations. Thus, this Recommendation is intended for use by various groups specifying network management interfaces.

1.2 Application

The approach taken in the CORBA TMN framework Recommendations is to model manageable network resources as software objects accessible using CORBA. Information models written in the CORBA Interface Definition Language (IDL) describe the object interfaces.

CORBA provides location-transparency, enabling one software object to interact with another regardless of its location. A software object is accessed using what CORBA refers to as an Interoperable Object Reference (IOR).

The original CORBA TMN framework models each manageable resource as an independent CORBA object, each with its own unique IOR. This approach flexibly allows each object to reside anywhere. It does, however, require that managing systems have on hand an IOR for each object they wish to access. This is a burden that many companies and administrations in the telecommunications industry have sought to avoid. It also could require a managed system to support large numbers of IORs, which some managed system suppliers would like to avoid. This Recommendation, along with ITU-T Q.816.1, defines how the TMN CORBA framework is to be extended to avoid the need for large numbers of IORs.

CORBA-based interfaces using the approach where each manageable resource is addressable with a unique IOR have become known as "fine-grained" interfaces. Alternatively, those where an IOR is not assigned to each manageable resource are known as "coarse-grained" interfaces.

Because this Recommendation defines a slightly different approach to modelling manageable resources on coarse-grained interfaces, interface model specifications will be slightly different for the fine-grained and coarse-grained approaches.

1.3 Recommendation roadmap

This Recommendation has the following structure:

Clause 1 Introduction, roadmap and updates.

Clause 2 References.

Clauses 3 and 4 Definitions and abbreviations used throughout this Recommendation.

Clause 5 Conventions.

Clause 6 Design considerations that must be addressed as support for coarse-grained interfaces is added to the framework.

Clause 7 TMN CORBA framework and coarse-grained requirements overview.

Clause 8 Providing facade interfaces for accessing managed objects. This clause covers the model-specific interfaces that must be implemented on coarse-grained interfaces.

Clause 9 Guidelines for defining coarse-grained CORBA interfaces.

Clause 10 Guidelines for translating fine-grained CORBA interface specifications to coarse-grained interface specifications.

Clause 11 Compliance and conformance guidelines.

Annex A The IDL module for the coarse-grained modelling guidelines specification. This annex is normative.

2 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; 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 X.780 (2001), *TMN guidelines for defining CORBA managed objects*.

[2] ITU-T Q.816 (2001), *CORBA-based TMN services*.

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

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

3 Definitions

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 terms used in this Recommendation are defined in the Open Distributed Management Architecture (ITU-T X.703):

– notification.

3.3 Additional definitions

3.3.1 facade: An object interface defined to provide access to a set of managed objects, all of which are of the same class.

4 Abbreviations

This Recommendation uses the following abbreviations:

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
ID	Identifier
IDL	Interface Definition Language
IIOP	Internet Interoperability Protocol
IOR	Interoperable Object Reference
ITU-T	International Telecommunication Union – Telecommunication Standardization Sector
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

POA	Portable Object Adapter
QoS	Quality of Service
RDN	Relative Distinguished Name
TMN	Telecommunications Management Network
UID	Universal Identifier
UML	Unified Modelling Language
UTC	Universal Time Coordinated

5 Conventions

5.1 Conventions

A few conventions are followed in this Recommendation to make the reader aware of the purpose of the text. While most of this 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) 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 TMN specific services, and supporting data types, are included in Annex A. The IDL is written in a 9-point courier typeface:

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

5.2 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. Annex A contains code that implementers will want to extract and compile. Annex A is normative and should be used by developers implementing systems that conform to this Recommendation. The IDL in this Recommendation has been checked with two compilers to ensure its correctness. A compiler supporting the CORBA version specified in ITU-T Q.816 must be used.

Annex A has been formatted to make it simple to cut and paste 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_1.idl" in a directory where it will be found by the IDL compiler.

3) The headings embedded in Annex A need not be removed. They have been encapsulated in IDL comments and will be ignored by the compiler.

4) 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.

5) Annex A has 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.

6 Coarse-grained interface design considerations

This clause identifies several design considerations that must be addressed by the framework as support for coarse-grained interfaces is added.

6.1 Coarse-grained object creation and deletion

It must be possible to create and delete coarse-grained representations of managed resources. The possibility of including the create operation on the coarse-grained interface should be investigated.

6.2 Attributes

The framework must support associating attributes with managed resources accessed through a coarse-grained interface.

6.3 Notifications

The framework must support event notifications from managed resources accessed through coarse-grained interfaces.

6.4 Coarse-grained access to all managed resources

The framework must enable and require implementations to allow managing systems to access all managed resources through coarse-grained interfaces.

6.5 Exceptions

Coarse-grained interfaces shall enable managed resources to raise an exception on the invocation of an operation. These exceptions must be explicitly clarified for each operation.

6.6 Support for all operations

A coarse-grained interface shall support all of the operations applicable to a managed resource.

6.7 Prescriptive mapping

The mapping between fine-grained information models and coarse-grained information models shall be prescriptive. The mapping shall be capable of being performed algorithmically. If an IDL information model is developed by translating a GDMO information model, any optimizations performed by hand during the translation shall appear in both the fine-grained and coarse-grained models.

6.8 Retrieval of attributes from multiple objects

There is a need to retrieve attributes of multiple managed object instances of the same type in a single, strongly-typed operation. Since all of these objects would be accessed using the same facade, this can be accomplished with a single operation carried out by the facade.

7 Framework and requirements overview

Clause 6 outlined the design considerations that must be resolved as support for coarse-grained interfaces is added the framework. This clause and the rest of this Recommendation provide the details on how the framework will be extended to address these issues. ITU-T Q.816.1 focuses on the framework support services for coarse-grained interfaces, while this Recommendation defines guidelines for developing information models for coarse-grained interfaces. First, a brief overview of the current framework is presented, then an overview of the extensions.

