TTC標準
Standard

ＪＴ－Ｘ７８０

CORBA 管理オブジェクト定義のための TMN ガイドライン

TMN guidelines
for defining CORBA managed objects

第１版

2004 年 4 月 20 日制定

社団法人
情報通信技術委員会

THE TELECOMMUNICATION TECHNOLOGY COMMITTEE

TTC Telecommunication Technology Committee

目　次

＜参考＞

１．国際勧告等との関連

本標準は、ITU-T 勧告 X.780(01/2001)に準拠したものである。

２．上記国際勧告等に対する追加項目等

2.1 オプション選択項目

なし。

2.2 ナショナルマター項目

なし。

2.4 原勧告と章立ての構成の相違

なし。

３．改版の履歴

版　数	制　定　日	改　版　内　容
第1版	２００４年４月２０日	制　定

４．工業所有権

本標準に関わる「工業所有権等の実施の権利に係る確認書」の提出状況は、ＴＴＣホームページでご覧になれます。

５．その他

（１）参照している勧告・標準等

ITU-T 勧告：M.3010(2000), M.3020(2001), M.3120(2001), Q.816(2001), X.701(1997), X.703(1997), X.720(1992), X.721(1992), X.722(1992), X733(1992), Q.821(2000)

１．範囲

ITU-T M.3010 に定義されている TMN アークテクチャは分散処理からコンセプトを導入し、また複数の管理プロトコルを使用することができる。TMN 内と TMN 間のインタフェースのために、初期の TMN インタフェース仕様は、プロトコルとしての共通管理情報プロトコル(CMIP)とともに OSI システム管理から管理オブジェクト定義ガイドライン（GDMO）表記を用いて開発された。TMN 間インタフェース（X）は応用層での可能な選択として CMIP と CORBA GIOP/IIOP を含んでいた。

分散処理技術である CORBA は、主に情報技術分野で受け入れられているという理由から TMN 通信アーキテクチャの中で使用を検討されている。この受け入れは CORBA ベースのインタフェース開発におけるよりよい開発ツールや広まった専門性によって、CORBA ベースのインタフェースの可用性を強化すると期待される。Object Management Group (OMG)によって開発された本技術はまた、複数の業界によって検討されている。本技術を用いる仕様は標準の応用プログラムインタフェース（API）とプログラム言語に対する言語結合のサポートを提供し、ソフトウェア移植性を円滑にする。ORB 間プロトコルとの組合せでオブジェクトリクエストブローカが提供する相互運用性解決法は、クライアント・サーバ間の相互運用性に対処する。CMIP と情報モデルがマネージャ・エージェントシステム間の相互運用性に解決法を与える一方で、CORBA はオブジェクトが分散され得るオブジェクト間相互作用を定義する。

1.1 目的

幾つかのグループは、CORBA サービスに伴う表記法としての IDL による CORBA モデル化技法を用いるネットワーク管理仕様の開発を行っている。本標準の範囲は、相互運用可能な CORBA ベースのネットワーク管理インタフェース仕様において適用可能なガイドラインを定義することである。″X″インタフェースに対する要求は、“内部”管理で使われる″Q″インタフェースに対するそれとは異なる。本標準の範囲は、CORBA が使用され得る TMN において全てのインタフェースをカバーする。ここで定義される能力とモデル全てが全 TMN インタフェースにおいて要求されるとは期待されていない。これはフレームワークが全ての抽象化レベル（管理相互及び管理内部）において管理システムとネットワーク要素間と同様に管理システム間のインタフェースに使用可能であることを意味する。

ITU-T Q.816 [1]は、CORBA ベースの TMN インタフェースに必要なサービスの集合を定義する。本標準は、サービスの適用が可能な CORBA IDL で書かれた情報モデルを規定するためのガイドラインを定義する。さらに本標準では既存の GDMO モデルを IDL に翻訳するための規則を提供する。最終的に本標準では全ての CORBA ベースの TMN 情報モデルに使用される幾つかの基本 IDL コードを定義する。本標準と ITU-T Q.816 の組み合せにより CORBA ベースの TMN インタフェースの定義と実装のための *フレームワーク*が構成される。

通信管理インタフェースでの共通フレームワーク使用は幾つかの利点を持つ。幾つかの例として、通信の一般的な要求に合致するために開発されたモデル再利用を容易にする；通信業界で CORBA サービスを使用するに当たっての規範となる；TMN の新サービス定義を容易にする；既存の豊富なモデル群のセマンティクスを再利用する；CMIP に対する ITU-T X.720, X.721 および X.722 と類似の単一ソースを用いたグループ横断のモデル化アプローチを調和する、といったことがあげられる。モデル化資源に対する共通アプローチの再利用および様々なネットワーク技術とネットワーク管理アプリケーションに対する汎用情報モデルの再利用はネットワーク管理システム開発コスト低減を図りながら新しいネットワークサービスの導入を早める。

通信業界は CMIP ネットワーク管理プロトコルのための情報モデルの開発に非常に多くの時間とエネルギーを投資してきた。TMN CORBA フレームワークの第一の目標は、これらの情報モデルのセマンティクスを殆ど変更せずに CORBA Interface Definition Language (IDL)への翻訳を可能にすることで再利用することにある。結果として、初期の IDL 情報モデルは CMIP モデルから導き出されると期待される。

1.2 応用

ITU-T M.3020 は TMN 仕様の開発において３つのフェーズを定義する。3 つのフェーズとは要求、分析そして設計である。図１は、このプロセスに関連する CORBA ベースのインタフェース仕様開発のために、このプロセスと本標準の範囲を示す。

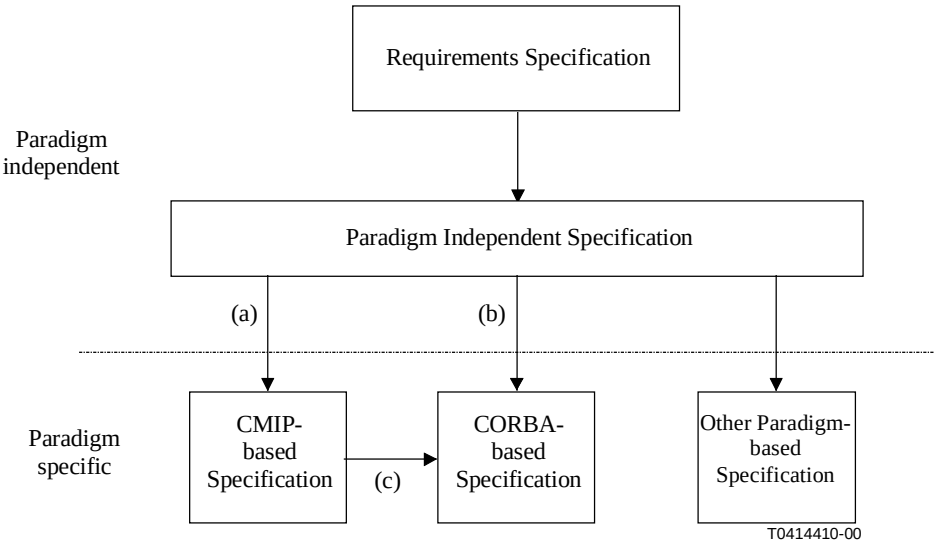

図 1 /JT-X780　　CORBA ベース仕様

(ITU-T X.780)

要求と分析は、ネットワーク管理技術パラダイム特有ではない手法を使って規定される。パラダイム非依存の仕様である分析フェーズからの出力は、パラダイム固有の設計フェーズに対する入力として使用される。

設計フェーズではネットワーク管理パラダイム固有の特長が情報モデルの定義に使用される。これらのパラダイム固有の仕様は振舞い（通常、自然言語による）と正式なインタフェースシグネチャ（例えば、GDMO、IDL）を統合する。

(a)や(b)のような記号のついた矢印は、分析の出力が CMIP と共に使用される GDMO/ASN.1 ベースのモデル、または CORBA/IIOP と共に使用される IDL モデルにそれぞれ写像されることを示す。現段階でこれらのモデルを生成するための利用可能な処方的規則は存在しない。将来、ITU-T M.3020 においてそのような規則を開発する可能性があり得る。

本標準は、CORBA/IIOP が CMIP の代わりに使用されるならば CMIP パラダイムにおいて開発済みの既存モデルの再利用に対応する。(c)で示される矢印は本標準によって対応される。

GDMO/ASN.1 定義から CORBA/IDL への変換を開発することにおいて、二つの手法が可能である。

- 最初の手法においては、構文の全ての要素は充分に規定されたアルゴリズムまたは処方的定義を使用してCORBA/IDLへ翻訳される。この手法は相互運用性を支援するためにゲートウェイが使用されるJoint Inter-Domain Management (JIDM)によって取られている手法である。
- 二番目の手法（本標準で使用されている）は、構文の全ての要素を処方的に翻訳しない。むしろ、セマンティクスを保存しCORBAの特長を使用するやり方で要素は既存のGDMOから翻訳される。この手法はゲートウェイ経由で相互に作用するためには使用されないで、通信のコンテキストに合致させるために開発されたモデルの要求とセマンティクスを保存するために使用される。それは、管理システムと被管理システムがCORBA/IIOPを用いて通信するように設計されるとき適用される。

ここで定義される GDMO 情報モデルからの翻訳に対する勧告に追加して、ITU-T Q.816 は通信ネットワーク

管理に使用されるCORBAサービスのための勧告を定義する。Q.816フレームワークの見方はどのようにCORBAベースの仕様が開発されるかに関わらず適用可能である（即ち、図１における(b)または(c)に指定されたパスを使用する）。

CMIP情報モデルをうまく利用することに加えて、ガイドラインのもう一つの目的はCORBAをうまく利用することである。フレームワークは、Common Object Service群を含んでCORBA仕様に定義されている機能を活用する。これらのガイドラインは又、適宜CORBA手法とデザインパタンを再利用する。最終的に、既存のモデルの再利用が重要である一方で、フレームワークが新たなモデルの開発をサポートすることも同等に重要である。これらのガイドラインは、IDLモデルの開発に先立ってGDMOモデルが開発されていることを必要としない。事実、このフレームワーク中で使用されるために新たなIDL情報モデルの開発は、直接的であり、そうするためのガイドラインが提供される。

ITU-T M.3120 [11]はもともとITU-T M.3100で定義された汎用ネットワーク情報モデルのCORBA IDLバージョンを提供する。IDL バージョンはここで定義されるオブジェクトモデル化ガイドラインに従い、かつITU-T Q.816に定義されるCORBAベースのTMNサービスを使用するように設計されている。

1.3 ロードマップ

本標準は以下のような構造を持つ：

1節　導入、ドキュメントロードマップ、更新。

2節　参考文献

3節　勧告の以降の部分を通して使用される略号の定義。

4節　オブジェクトモデル化ガイドラインに対する要求。これらはガイドラインが満たさなければならないデザインの目標である。

5節　ネットワーク管理インタフェース仕様のなかで使用されかつサブクラス化されるインタフェースを定義するCORBA IDLモジュールの記述。実際のIDLは付属資料 AおよびBにある。

6節　CORBAベースのTMN情報モデル定義のためのガイドライン。これらのガイドラインは、ITU-T Q.816のTMN CORBAベースのサービスを使用するIDLオブジェクトのために特に設計されている。

7節　ITU-T Q.816のTMN CORBAベースのサービスとともに使用されるのが適当なIDLモデルへGDMO情報モデルを翻訳するためのガイドライン。

8節　CORBA IDLネットワーク管理インタフェース仕様のための様式成句。

9節　遵守と適合ガイドライン

付属資料　A　モデル化ガイドライン仕様のためのIDLモジュール。本付属資料は規範的である。

付属資料　B　モデル化ガイドラインが使用する定数を定義する追加のIDL。本付属資料は規範的である。

1.4 約束事

読者が本文の目的に注目するように仕向けるため本標準においていくつかの約束事に従う。勧告の大半は規範的である一方で、管理システム（管理する、且つ又管理される）が満足すべき必須の要件を簡潔に述べる項の先頭には括弧に囲まれた太字“R”が記載してある。

例えば：

(R)例－１　必須の要件例。

管理システムが任意に実装しうる要件は同様に“R”のかわりに“O”が先頭に記載される。例えば：

(O)オプション－１　任意の要件例。

要件記述は遵守と適合プロフィールを生成するために使用される。

CORBA IDLの多くの例が本標準に含まれ、またデータ型と基底クラスを規定するIDLが規範的付属資料に含まれる。IDLは9ポイントクーリエ書体で表現される。

```
// Example IDL
interface foo {
```

```
            void operation1 ();
    };
```

本標準の電子バージョンから IDL を取り出しコンパイルするための指示は 1.5 節に表されている。

1.5 IDL のコンパイル

ネットワーク管理インタフェースを規定するために IDL を使用することの利点は、ORB を伴ったツールによって IDL はプログラミングコードに“コンパイル”可能なことである。これは、ネットワーク管理アプリケーションの相互運用を可能とするためのコードのいくらかの開発を実際に自動化する。本標準は、実装者が取出してコンパイルしようとするコードを含む二つの付属資料を持つ。付属資料 A、付属資料 B 共に規範的であり、本標準に適合するシステムを実装する開発者によって使用されるべきである。本標準にある IDL は正確さを保証するため二つのコンパイラを使って調査された。CORBA2.3 仕様をサポートするコンパイラを使用しなければならない。