7.1 Framework overview

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

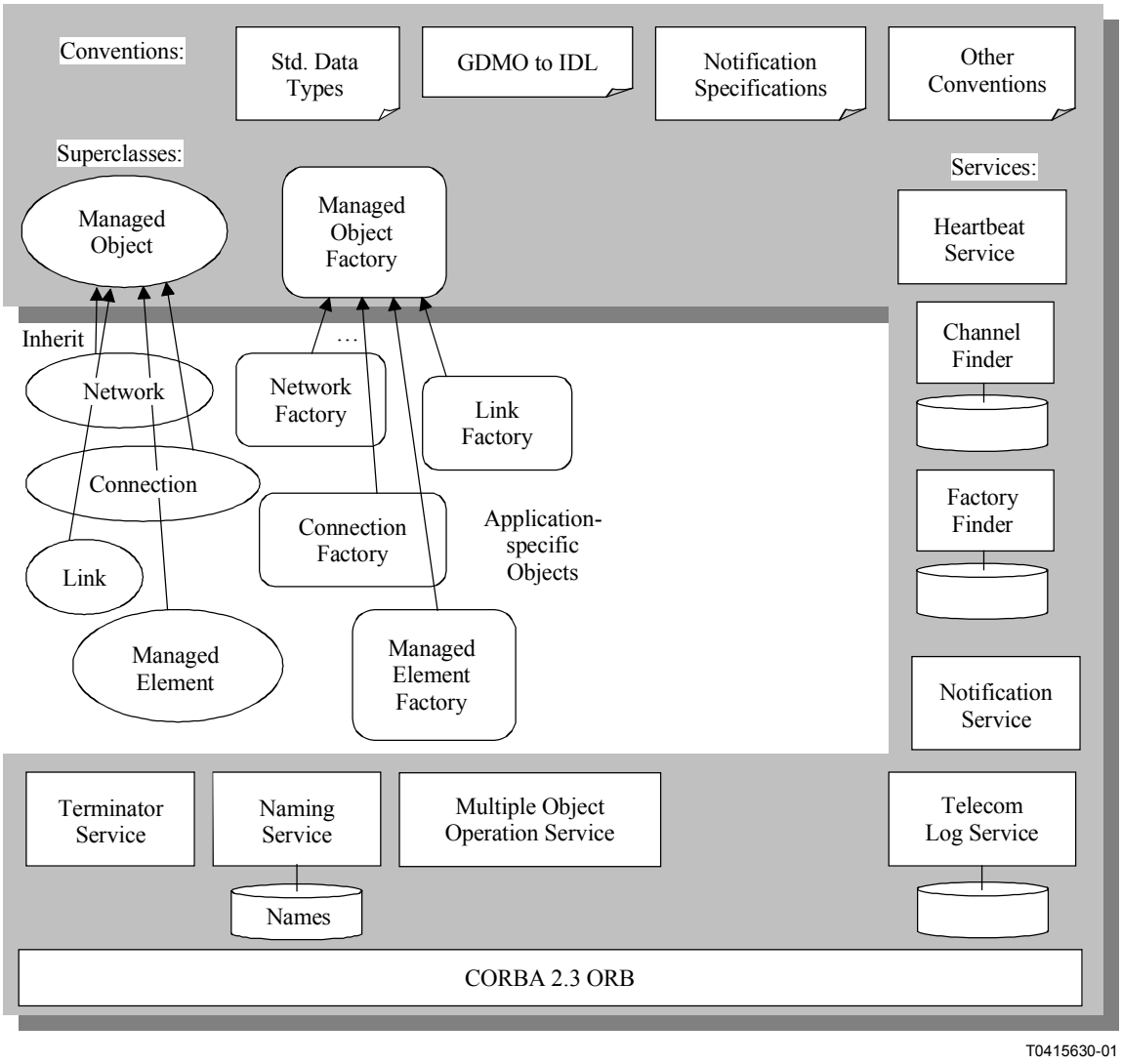

Figure 1/X.780.1 – Overview of framework

The framework is depicted graphically in Figure 1. The figure shows the framework in gray. In the middle are the application-specific objects that are supported by the framework. Along the bottom is a box representing the CORBA ORB. Above that are a number of boxes with names in them representing the services that compose the framework. (Some also have icons depicting the databases they would have to maintain to perform their functions.) These services, along with ORB version requirements, are defined in ITU-T Q.816. Along the top of the figure are icons representing two superclasses, one for managed objects and one for managed object factories. Each of the managed objects and managed object factories supported by this framework must ultimately inherit from these superclasses, respectively. Also shown on the figure are icons of pages with up-turned corners representing standard object modelling conventions. These conventions and the superclasses are defined in ITU-T X.780.

7.2 Coarse-grained extensions overview

This clause provides an overview of the extensions to the framework required to support coarse-grained interfaces.

7.2.1 The facade design pattern

The most significant change to the framework required to support coarse-grained interfaces is the way managed objects are accessed. The number of managed objects on a managed system must be able to go up while the number of IORs supported by the system does not. It is still desirable, though, that access to the managed objects remain strongly-typed. This leads to the use of a design pattern referred to here as the "facade" pattern. A facade can be thought of as a false front, or as a portal. Using the facade design pattern, a managed system will support a small number of facade 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 invoke an operation on a managed object by actually invoking the operation on a facade for that type of managed object on that system. In the facade 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 facade approach does not need to implement the fine-grained managed object interfaces.

It is best to think of a facade not as a managed object, but as an intermediary object that enables a managing system to access managed objects. The facade object has a CORBA interface and is accessible using CORBA. The managed objects, however, may not have CORBA interfaces and might not be directly accessible using CORBA. The facade itself does not represent a manageable network resource; its purpose is to enable interaction with the objects that do represent manageable resources. All facade objects are created automatically by the managed system, and exist as long as the managed objects are accessible through the facade. Multiple facades for the same type of managed objects may exist on a coarse-grained interface, but a managed object shall be accessible through only 1 facade. See Figure 2 below.