付属資料はコンパイル可能な通常テキストファイルに切り貼りを簡単に行えるように書式を整えられている。以下はこれがどのように行われるかのヒントである。

1) 本標準のMicrosoft® Word®版ファイルからの切り貼りがより良く働くように見える。Adobe® Acrobat®ファイル形式はコンパイルできないページヘッダおよびフッタを含むように見える。

2) ″/* This IDL code…″行で始まって最後まで通して、付属資料 A全ては、IDLコンパイラが発見するディレクトリ中で″itut_x780.idl″と名前の付けられたファイルに保存されなければならない。

3) ″/* This IDL code…″行で始まって最後まで通して、付属資料 B全ては、IDLコンパイラが付属資料 Aを含むファイルと同じディレクトリの中で見つける″itut_x780Const.idl″と名前の付けられたファイルに保存されなければならない。

4) これらの付属資料に埋め込まれた見出しを除去する必要はない。これらはIDLコメントの中にカプセル化されていて、コンパイラは無視する。

5) 特別な″/**″列で始まるコメントはIDLをHTMLに変換するコンパイラによって認識される。これらのコメントはしばしばこのようなコンパイラに対して特別な書式化命令を持つ。IDLと連動するそれらは、結果のHTMLファイルがファイル全体の迅速なナビゲーションに役立つリンクを持つようなHTMLを生成し得る。

付属資料は殆どどのようなテキストエディタでもテキストを扱えるように、8 スペース間隔のタブスペースとハード改行によって書式を整えられている。

２．参考文献

2.1 規範的参考文献

以下の ITU-T 勧告およびその他の参考文献は、この文章での参照を通じて、本標準の条件を構成する条件を含む。出版の時点で表示された版は有効である。全ての勧告および他の参考文献は改版に従う；それ故、全ての使用者に以下に列挙された勧告および他の参考文献の最も新しい版の適用可能性を調査することを奨励する。現在有効な ITU-T 勧告のリストは定期的に発行されている。

[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．定義と略号

3.1 ITU-T X.701 からの定義

本標準に使用されている以下の用語は Systems Management Overview (ITU-T X.701)にて定義される：

– 管理オブジェクトクラス；

– マネージャ；

– エージェント。

–

3.2 ITU-T X.703 からの定義

本標準で使用される以下の用語は Open Distributed Management Architecture (ITU-T X.703)にて定義される。

– 通知。

3.3 略号

本標準は以下の略号を使用する：

ASN.1 抽象構文記法 No.1 (Abstract Syntax Notation No. 1)

ATM 非同期転送モード (Asynchronous Transfer Mode)

CMIP 共通管理情報プロトコル (Common Management Information Protocol)

CORBA 共通オブジェクトリクエストブローカアーキテクチャ (Common Object Request Broker Architecture)

COS 共通オブジェクトサービス (Common Object Services)

DN 識別名 (Distinguished Name)

EMS 要素管理システム (Element Management System)

GDMO 管理オブジェクト定義のためのガイドライン (Guidelines for the Definition of Managed Objects)

GIOP 汎用相互運用プロトコル (General Interoperability Protocol)

HTML 超テキストマークアップ言語 (Hypertext Markup Language)

ID 識別子 (Identifier)

IDL インタフェース定義言語 (Interface Definition Language)

IIOP インタネット相互運用プロトコル (Internet Interoperability Protocol)

IOR 相互運用可能オブジェクト参照 (Interoperable Object Reference)

ITU-T 国際通信連合－通信標準化部門 (International Telecommunication Union - Telecommunication Standardization Sector)

JIDM 共同領域間管理 (Joint Inter-Domain Management)

MO 管理オブジェクト (Managed Object)

NE ネットワーク要素 (Network Element)

NMS ネットワーク管理システム (Network Management System)

OAM&P 運用、管理、保守および設定 (Operations, Administration, Maintenance, and Provisioning)

OID オブジェクト識別子 (Object Identifier)

OMG オブジェクト管理グループ (Object Management Group)

ORB オブジェクトリクエストブローカ (Object Request Broker)

OSI 開放型システム相互接続 (Open Systems Interconnection)

PDU　プロトコル情報単位（Protocol Data Unit）

QoS　サービス品質（Quality of Service）

RDN　相対識別名（Relative Distinguished Name）

TMN　通信管理ネットワーク（Telecommunications Management Network）

TTP　トレール端点（Trail Termination Point）

UID　万国共通識別子（Universal Identifier）

UML　統一モデル化言語（Unified Modelling Language）

UTC　世界時コード（Universal Time Code）

４．CORBA モデルの目標と必要条件

この節は、CORBAを使用して、TMN資源をモデル化するための重要な目標、およびこれらの目標を支援するためにモデル化するガイドラインが満たさなければならない必要条件について記述する。4.1節は、モデル化するガイドラインの目標を紹介する。さらに、その後のサブ節では用語と必要条件を提供する。4節で記述する必要条件はフレームワークが満たすべき必要条件である。それらはテレコミュニケーション管理の要求に基づく。その後、5節、6節、7節および8節では、これらの要求を満たし、ある方法でCORBAを使用することにより、4節の必要条件をどのように実現するかを定義するモデル化のガイドラインを記述する。CORBAを使用する上での規範が、5節、6節、7節および8節で必要条件として記述されている。

4.1　目標

本標準は、テレコミュニケーション・ネットワーク管理システムおよびネットワークエレメントがサポートされているインタフェース上で CORBA 管理対象オブジェクトをどのように定義すればよいのかというガイドラインを指定する。モデル化するガイドラインのいくつかの重要な目標は次のとおりである:

.アプリケーションの相互運用性

.CORBA 共通オブジェクト・サービスの一般的用法

.情報モデルの透過性

この節はこれらの 3 つの目標をさらに詳しく述べる。

4.1.1　適用相互運用

TMN アーキテクチャーの重要な目標の中で、情報アーキテクチャーはネットワーク管理システムサプライヤの種々のセットからのシステム間の相互運用および情報交換の提供のために標準のフレームワークを促進することである。システム間の相互運用は、開発の多くの様相を含んでいる。その最も低い層では、共通のコミュニケーション・メカニズムが、共通のシンタックス、接続の確立、およびシステム間のオペレーションの要求/応答の交換を提供するのに適切でなければならない。相互運用のこの様相は、CORBA の仕様により本質的に提供される。