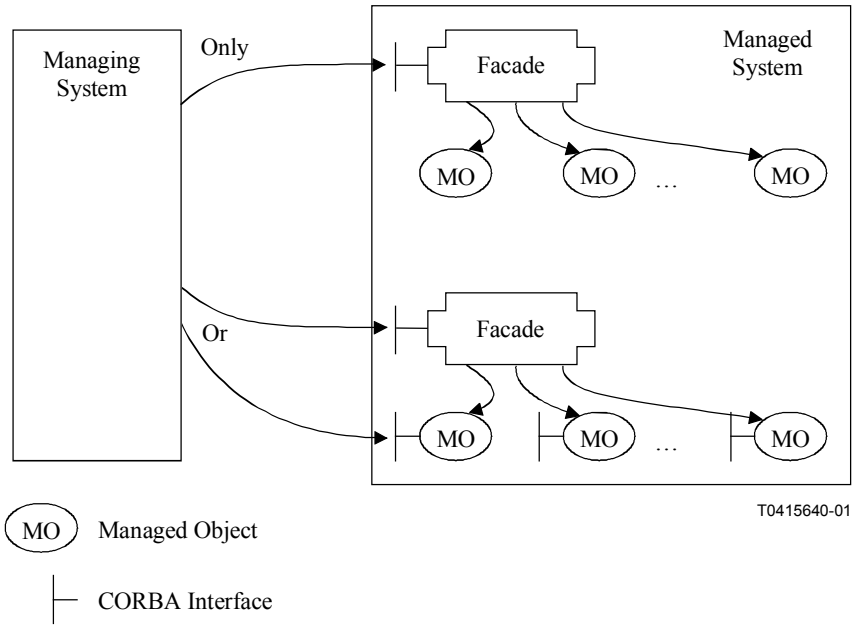

Figure 2/X.780.1 – The facade role

The figure shows a managing system accessing a managed system that supports the coarse-grained approach. The managed system has two facade interfaces 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 facade. The managed objects at the bottom also support direct CORBA interfaces and can be accessed either through the facade or directly. Direct CORBA access is optional, but a managed system that supports the facade approach must provide facade interfaces for each of its managed object instances.

A facade may use a managed object's CORBA interface to invoke an operation on it, or some other implementation-specific means. A managed system, in fact, need not even implement managed objects as individual objects internally. By implementing an interface based on this framework, however, it will give the illusion that managed objects are internally implemented as objects.

When an operation is invoked on a managed object through a facade, the facade must then invoke the operation on the actual managed object or entity. Because many managed objects will be accessed through a single facade, the facade 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 facade operation directed at a managed object.

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.

7.2.2 Managed object name extension

As mentioned above, managed objects accessed through a facade will still have a name even though they may not have an individual CORBA interface. It is important that a managing system be able to determine which facade to use based on the managed object's name. If it cannot it will have to query the managed system or persistently associate a facade IOR with every managed object's name. To support the ability to determine a managed object's facade based on only its name, the names of managed objects accessible through a facade are extended slightly beyond the names of managed objects not accessible through a facade. In the final name component, which always has an *ID* string with the value "Object" (or, as extended by ITU-T Q.816.1, <empty>), the *kind* string is set to the value of a facade identifier assigned to the facade through which the object may be accessed. For managed objects not accessible through a facade, this *kind* string is empty. ITU-T Q.816.1 provides additional details on how the *kind* string in the final managed object name component is used to identify a facade.

7.2.3 Support services for facade-accessible managed objects

The framework support services provided on interfaces that use the facade approach will be largely the same as those defined in ITU-T Q.816. Some, such as the Factory Finder and Channel Finder services, require no change at all. Others, such as the Terminator and Multiple Object Operation (MOO) services, require no changes to their interfaces or the way they are used by managing systems, but may require slight changes to their implementations if they access managed objects using the managed objects' facade interfaces (rather than some implementation-specific method). ITU-T Q.816.1 provides details on the framework support service changes required to support coarse-grained interfaces.

The biggest change to the support services comes in the area of support for naming. The facade interfaces are bound to names in the naming service, much the same way the support service interfaces are. On interfaces that use facades, however, the managed objects' names are not required to be bound to IORs in the OMG Naming Service. Instead, a new service is introduced as a place to store managed object names and containment relationship information. This new service, the Containment Service, is defined in ITU-T Q.816.1.

7.2.4 Facade modelling

To support the facade design pattern and the definition of facades usable with this framework, a new base interface is introduced. This interface will be known as the Managed Object Facade interface. It plays the same role in coarse-grained interfaces as the Managed Object interface does in fine-grained interfaces. That is, it is the base interface from which all managed object facade interfaces must either directly or indirectly inherit to work with the framework. The Managed Object Facade interface is quite similar to the Managed Object interface defined in ITU-T X.780. See clause 8 for the definition of the Managed Object Facade interface.

The changes to the framework are reflected in Figure 3 below. A new superclass, *ManagedObjectFacade*, is added to the figure. Also, the Containment Service is added. Note that it provides access to a database of managed object names. The managed object name database maintained by the Naming Service is shown with dotted lines, indicating that it need not store the names and IORs of managed objects. The Naming Service is still required, however, to enable managing systems to find facade interfaces and support service references. Finally, the managed objects are also shown drawn with dotted lines, to indicate that they need not be directly accessible.

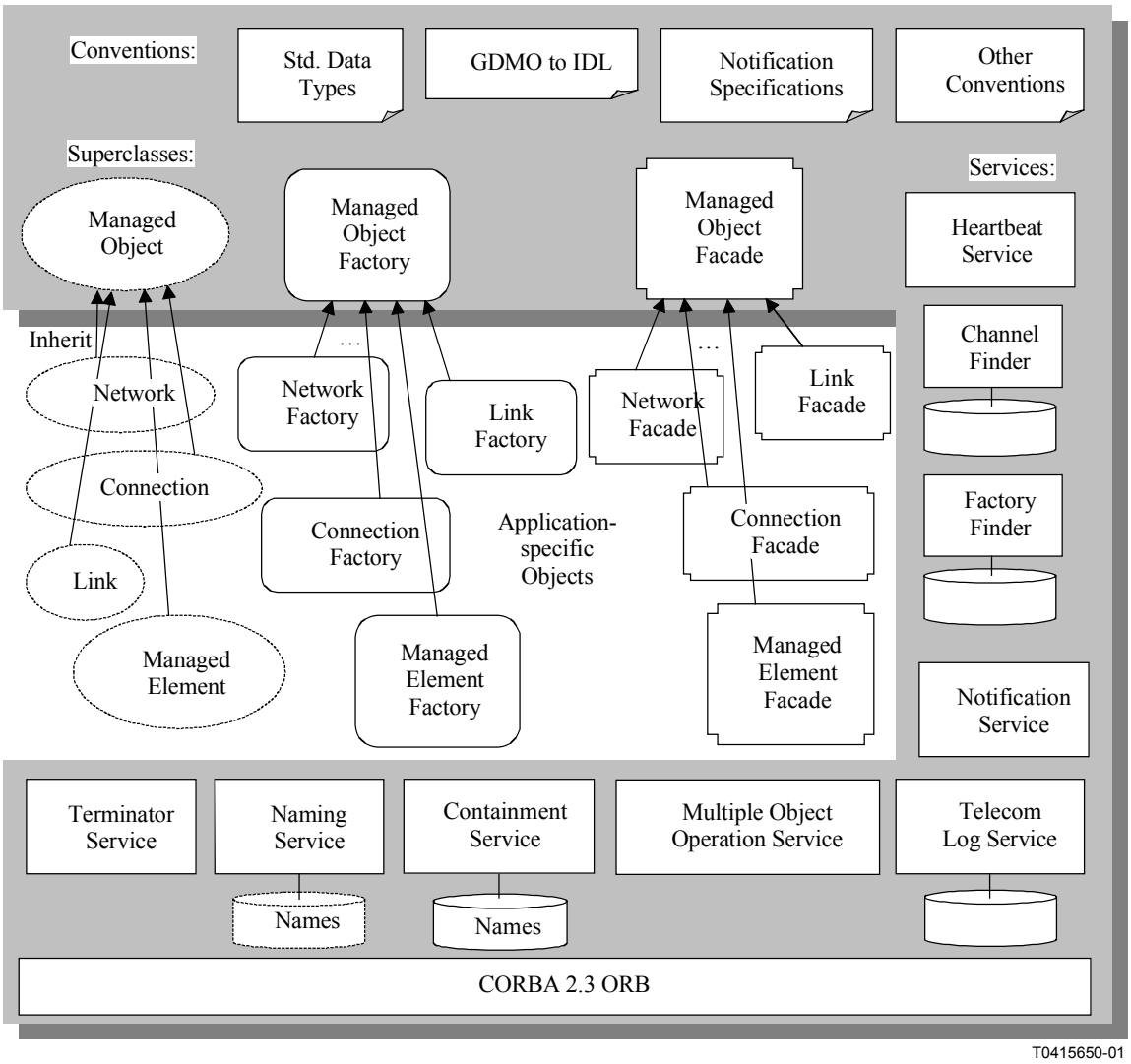

Figure 3/X.780.1 – Framework with extensions to support coarse-grained interfaces

8 Providing facade interfaces for accessing managed objects

As described above, facade interfaces provide a different way of accessing managed objects. On a coarse-grained interface (one where facade interfaces are present), managed objects may or may not be accessible using individual CORBA interfaces. So, a managing system may use a facade to access managed objects by invoking operations on the facade. Even if the managed system does not implement managed objects as separate objects, by supporting this Recommendation it gives the illusion of doing so.

8.1 Facade instantiation

This clause defines requirements managed systems shall follow when providing facade interfaces for accessing managed objects.

(R) FACADE-1 – A managed system shall provide at least one facade interface for each class of managed objects that may be instantiated on it, even if these objects also support direct CORBA interfaces. Facade interfaces for managed object classes that cannot be instantiated on the system need not be provided. This includes superclasses. Thus, a facade for a superclass of managed objects need not be instantiated if managed objects of that type may not be instantiated, even if managed objects of the subclass may be instantiated. This does not mean superclass interfaces do not need to be defined, though. The definition of facade interfaces follows the same inheritance hierarchy as the managed objects. See clauses 9 and 10 below. Name binding requirements for facade interfaces are defined in ITU-T Q.816.1.

Relieving the managed system from providing facade interfaces for superclasses that are not instantiable on the system does not preclude a managing system from taking advantage of polymorphism. A managing system may still treat a facade interface as a superclass of the facade and access capabilities available through the superclass in the same way that any CORBA client may treat an object as a superclass. The operation, though, will actually be invoked on the subclass facade, using that facade's IOR. The programming language on the client system handles the polymorphism.

(R) FACADE-2 – A managed system may provide multiple facade interfaces for a given class of managed objects. Managed objects accessible through one facade interface shall not be accessible through any other. ITU-T Q.816.1 provides details on how to identify which facade to use to access a managed object based on its name.

(R) FACADE-3 – All facade interfaces are created and destroyed by the managed system. A facade interface must exist for the entire time that any of the managed objects accessible through it exist. No notifications are sent when a facade is created or deleted. Therefore, a managing system will become aware of the existence of a new facade if and when managed objects with names indicating that they are accessible through the new facade begin to appear.

(R) FACADE-4 – Each facade operation that is directed at a specific managed object will contain the name of that managed object in the first parameter of the operation. The facade shall invoke that operation on the named managed.object and return the results including exceptions. If the named managed object is not accessible through that facade, the facade shall raise an *applicationError* exception in response to the operation. The error code in this exception shall be set to the *objectNotFound* code defined in the IDL in Annex A.