TMNについては、アプリケーションの相互運用を提供する必要がある。すなわち、種々のサプライヤから提供される管理システムが、そのネットワークの管理を行うために必要な異なる機能を提供するために、単一の運用管理のTMNの内で利用されるであろう。これらの様々なサプライヤーのシステムの統合を単純化するために、彼らは交換されている情報のセマンティクスに同意しなければならない。これは情報モデルの仕様により実現される。この節はこれらの情報モデルの定義のための規則を指定する。

4.1.2　CORBA 共通オブジェクト・サービスの一般的な使用方法

これらのガイドラインにおける別の観点は一般的に信頼のおける使用方法および選択された分散処理環境のプロファイリングである。これらガイドラインにおいては、それぞれの情報モデルにおいてオブジェクトのネーミングやアラームのフィルタリングのようなネットワーク管理の共通機能を提供するために必要なインタフェース機能を再定義するより、サポートサービスを信頼する。情報モデルはこれらのサポートサービスによってより単純になり、さらに、相互運用を増強することができる。CORBA ベースのインタフェースで必要とされているサポートサービスは、ITU-T Q.816 の中で指定される。

4.1.3　情報モデルの透過性

既存の情報モデル(例えば GDMO)が確立されていおり、CORBA が TMN アーキテクチャーの範疇で使用

される場合、フレームワークはそれらのモデルの再使用を推奨する。

同じサービスセットのアプリケーションには同じアプリケーションプロトコルが使われるために、OMG IDL にこれらの GDMO 情報モデルを写像する単一的な基準が必要である。

4.2 実体

実体のタイプは、実際の世界にある“もの”のように記述する。各実体タイプは、属性と呼ばれる特別の特性を持っている。

実体のインスタンス(あるいは実体)(例えば回路パック#1)は実体タイプ(例えば回路パック)である。各実体の属性は、そのインスタンスの状態を表わす値を持っている。

さらに、各実体は唯一でなければならない。CORBA では、異なるメソッドによって実体がアクセスされるかもしれない。例えば、実体は、IDL データ構造、値タイプあるいはインタフェース・タイプによってアクセスされるかもしれない。この勧告では CORBA がどのように実体タイプを定義するために利用されるかを提示する。

4.2.1 アクセスの粒状性

TMN オペレーション環境では、粒状性が、システム間で露出される抽象のレベルを定義する。アクセスの粒状性は、実体のアクセスレベル(つまり、情報は、どのようにインタフェース経由で晒されるか)を識別する。CORBA については、各 CORBA オブジェクトが、相互運用可能なオブジェクト参照(IOR)として知られているユニークなアドレスが使われる。IOR は、サーバ側 CORBA オブジェクトとのコミュニケーションのためにどのサーバ・システムと接続するべきであるかを識別するクライアント・システムにアドレスを供給する。

CORBA では、TMN(例えば ITU-T M.3100)のために定義された実体とは異なるアクセス抽象的概念(つまりアクセスの粒状性)を定義することが可能である。2 つの異なるアクセス抽象的概念がここに定義される:

1) インスタンス粒状：各実体はそれ自身の IOR を持っている。新しい実体の生成については、これが、新しい CORBA オブジェクトのインスタンス化を意味する。

   -1 IOR/実体のインスタンス

   例えば、ATM 領域の実体タイプは atmLink である。インスタンス粒状のアプローチでは、それが表わす実体タイプと同じ属性を提供する CORBA オブジェクトが定義される。atmLink の各インスタンスについては、独立した CORBA オブジェクトが作成される。したがって、その IOR は各 atmLink をユニークにアドレスすることができる。

2) アプリケーションに特有の粒状性：実体タイプの十分に定義されたセットの実例は単一の IOR(一つのインタフェース)によってアクセスされる。

   -1 IOR/ファミリー(セット)の実体タイプ

   Bulk オペレーションは CORBA IDL インタフェースの中で特有のアプリケーションとして定義される。それらインタフェースのパスの同一性や管理された実体の状態については IDL 構造化されたタイプを使用したパラメータがオペレーションで使用される。

この CORBA オブジェクトをモデル化する為のガイドラインは管理されたインスタンス粒状に透過的にアクセスする為のインタフェースのオブジェクトを特定する為に適用される。TMN 基準もアプリケーションに特有のアクセスの粒状性を使用して定義されるかもしれない。しかしながら、そのようなインタフェース仕様書はこの勧告の範囲の外にある。

4.3 包含の法則と名前付け

包含は、1 つのタイプの実体がどのように別のタイプの実体を含んでいるかの論理的な表現である。包含のツリー構造は、実体インスタンス間の関係を定義する。実体インスタンスはひとつもしくはひとつだけづつの実体インスタンスを含んでいる。

実体インスタンスは有向グラフを形成するフォームを別の実体インスタンスとして含んでいる。有向グラフは、“名前付け(もしくは包含) ツリー”と呼ばれるものを形成する。

包含関係は実際の世界の一部分(例えば会議、サブ会議およびコンポーネント)の階層あるいは実際の世界の組織的な階層(例えば会社名、組織名)をモデル化するために利用することができる。可能な包含ツリーの例は、図 2 の中で下に示される。

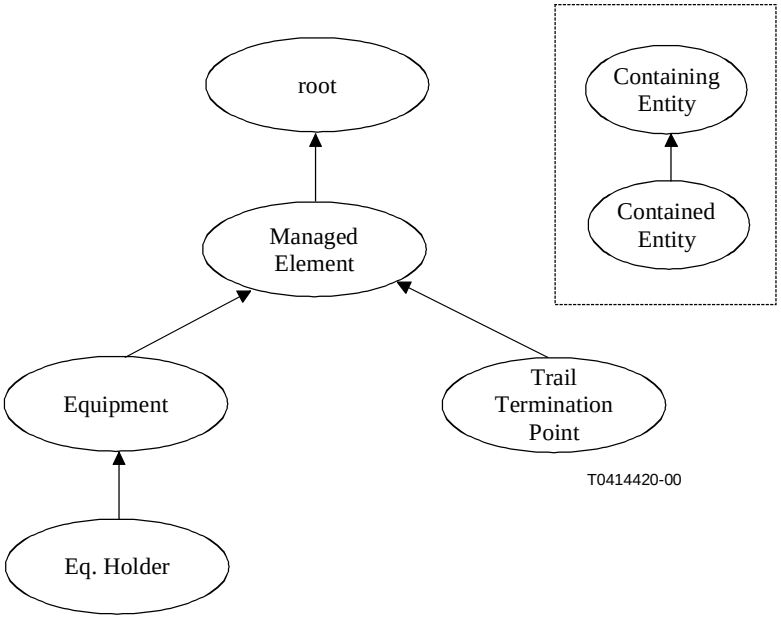

図 2/JT-X780　継承の例

(ITU-T X.780)