8.2 The facade interface base class

This clause describes a managed object facade base class that is defined in the IDL in Annex A. All managed object facade interfaces provided on a managed system must inherit, either directly or indirectly, from this interface. This interface provides a set of basic operations all facade interfaces must support to be usable with this framework.

The Managed Object Facade interface, called "*ManagedObject_F*" in the IDL code in Annex A, plays the same role on coarse-grained interfaces as the *ManagedObject* interface plays on fine-grained interfaces. Thus, it is very similar to the *ManagedObject* interface defined in ITU-T X.780. Most of the operations on the *ManagedObject_F* interface are nearly identical to the operations on the *ManagedObject* interface. One new operation, *attributesBulkGet,* is added, and one, *nameGet*, dropped.

8.2.1 Managed object facade basic capabilities

The capabilities that all managed object facade interfaces must support are:

- A method that returns the class name of a named managed object.
- A method that returns the conditional packages supported by a named managed object.
- A method that returns the creation source of a named managed 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 a named managed object. 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 named managed object.
- A method that returns a CORBA value type object containing all of the readable attributes for a set of managed objects.
- A destroy operation.

8.2.2 Managed object facade IDL

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

```
interface ManagedObject_F {
     ObjectClassType objectClassGet(in NameType name)
          raises (ApplicationError);

     StringSetType packagesGet (in NameType name)
          raises (ApplicationError);

     SourceIndicatorType creationSourceGet(in NameType name)
          raises (ApplicationError);

     DeletePolicyType deletePolicyGet (in NameType name)
          raises (ApplicationError);

     ManagedObjectValueType attributesGet (
          in        NameType name,
          inout     StringSetType attributeNames)
          raises    (ApplicationError);

     boolean attributesBulkGet (
          in        NameSetType                  names,
          in        StringSetType                attributeNames,
          in        unsigned short               howMany,
          out       AttributesGetResultSet       attributes,
          out       AttributesGetResultIterator  iterator)
          raises    (ApplicationError);

     void destroy(in NameType name)
          raises    (ApplicationError,
                     DeleteError);

}; // end of ManagedObject_F interface
```

8.2.3 The *objectClassGet*() operation

The *objectClassGet*() operation returns the scoped interface name of the named managed object. The named managed object is the managed object with the name included in the first parameter, named "name".

Note that this value is different from the class name of the facade. Since it is possible for managed objects on a coarse-grained interface to also support direct CORBA interfaces, the decision was made to require this operation to return the same value a managing system would get if it invoked the equivalent operation directly on the managed object. If no fine-grained managed object interface has been defined for the class of managed objects accessible through the facade, the response shall be the scoped interface name of the facade without the trailing "_F". The same value returned in response to this operation will be included in notifications from the object, and in the value type returned for the object. The return type, *ObjectClassType*, is a type definition for string.

Given a reference to a facade, a managing system can determine what type of facade it is through standard CORBA calls (e.g. the get_interface call on interface CORBA::Object). Therefore, a separate operation for this is not defined on the base facade interface.

8.2.4 The *packagesGet*() operation

The *packagesGet*() operation returns the list of conditional packages supported by the named managed object. It is possible for managed objects accessed through the same facade to support different conditional packages. The notion of conditional packages, each with a string name, is defined in ITU-T X.780. *StringSetType* is a type definition for a sequence of strings.

8.2.5 The *creationSourceGet*() operation

The *creationSourceGet*() operation returns a value indicating the system that caused the named managed 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.

8.2.6 The *deletePolicyGet*() operation

The *deletePolicyGet*() operation returns the delete policy for the named managed object. 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.

8.2.7 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 or facade 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 in ITU-T X.780. The managed object must return the subclass 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 an empty 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 empty 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 predefined, 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.

8.2.8 The *attributesBulkGet*() operation

The *attributesBulkGet*() method is used to return multiple attributes from multiple managed objects of the same type that are accessible through the facade. The managing system supplies a list of attribute names, and a list of managed object names from which to retrieve those attributes. The list of attribute names is handled in the same way as it is for the *attributesGet* operation described above. Processing the list of managed object names is described below.

8.2.8.1 Determining the objects from which to retrieve attributes

If the list of names is empty, all objects accessible through the facade are implicitly requested.

If the list includes an item that does not include the final component with an *ID* value of either "Object" or <empty>, then that list item serves as a partial name, which implicitly requests the set of managed objects (accessible through the facade) whose names start with that partial name. Such a name would be created by removing one or more name components from the end of a managed object's name.

For example, start with a Managed Element named with an *ID* of "me1" and using a *kind* value "ME" under a local root defined for acme telecom:

acme\.com/me1.ME/.facadeID1

Notice that in the final name component (after the last '/'), the *ID* string (which precedes the '.') is empty. Therefore, this is not a partial name. Next, assume there is an Equipment object "eq1", named under me1 using "EQ" as kind:

acme\.com/me1.ME/eq1.EQ/.facadeID2

Finally, assume there are Equipment Holder managed objects named under this equipment object (using kind "EH"), as shown in Figure 4 below, all accessible through an equipment holder facade with the ID "facadeID3".

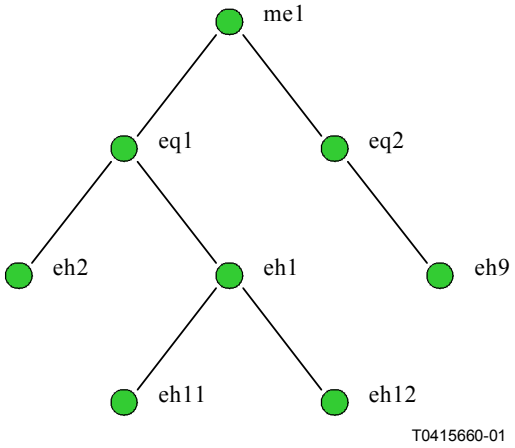

Figure 4/X.780.1 – Example naming tree

If the partial name "acme\.com/me1.ME/eq1.EQ" is used in the list of managed object names parameter of the *attributesBulkGet* operation on the Equipment Holder facade, attribute values from the following equipment holder objects shall be returned:

acme.com/me1.ME/eq1.EQ/eh1.EH/.facadeID3

acme.com/me1.ME/eq1.EQ/eh2.EH/.facadeID3

acme.com/me1.ME/eq1.EQ/eh1.EH/eh11.EH/.facadeID3

acme.com/me1.ME/eq1.EQ/eh1.EH/eh12.EH/.facadeID3

The facade can recognize that the name is a partial name because the ID value in the final name component is not "Object" or <empty>.