4.3.1 名前付け

包含関係の 1 つの目的は実体の指定のためにある。名前は、特定のコンテキスト下において識別できることを目指しており、TMN については、このコンテキストは含まれている実体インスタンスにより決定される。

別の実体のコンテキストの中で名前付けられる実体は“下位の実体”と名付けられる。他の実体のために名前付けのコンテキスト(この用語は一般の中で使用され、サービスを指定するコンテキストと命名する COS の直接の含蓄があってはならない)を確立する実体は、“上位の実体”と呼ばれる。

“下位の実体”は以下のものの組み合わせによって指定される:

- その“上位の実体”の名前。
- その上位の実体の範囲内でこの“下位の実体”をユニークに識別する情報。

ローカルなコンテキストにおける実体の名前は、より大きなコンテキストの名前の中では有効な名前にならないかもしれない。しかしながら、ローカルで名前付けしたコンテキストがより大きなコンテキストにおいてもユニークである場合、ローカルの名前はそのコンテキストの名前においてユニークになることができる。コンテキストの名前は修飾語として使用される。この配置は、オブジェクトからコンテキストの名前を指す、各端(あるいは矢)を備えた有向グラフとして視覚化することができる。

別のコンテキストの名前はコンテキストの名前に対し再帰的に資格を与えることができる、したがって、完全な構造の名前は単一に根を下ろした階層として視覚化することができる。この階層はツリーの名前と呼ばれる。したがって、“上位の実体”はコンテキストの名前になり、また、それらの名前はコンテキストの名前になる。オブジェクト名は、その上位の実体のコンテキスト内で唯一である必要がある;より広いコンテキスト内では、その名前が、その名前によって常に上位の実体の資格を与えられる。

4.3.2 実体の識別

“上位の実体”が同じタイプの多数の“下位の実体”を含んでいるかもしれないので、含まれている各々の同じタイプの実体はお互いに識別できなければならない。包含している実体に含まれる中で実体の相対的な名前は相対識別名(RDN)と呼ばれる。例えば、管理対象の要素内にいくつかの装置ホルダがあるかもしれない。ユニークに管理された要素内の各設備ホルダを識別するために、設備ホルダは RDN をもたなければならない。

RDN は、ITU-T X.780 における実体タイプとして名前と実体を含んでいる範囲を決めなければならない。ITU-T X.720 の中で指定されるように、RDN は絶対識別名(DN)の基礎的な要素である。DN は、特定のコンテキストからスタートする RDN の並びによって定義される。DN はこのコンテキストに関連のあるユニークな名前を与える。

4.4 管理オブジェクトのクラス

モデル化するガイドラインは、各実体タイプが CORBA の運用上のインタフェースで 1 対 1 で対応することを明示する。実体タイプがこの方法で対応付けられる場合、実体タイプを表わす CORBA オブジェクトは Managed Objects クラスと呼ばれる。また、Managed Objects クラスは通知を送る能力を示さなければならない。(ITU-T X.703 を参照)

“Managed Objects クラス”は ITU-T X.720 に定義される。ITU-T X.703 の中で説明されるように、Managed Objects クラスおよびサブクラスはインタフェースおよび派生したインタフェースへ対応付けられる。

4.5 パッケージ

CORBA IDL の中のパッケージについての概念を捉えることは必要である。パッケージは Managed Object Instance に条件付きで提供されるかもしれない能力(属性、アクションあるいは通知)のグループである。管理システムには、どのパッケージが Managed Object Instance によって提供されるかを決定する能力がなければならない。何らかの操作が ある Managed Object に実行されて、それらの操作がその Managed Object のためにインスタンス化されない Conditional パッケージに含まれるならば、Managed Object はエラーを示さなければならない。

4.6 属性

ガイドラインは、Managed Objects クラス上で属性(つまり可視の特性)の定義を提供しなければならない。

4.6.1 Get と Set

属性の値は、標準インタフェースの向こう側に観測可能であるか、あるいは修正可能であるかもしれない。

観測可能であるならば、情報モデラはその属性のために“Get”というメソッドを定義しなければならない。修正可能ならば、情報モデラはその属性のために“Set”メソッドを定義しなければならない。

4.6.2 一般的な属性の取得

CORBA ベースの TMN 情報モデルは管理システムに単一の操作でただ一つの管理オブジェクトから任

意のグループの属性を読む能力を許容するべきである。このサービスは単一の操作で多くの管理タスクを実行させる。Generic Attribute Get のサポートが必要である。

4.6.3 Set で評価された属性

値のリストを含む属性のために、モデラには、オリジナルのリストの中ですべての情報を再送するというわけではなく、個々の値をリストから(に)加えるか、または削除するためにシステムを管理するのを許容する能力があるべきである。

4.7 Managed Objects の生成および削除

Managed Objects(MO) の存在は、密接に MO の包含関係と関係がある。MO の存在は、その MO の上位の MO インスタンスの存在に結び付けられる。指定された“上位の MO”が“下位の MO”のために存在していないならば、その“下位の MO”を作成することができない。同様に、MO の“上位の MO”が削除されるならば、その“下位の MO”(そして、“下位の MO”の下位)はもはや存在することができない。
これを与えられて、TMN CORBA フレームワークによって強化されなければならない
生成および削除セマンティクスがある。

次のサブ節は、オブジェクト生成および削除のためにサポートされるに違いない、ハイ・レベルの必要条件を定義する。ITU-T Q.816 は、かつては総括的なサービスが生成(つまり factory)および削除(つまりターミネータ・サービスを備えた調整での factory)を実行したことを説明する。
第 6 節はこの節で定義される要件がどう提供されるかモデル化ガイドラインを定義する。

4.7.1 生成

Managed Object を生成するとき、MO の存在の 3 つの局面が特定されなければならない:

.. MO の名前;

.. MO の属性値;

.. 新しい MO の生成でインスタンス化されうる MO の条件付きのパッケージ。

create リクエストでのこれらの様相の定義が、明示的である場合も暗黙である場合もありうることに注意すること。MO の存在のこれらの様相の識別に対するオプションは次の 3 つの節に定義される。

4.7.1.1 MO 名の識別

作成される MO の名前は 2 つの方法のうちの 1 つで決定することができる:

1) マネージャがパラメータとして指定するかもしれない、操作、参照を
   新しい MO の上位であることになっている既存の MO に作成して、
   中で新しい MO の RDN を指定するかもしれない、操作の属性リストを作成する。
   これはマネージャによって提供される MO 名前の完全な仕様をもたらす。
2) マネージャは、生成操作のパラメータとして新しい MO の上位であることになっている
   既存の MO の参照を指定して、新しい MO の RDN を指定するのを忘れるかもしれない。
   この場合、新しい MO の RDN は管理されたシステムによって割り当てられる。

関連する情報が正確でないか、create オペレーションが実行することができない
他のある理由のための場合、その後、オペレーションを実行することを試みる factory はエラーを示すものとする。

4.7.1.2 MO 属性の識別

MO が作られる場合、その属性は属性のタイプに有効な値を割り当てられる。これらの値は、Create オペレーション、および以下にリストされた 2 つの方法のうちの 1 つの MO クラス定義中の情報に由来する:

1) create リクエストは、個々の属性に対する明示的な値を指定することを許される。
   MO が作られる場合､明示的な値は MO クラス定義によって求められるような属性に割り当てられる。
2) MO クラス定義は、create オペレーションによってセットされない属性にデフォルト値がどのように割り当てられるか明示することを許される。

デフォルト値が属性に対して指定されない場合、管理するシステムは create リクエストでのその属性に対する値を供給するに違いない。値がその属性に対して指定されない場合、エラーが生じるべきである。

明示的な値が create リクエストの中で特別の属性に対して定義される場合、MO はその属性のために指定されるかもしれないすべての潜在的なデフォルト価値に関する指定された属性に見合うその価値をとるであろう。

4.7.1.3 インスタンス化のための MO パッケージの識別

要求された能力で根本的な資源を実証することができることを保証するために､マネージャーは､Managed Objects が実証するべきだった能力(つまり条件付きのパッケージ)を指定することができなければならない。

関連する条件が実証されている Managed Objects に対して満たされる場合、条件付きのパッケージのインスタンス化が生じるであろう。マネージャーは、さらに、create リクエストのパッケージ属性にそれを含めることにより、create リクエストの一部として条件付きのパッケージのインスタンス化を要求するかもしれない。

4.7.2 削除

削除については、削除セマンティクスでは、包含するすべての実体の削除を提供するかもしれないが、一方で、下位に実体を包含しているために削除メソッドが即座に失敗するかもしれない。これら、セマンティクスは各実体タイプのために維持されなければならない。

4.8 継承

1 つの“Managed Objects クラス”は継承の利用により、別の“Managed Objects クラス”の特殊化として定義されるかもしれない。“Managed Objects クラス”の特殊化は、superclass 上で定義された方法および属性もすべて下位分類にサポートされるだろうということを暗示する。

CORBA IDL では、属性かオペレーションは 1 つを越えるインタフェースから継承することができない。また、下位分類は継承したオペレーションか属性を再定義することができません。(一般に、CORBA 情報モデルがまた、その同じ方法か属性が“スーパー-クラス”で定義されるかもしれないクラスで方法か属性を定義すると予想されないのに注意すること。例えば、属性が指定する GDMO が値を可能にして、必要としたので、GDMO の“サブ-クラス”は時々同じ属性を再定義するかもしれない。注意はそれを IDL に対応させるとき、取られて、同じ属性が再定義されないということであるに違いない。)

CORBA の“サブ-クラス”は 1“スーパー-クラス”から同じ属性か方法(同じ名前がある)を継承することができない(彼らが同じ基底クラスからそれを順番に引き継がなかったならば)。また、“サブ-クラス”はその“スーパー-クラス”の 1 つで定義される同じ属性か方法(同じ名前がある)を再定義することができない。

これらのガイドラインは、CORBA 継承に関する制約を置かない。

５．オブジェクトモデル IDL モジュール

CORBA インタフェース定義言語(IDL)[2]を使用した TMN 管理オブジェクトを定義するための規則について記述する前に、本節で、オブジェクトインタフェース一式を含み、かつ CORBA IDL の中で指定されたデータ構造をサポートするネットワーク管理モジュールを示す。

この IDL モジュールは、CORBA に基づいたネットワーク管理における役割を果たすように意図されており，CMIP に対する ITU-T X.721[6]での GDMO や ASN.1 の定義によってなされる役割に類似している。 それは、情報モデルを構築する際の基となる IDL 定義の基本的な集合を提供する。

IDL は、本標準の付属資料 A および B に含まれている。 付属資料 A は基底クラス(インタフェース)、データ構造および通知を含んでいる。 付属資料 B は定数定義のみを含んだ別個のファイルである。 これらは両方とも、ITU T X.721 での GDMO と ASN.1 の定義に基づいている。

ITU-T X.721 はネットワーク管理情報モデルの中で提供されるべき能力の有用な情報源である。 ITU-T X.721 は、GDMO を使用して以下の管理オブジェクトクラスを定義している:

- 9 のレコード型 (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 とEvent Forwarding Discriminator;
- Log;
- System;
- Top.

これら各々は属性、動作およびサポートするデータ型およびパラメータを有する。 さらに、ITU-T X.721 は 15 の通知を定義する。

上に列挙された管理オブジェクトクラスに着目すると、フレームワークに既に含まれている CORBA Common Object Service によってこれらのうちの多数が網羅されていることは明らかである(TMN CORBA に基づいた TMN サービスの詳細については ITU-T Q.816 を参照):

- CORBA Telecom Event Log Serviceはログレコード保持のための構造を定義しており、したがって、レコードクラスを再定義する必要がない。 (CORBA Telecom Event Log Serviceの使用を明示することにより、TMN CORBAフレームワークがログレコードをオブジェクトではなくデータ構造として扱うことに注意。)
- CORBA Notification Serviceはフィルタリング能力を定義しており、したがって、弁別器およびイベント転送弁別器を再定義する必要がない。
- CORBA Telecom Event Log Serviceは、X.721のLogに相当するものを定義する。

通知の他に、System および Top が残っている。 System は実際にフレームワーククラスでなく、(それが必要な場合)その代わりに一般的な情報モデルに属する。したがって、付属資料 A の中の IDL は“管理オブジェクト”と呼ばれる“top”管理オブジェクトインタフェースを定義している。それは、“Top”と命名される管理オブジェクトクラスがすべての CMIP 管理オブジェクトクラスをサブクラス化するように、他のすべての管理オブジェクトインタフェースをサブクラス化するよう意図されている。

さらに一般的な“factory”オブジェクトも含まれている。 管理オブジェクト factory はオブジェクト生成のために使用される。 (ITU-T Q.816 に定義された CORBA ベースの TMN サービスには、オブジェクト型に依存しないオブジェクト削除を扱う *Terminator* サービスが含まれているが、オブジェクト生成ではオブジェクト生成操作が強くタイプ付けされ得るように、クラス固有の factory によって扱われる。)
通知は 3 番目の IDL インタフェース上で定義される。 さらに、多くの IDL データ型が定義される。 最終的