While this example did not demonstrate it, the object at the root of the tree identified by the partial name is included in the scope of the operation.

8.2.8.2 Returning the results

Data is returned in strongly typed managed object value types, one from each managed object named. If the facade does not provide access for a managed object name provided by the client, no value type for that object is returned. Since a managed system may provide multiple facade interfaces of the same type, the client may have to invoke this operation on multiple interfaces to retrieve values from all of the managed objects of a given type on a system.

Even if the client does not request that values for the "name" attribute be returned, the facade shall return the name in each managed object value type. It must do this so the client will know which values apply to which managed object.

Along with each managed object value type returned is a list of the names of the attributes in that value type that have valid values. This list may not match the list of requested attributes, as the instance may not support all of the requested attributes. If the instance supports none of the requested attributes the facade shall return a managed object value type for that instance with only the name attribute containing a valid value.

Since a potentially large amount of data may be returned, the iterator design pattern is used. The client specifies the maximum number of value types to be returned. The rest must be returned in an iterator. If an iterator is needed, the return value shall be true. Otherwise, it shall be false and the iterator reference shall be nil.

8.2.9 The *destroy*() operation

The final operation on the base facade, the *destroy*() operation, is used to release any resources associated with the named managed object and to delete it. The *DeleteError* exception is raised if the object 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) FACADE-5 – The facade interfaces defined for a coarse-grained CORBA interface shall inherit (directly or indirectly) from the *ManagedObject_F* interface described above and defined in the CORBA IDL in Annex A. The capabilities described above shall be supported.

8.3 The *AttributesBulkGet* iterator interface

As mentioned in 8.2.8, if many results are to be returned in response to an *attributesBulkGet* operation, an iterator may be required. The iterator design pattern is a well-known CORBA design pattern. When large amounts of data are to be retuned in response to an operation, a reference to an iterator interface is returned instead. The client may then query the iterator to retrieve the results in manageable chunks.

The IDL description of the "attributes get result" iterator is shown below.

```
interface AttributesGetResultIterator {

     boolean getNext(in unsigned short howMany,
          out AttributesGetResultSet results)
          raises (ApplicationError);

     void destroy();

}; // end of interface AttributesGetResultIterator
```

(R) FACADE-6 – The managed system shall instantiate an iterator with an interface matching the description of the *AttributesGetResultIterator* definition in the IDL in Annex A when the number of results to be returned in response to an *attributesBulkGet* operations exceeds the number requested by the client.

(R) FACADE-7 – Each time a client invokes a *getNext* operation on the iterator, it shall return the next set of results. The iterator shall keep track of how many results have already been retrieved by the client, and return all of the results once. The results initially returned in response to the *attributesBulkGet* operation shall not be returned again by the iterator. The iterator shall return in response to a *getNext* operation at most the number of names indicated by the value of the *howMany* parameter. The iterator may return less than the requested batch size, balancing the efficiency of returning results in a large batch with the possible need to block until more results are available. If there are more results to return (in addition to those being returned), the return value of the *getNext* operation shall be true, otherwise false. The iterator shall not return an empty result set unless *howMany* was set to zero or there are no more results to return, as doing so would force the client to poll the iterator.

(R) FACADE-8 – The managed system shall control the life-cycle of the iterator. A destroy operation, however, is provided if the manager wants to stop retrieving results before reaching the last result. Upon invocation of the *destroy* operation, the iterator shall free any resources it is using and delete itself. Upon returning the last result, the iterator shall destroy itself. The iterator may also be destroyed by the managed system if it is unused for an unreasonably long period of time.

8.4 Factory instantiation

Factories are persistent object interfaces that are used to instantiate other objects. The use of factories follows a well-known CORBA design pattern. Factories are used on fine-grained interfaces to provide a managing system with a method to create new instances of managed objects. Even though not strictly required on coarse-grained interfaces because the facade could play the role of the factory, separate factory interfaces shall be used on coarse-grained interfaces. One advantage of doing this is to make the coarse-grained and fine-grained approaches more compatible. Another advantage is to prevent the inheritance of create operations for superclass objects by subclasses. This problem does not occur with separate factories because factory interfaces do not follow the inheritance hierarchy of the managed objects. The factory interfaces used to create managed objects on coarse-grained interfaces are the same as those defined for fine-grained interfaces. See clause 9.

(R) FACADE-6 – A managed system shall provide at least one factory interface for each class of managed object that may be instantiated on it. Factory interfaces for managed object classes that cannot be instantiated on the system need not be provided. Factory interfaces are registered with the Factory Finder service, defined in ITU-T Q.816. Factories on coarse-grained interfaces may return a nil reference in response to a create operation rather than a reference to the newly created object. The managed object name returned by the factory in response to a create operation shall indicate the facade that may be used to access the new object according to the managed object naming rules defined in ITU-T Q.816.1.

9 Coarse-grained CORBA modelling guidelines

This clause defines the rules for defining coarse-grained IDL interfaces. A coarse-grained interface is created by first defining a fine-grained interface according to the guidelines defined in ITU-T X.780. ITU-T X.780 covers both creating IDL interfaces from scratch as well as translating a GDMO interface to IDL. Once a fine-grained interface is defined, facade interfaces for each of the managed object interfaces are then developed according to the rules defined in clause 10 below. The fine-grained managed object interface specifications shall be retained. All of the other constructs defined for the fine-grained interface, including data types, value types, exceptions, notifications, and factories, are reused without modification on the coarse-grained interface.

10 Guidelines for translating fine-grained models to coarse-grained

A coarse-grained IDL model may be created from a fine-grained IDL model by following the steps below.

1) A facade interface shall be created for each managed object interface. A managed object interface is an interface that is derived either directly or indirectly from the *ManagedObject* interface.

2) The facade interfaces shall be created within the same IDL module as the fine-grained object interfaces. This relieves the modeller from having to include type definitions for all of the types defined for the fine-grained model.

   The facade interfaces may be created in a separate file, or included in a new version of the fine-grained file. If in a separate file, the file containing the fine-grained model will have to be included for compilation.

Splitting a module across files is allowed by OMG standards. Basically, an IDL module defines a name space. Within a module, all names must be unique. So, for example, a module cannot contain two interfaces named "ManagedObject_F." Two different modules can contain identical names, though, and modules can be contained within other modules. When a module is split across files the uniqueness rule still applies. Duplicate names in the same module, but in separate files, are not allowed.

The IDL in Annex A is defined within a single module named "itut_x780", which is the same module used in ITU-T X.780. Thus, this module is split across files. One impact is that no names used in the IDL in ITU-T X.780 can be reused in the IDL in this Recommendation. An advantage, however, is that any IDL constructs defined in ITU-T X.780 can be reused in this Recommendation by simply including the file with a precompiler directive. No type definitions or scoped names are required, as would be the case if the IDL was in separate modules. Note that compliance and conformance are still based on documents, not IDL modules. Thus, a system may be conformant to ITU-T X.780 without being conformant to this Recommendation.

3) The name of the facade interface shall be the name of the fine-grained interface from which it is created, appended with "_F" (an underscore followed by a capital 'F'). So, the facade interface created for the "Equipment" managed object shall be named "Equipment_F".

4) If the fine-grained object interface inherits directly from the *ManagedObject* interface, the facade interface created for it shall inherit from the *ManagedObject_F* interface. If the fine-grained object interface instead inherits from a subclass of the *ManagedObject* interface, the facade interface created for it shall inherit from the facade interface created for that subclass. So, the inheritance hierarchy of the facade interfaces matches that of the managed object interfaces. For example, say the *EquipmentHolder* fine-grained object interface inherits from the *Equipment* interface. The *EquipmentHolder_F* interface shall then inherit from the *Equipment_F* interface. Facade interfaces for all of a fine-grained object's superclasses must be created before the fine-grained object's facade may be created.

5) The entire contents of the fine-grained object interface shall be copied to the facade interface with the following changes:

- An *in* parameter of type *NameType* and named *name* shall be added as the first parameter to each operation. This parameter shall be used to pass in the name of the target managed object on which the operation should be invoked. If the operation already has a parameter named "name," it shall be renamed.
- Any parameter or return type that uses a managed object IOR must be translated to a *NameType*. A managed object IOR value will be identified as a reference to a *ManagedObject* interface or subclass interface. The use of such types of values is discouraged on fine-grained object interfaces, so few if any should be encountered.
- It is acceptable for a behaviour description on a coarse-grained interface specification to reference a behaviour description of the equivalent construct on a fine-grained interface specification rather than duplicate it.

6) Fine-grained modelling guidelines currently prohibit the use of OMG IDL attributes to model managed object attributes. This is because OMG IDL does not allow user-defined exceptions to be raised on attribute access operations. So, instead, managed object attributes are modelled with separate operations used to get or set the value of the attribute, or add or remove values to or from the attribute. In the future, though, the OMG will likely allow user-defined exceptions on attribute access operations. Should this happen, and should a fine-grained object interface use IDL attributes, then when creating a facade for this object the attributes shall be translated into separate IDL operations, then the *name* parameter shall be added as the first parameter of those operations.

7) The rest of the fine-grained interface IDL such as data types and value types is used on the coarse-grained interface without modification.

11 Coarse-grained IDL compliance and conformance

This clause defines the criteria that must be met by other standards documents claiming compliance to these guidelines and the functions that must be implemented by systems claiming conformance to this Recommendation.

11.1 Standards document compliance

Any specification claiming compliance with these guidelines shall:

1) Support all of the standards document compliance requirements of ITU-T X.780.

2) Follow the fine-grained IDL to coarse-grained IDL mapping rules defined in clause 10.

11.2 System conformance

An implementation claiming conformance to this Recommendation shall:

1) Meet all of the facade, factory, and iterator instantiation requirements specified in clause 8.

2) Implement an IDL interface compliant with the guidelines in this Recommendation. See clause 11.1.