に、いくつかの IDL プリコンパイラマクロは管理オブジェクトインタフェース仕様を緩和するように定義される。 これについて以下に論する。

5.1 基底（Top）管理オブジェクトインタフェース

付属資料 A に定義された第 1 のインタフェースはすべてのデータ型定義の後に見出される管理オブジェクトインタフェースである。 それは、他のすべてのインタフェースが継承する基底管理オブジェクトインタフェースであるように意図される。 それは、管理オブジェクトインスタンスがすべてサポートすべき能力の集合を定義する。 これらの能力は以下のとおり:

- オブジェクトの名前を返すメソッド。
- オブジェクトのインタフェース(実際のクラス)名を返すメソッド。
- オブジェクトインスタンスによってサポートされる条件付きのパッケージを返すメソッド。
- オブジェクト(被管理資源によって自律的に生成された、管理操作に対応した、もしくは、未知である)の生成源を返すメソッド。
- インスタンスのために削除ポリシーを返すメソッド。 これは列挙された値で、オブジェクトが削除可能でないかどうか、オブジェクトを含んでいない場合のみ削除可能かどうか、削除された場合に含まれる全てのオブジェクトが削除されるかどうかを示す。
- オブジェクトに関する可読な属性をすべて含んでいるCORBA値型オブジェクトを返すメソッド。
- 破壊操作。