11.3 Conformance statement guidelines

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

ANNEX A

Coarse-grained modelling IDL

```
/* This IDL code is intended to be stored in a file named "itut_x780_1.idl"
located in the search path used by IDL compilers on your system. */

#ifndef ITUT_X780_1_IDL
#define ITUT_X780_1_IDL

#include <itut_x780.idl>

#pragma prefix "itu.int"

module itut_x780 {

// IMPORTED TYPES
// DATA TYPES

     /** This structure holds the results of retrieving a set of attribute
     values from a single managed object. The attribute values are placed
     in the strongly typed attributes member. Because not all values in
     the attributes may have been requested or supported, the attribute
     names member holds the list of attribute names for which the
     attributes member holds valid values. The rest are invalid. */
```

```
    struct AttributesGetResultType {
         ManagedObjectValueType  attributes;
         StringSetType           attributeNames; };

    typedef sequence <AttributesGetResultType> AttributesGetResultSetType;

    interface AttributesGetResultIterator;

// ATTRIBUTES GET RESULT ITERATOR INTERFACE

    /** The Attributes Get Result Iterator interface is used to retrieve
    the results from an attributesBulkGet operation using the iterator
    design pattern. */

    interface AttributesGetResultIterator {

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

         boolean getNext(in unsigned short howMany,
              out AttributesGetResultSetType results)
              raises (ApplicationError);

         /** This method is used to destroy the iterator and release its
         resources. The iterator, though, is automatically destroyed after the
         last results are returned, and may be destroyed if unused for an
         unreasonably long period. */

         void destroy();

    }; // end of interface AttributesGetResultIterator

// MANAGED OBJECT FACADE

    /** The Managed Object facade is intended to be the base interface from
    which all other managed object facades inherit. It is a central place to
    specify basic functions which all managed object facades are expected to
    support. */

    interface ManagedObject_F {

         /** This method returns the scoped name of the most-specific class of
         the managed object (e.g. "itut_x780::EquipmentR1").
         *NOTE* This operation returns the class name of the object, not the
         facade. This is the name of the facade interface minus the trailing
         "_F". This is also the same name that goes in the objectClass parameter
         of notifications, even on interfaces supporting facades.
         @param name    The name of the managed object instance on which the
                        operation is to be invoked.
         @return        The interface name of the managed object.
         */

         ObjectClassType objectClassGet(in NameType name)
```

```
    raises (ApplicationError);

/** This method returns a list of all the conditional packages
supported by this instance.
@param name   The name of the managed object instance on which the
              operation is to be invoked.
@return       The list of package names suported by the managed object
*/

StringSetType packagesGet (in NameType name)
    raises (ApplicationError);

/** This method returns an indication of how the object was created.
@param name   The name of the managed object instance on which the
              operation is to be invoked.
@return       An indication of whether the named managed object was
              created autonomously or by a managing system
*/

SourceIndicatorType creationSourceGet(in NameType name)
    raises (ApplicationError);

/** This method returns a value indicating if the object may be deleted
and if it may, if all contained objects are automatically deleted.
@param name   The name of the managed object instance on which the
              operation is to be invoked.
@return       The delete policy of the named managed object
*/

DeletePolicyType deletePolicyGet (in NameType name)
    raises (ApplicationError);

/** This method may be used to generically get all of the attributes
supported by an instance. Each interface is expected to sub-class the
Managed Object value type and add the other attributes supported by
that interface. The managed object must return a value object of that
type. The client must then narrow the reference to access all the
attributes. <p>

The client may also submit a list of names indicating the attributes it
wishes to receive. These names must match the member names in the value
object. For members not on the list, and for members that are part of
packages that are not supported, the server may return any value but it
should be as short as possible. The server also returns the list of
attributes, which may be shorter due to exclusion of attributes in
unsupported packages. The client must regard the value of any member
not in the returned list as garbage. <p>

A null attribute names list indicates that all supported attributes are
to be returned. The server must return the actual list.
@param name             The name of the managed object instance on
                        which the operation is to be invoked.
@param attributeNames   A list of names of attributes to be retrieved.
@return                 The value type containing the attriubtes.
*/

ManagedObjectValueType attributesGet (
    in       NameType name,
    inout    StringSetType attributeNames)
    raises   (ApplicationError);
```

```
/** This method is used to return multiple attributes from multiple
managed objects of the same type. The client supplies a list of
attribute names, and a list of managed object names from which to
retrieve the attributes. <p>

Data is returned in strongly-typed managed object value types, one from
each managed object named. If the facade does not provide access for a
managed object name provided by the client, no value type for that
object is returned. Since a managed system may provide multiple facade
interfaces of the same type, the client may have to invoke this
operation on multiple interfaces to retrieve values from all of the
managed objects of a given type on a system. <p>

Even if the client does not request that values for the 'name'
attribute be returned, the facade shall return the name in each managed
object value type. If it does not, the client will not know which
values apply to which managed object instance. <p>

Along with each managed object value type returned is a list of the
names of the attributes in that value type that have valid values. This
list may not match the list of requested attributes as the instance may
not support all of the requested attributes. The value "name" shall
always be on the returned list. If the instance supports none of the
requested attributes the facade shall return a managed object value
type for that instance with only the name attribute containing a valid
value. <p>

Since a potentially large amount of data may be returned, the iterator
design pattern is used. The client specifies the maximum number of
value types to be returned. The rest must be returned in an iterator.
If an iterator is used, the return value shall be true. Otherwise, it
shall be false and the iterator reference shall be null.
@param names            The names of the managed objects from which to
                        retrieve the attribute values.
@param attributeNames   The names of the attributes to retrieve.
@param howMany          The maximum number of value types to return in
                        the attributes parameter.
@param attributes       The first batch of results.
@param iterator         A reference to an iterator, if needed.
                        Otherwise, null.
@return                 True if an iterator is being returned,
                        otherwise false.
*/

boolean attributesBulkGet (
     in       NameSetType                  names,
     in       StringSetType                attributeNames,
     in       unsigned short               howMany,
     out      AttributesGetResultSetType   attributes,
     out      AttributesGetResultIterator  iterator)
     raises   (ApplicationError);

/** This method destroys the object. It is used to simply release any
resources associated with the managed object. It does not check for
contained objects or remove name bindings from the naming tree. <p>

The intent of this operation is to allow support services to destroy
the managed object. <p><b>

NOTE - Direct invocation of this operation from a managing system could
corrupt the naming tree and is recommended only under extraordinary
circumstances. Clients wishing to delete an object should instead use
the terminator service. </b>
@param name   The name of the managed object instance on
              which the operation is to be invoked.
*/
```

```
          void destroy(in NameType name)
               raises    (ApplicationError,
                         DeleteError);

     }; // end of ManagedObject_F interface

// ApplicationErrorConst Module

/** This module contains the constants defined for the error code contained in
Application Error Info structures returned with Application Error exceptions.
*/

module ApplicationErrorConst {

     /** This application error exception code indicates that a target object of
     an operation could not be found. */

     const short objectNotFound = 4;

}; // end of module ApplicationErrorConst

}; // end of module itut_x780

#endif // end of #ifndef ITUT_X780_1_IDL
```

SERIES OF ITU-T RECOMMENDATIONS

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

Printed in Switzerland
Geneva, 2002