ManagedObject インタフェース(コメントのない)を記述する IDL:

```
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 *nameGet*()操作

第 1 の操作である *nameGet()*は、オブジェクトの CORBA 名を返す。 *NameType* は CORBA Naming Service *Name* 型のための型定義である。 *NameType* は、本標準の中で後に定義される IDL 取り決めに適合するために使用される。 このメソッドは、オブジェクトが含まれているローカルなルートネーミングコンテキストに割り当てられた名前から始まるオブジェクトの複合名を返す。

すなわち、このメソッドはオブジェクトに対して“大域的にユニークな”名前を返す。被管理システムのルートネーミングコンテキストにユニークな名前の割り当てについての詳細に関しては、ITU-T Q.816 を参照のこと。 *ApplicationError* 例外は、操作がある資源の問題によって完了できない場合に、任意の管理オブジェクト操作によって上げられるものとして定義されている。 これおよび他のすべての例外についての詳細に関しては、節 5.5 を参照。

5.1.2 *objectClassGet*()操作

*objectClassGet()*操作は、オブジェクトの範囲指定されたインタフェース名(実際のクラス名)を返す。 範囲指定されたインタフェース名は、インタフェースが定義されるモジュールの名前を含んでいる。 返り値型である *ObjectClassType* は、文字列ための型定義である。 オブジェクトのクラスが別のクラス(例えば、“R1”クラス)の軽微な拡張である場合、返された文字列は実際のクラス(“R1”付き)の名前である。例えば、“EquipmentR1”がある。

5.1.3 *packagesGet*()操作

*packagesGet()*操作は、オブジェクトインスタンスによってサポートされた条件付きパッケージのリストを返す。各々文字列の名前を持った条件付きのパッケージについての概念は、これらのガイドラインによってサポートされる。 詳細に関しては、6.6 節を参照のこと。 *StringSetType* は文字列リスト用の型定義である。

このフレームワークが必須のパッケージ定義をサポートせず、条件付きのパッケージのみをサポートするので、これは CMIP オブジェクト上のパッケージの属性とわずかに異なることに注目すること。CMIP では、パッケージ属性は、必須パッケージを列挙することが可能である。必須パッケージの定義がこのフレームワークによってサポートされないので、明白にそれらは管理オブジェクトのパッケージ属性の中にはリストされない。

5.1.4 *creationSourceGet*()操作

*creationSourceGet()*操作は、オブジェクトを生成させたシステムを示す値を返す。 *SourceIndicatorType* は 3 つの値 : *resourceOperation*、*managementOperation*、および *unknown* を備えた列挙型である。 それはオブジェクトが管理操作に応じて資源によって自律的に生成されたかどうか、あるいはそれが未知の場合、なぜオブジェクトは生成されたのかを示している。

5.1.5 *deletePolicyGet*()操作

*deletePolicyGet()*操作は、オブジェクトインスタンスに対して削除するポリシーを返す。 これはオブジェクトが削除可能でないかどうか、オブジェクトを含んでいない場合のみ削除可能かどうか、削除された場合にオブジェクトを含む全てが削除されるかどうかを示す、列挙型の値である。 (含まれているオブジェクトではないオブジェクトの削除は許可されない。) このポリシーは、生成操作で識別されたネームバインディング情報に基づいてファクトリによりオブジェクトが生成される時に設定される。

5.1.6 *attributesGet*()操作

*attributesGet()*メソッドは、一回の操作でオブジェクト属性値のすべてあるいは任意の部分集合を返すために使用される。 情報モデル中の各管理オブジェクトインタフェースについては、インタフェース上で可読な属性の各々に対するデータメンバを含む CORBA *valuetype* が定義される。 (可読な属性とは<attribute name>Get() 操作に伴うもの) このメソッドは、任意の管理オブジェクトに対してこの値型を検索するために使用される場合がある。値型は、管理オブジェクトインタフェース(値型は多重継承をサポートできない以外)の継承階層に従って定義され、各々は *ManagedObject* インタフェースに対して定義された *ManagedObjectValueType* から最終的に引き出される。管理オブジェクトは、このメソッドに応じてそのインタフェースに対して定義された値型を返さなければならない。 したがって、クライアントが任意の管理オブジェクトに対する *attributesGet()*操作を起動した場合、操作が起動されたインタフェースに対して定義された値型に制限すろ（キャスト）*ManagedObjectValueType* への参照を受け取る。

クライアントがインスタンスから属性値すべての検索を望まない場合もあることやインスタンスが条件付きパッケージ中のすべての属性をサポートするわけではないという懸念がこのことを多少複雑にしてい

る。(値型は条件付きパッケージ中の属性を含む。) これは*attributeNames*パラメータを使用することにより対応される。呼び出しにおいて、クライアント、サポートされた属性がすべて返されなければならないといった特別な意味を持つ空のリストを伴って、関心のある属性の名前リストを提出し得る。リストにある有効な属性名ではない名前は、管理オブジェクトによって無視されなければならない。 その応答で、オブジェクトは、値が供給されるための属性の実際のリストを返す。 このリストが提出されたリストと一致しない場合があることを留意すべきである。 たとえ、提出されたリストが空である、または無効の名前があったとしても、オブジェクトは常に正確なリストを返さなければならない。 提出されたリストに載っている名前がすべて無効の場合、オブジェクトは空のリストおよび空の値型を返さなければならない。

値型の構造があらかじめ定義されるので、オブジェクトは要求されていない、またはサポートされていない属性に対して何らかの値を埋めなければならない。基本的に、オブジェクトは、これらの属性に対して何らかの値を返すが、値は効率的にできるだけ短くあるべきである。 したがって、空の値は、ある種の文字列、参照およびいかなる種類のリストに対して返される。いかなる値も整数や列挙型に対して返される。 クライアントは、オブジェクトによって返されたリスト中で指定されない属性に対するいかなる値も無効であることを考慮しなければならない。

基底インタフェース *ManagedObject* は、オブジェクトに対して可読な属性をすべて含んだ CORBA 値型 を返すメソッドのみを現在では有している。 すべての属性が設定できるとは限らないので、属性の設定についての類似したメソッドは含まれていない。

5.1.7 *destroy*()操作

オブジェクトに対する最終操作である *destroy()*操作は、管理オブジェクトに関連した任意の資源を解放し、かつそれを削除するために使用される。 それが *NotDeletable* という削除ポリシーを持っている場合、*DeleteError* 例外はオブジェクトによって上げられる。 *DeleteError* 例外は、モデル依存であるオブジェクトを破壊する問題を報告するための拡張的な手段でもある。 例えば、トレールが削除される前にトレール端点オブジェクトを削除する試みは、結果的に DeleteError となり得る。

しかしながら、ITU-T Q.816 は、削除ポリシーを強化し、ネーミング・ツリーの完全性を維持するために必要なロジックを実装するために“Terminator Service”と呼ばれるサービスを定義する。破壊操作は、実際にはこのサービスによって使用されるように意図され、管理システムによって直接起動されてはならない。Terminator Service についての詳細に関しては、ITU-T Q.816 を参照のこと。

(R) OBJECT-1. 被管理システム上の資源のモデル化に使用されるインタフェースは、上記された、かつ付属資料 A の中の CORBA IDL に定義された *ManagedObject* インタフェースから(直接あるいは間接的に)継承しなければならない。上述の能力がサポートされなければならない。

5.2 ファクトリ管理オブジェクト

時に管理オブジェクトは、被管理システムによって自動的に生成され、時にそれらは別のオブジェクト(ファブリックに対する接続動作に応じて生成されたクロスコネクトオブジェクトのような)に対する動作の結果として生成され、時にそれらはオブジェクトを生成するためのマネージャからの要求に応じて生成される。 この最後の場合においては、CMIP システム上で、生成操作が、CMIP エージェントフレームワークによって典型的に扱われる。 それはまだ生成されていないので、オブジェクト自身によってそれを扱うことができない。

CORBA 実装では、エージェントフレームワークは存在せず、管理システムがオブジェクトを生成することを可能にする、被管理システム上に何かが存在する必要がある。 CORBA システムでは、これが“ファクトリ”オブジェクトによってしばしば扱われる。 *ManagedObjectFactory* インタフェースは、他のファクトリ

インタフェースが継承するところの基底インタフェースであるように意図されている。 それには、すべての管理オブジェクトファクトリがサポートすると期待される能力を定義する。 現在、そのような能力は確認されておらず、そのためインタフェースは空(何からも継承せず、属性またはメソッドを持たない)となっている。 それは将来必要であればその能力が置かれるためのプレースホルダである。 さらに、それはすべてのファクトリに対する共通のスーパークラスの役目を果たす。

管理オブジェクトインタフェース(管理オブジェクトクラスがインスタンス化可能でない限り)に対して CORBA IDL 情報モデルはファクトリインタフェースを含むと期待される。 ファクトリは管理オブジェクトを生成するための操作を含んでいる。 これらの操作は、新たなオブジェクトの上位オブジェクト、新しいオブジェクトの名前および書き込み可能または生成時設定属性等の値といった、多くのパラメータを保持する。 新しいオブジェクトの生成に成功した場合、ファクトリはそれに対して参照を返す。

オブジェクトの生成に加えて、新しいオブジェクトに対して CORBA Naming Service の中でファクトリがさらにネームバインディングを生成することが期待される。 この機能を他のところで実装できるかもしれないが、ファクトリ中でそれを実装することで、管理オブジェクト実装からこの仕事を解放し、資源を代表することに集中させておくことにより実装を単純化すると信じられている。 TMN CORBA フレームワークが CORBA Naming Service をどう利用するかについての詳細に関しては、ITU-T Q.816 を参照のこと。

クライアントがファクトリを見つけるのを手助けするために、ITU-T Q.816 は Factory Finder Service を定義する。 このサービスは、クライアントとファクトリ間のブローカの役割を果たす。 基本的に、ファクトリはそれ自身をサービスに登録する。次に、クライアントは、特定のタイプのファクトリを見つけるための良く知られたサービスを尋ねる。 Factory Finder Service についての詳細に関しては、ITU-T Q.816 を参照のこと。

(R) FACTORY-1. 被管理システム上の管理オブジェクトを生成するために使用されるファクトリオブジェクトは、上述の、あるいは付属資料 A の中の CORBA IDL に定義された *ManagedObjectFactory* インタフェースから(直接あるいは間接的に)継承する。

(R) FACTORY-2. ファクトリはすべて、そのシステム上でインスタンス化された Factory Finder オブジェクト中に登録される。

5.3 通知インタフェース

付属資料 A に定義された第 3 のインタフェースは通知インタフェースである。ITU-T X.721 における通知の各々はこのインタフェースの上に対応する操作を持つ。通知は ITU-T Q.816 によって要求された型付けされたメソッド呼び出しとして定義される。OMG Notification Service は通知をフィルタし、同報するために利用される。 型付けされた通知メソッドは型付けされた通知をサポートする通知サービスと共に直接使用することができる。 型付けされたイベントメソッドと構造化されたイベントの間のマッピングについては ITU-T Q.816 で提供されている。

このインタフェースに定義されている通知操作は多くのパラメータ(それらのうちのいくらかは通知のすべてに共通である)を渡す。 通知のいくつかは同一のパラメータを持っているが、わずかに異なる理由で使用される。 通知インタフェース IDL はこのように見える:

```
interface Notifications {
 void equipmentAlarm (
        in ExternalTimeType               eventTime,
        in NameType                       source,
        in ObjectClassType                sourceClass,
        in NotifIDType                    notificationIdentifier,
        in CorrelatedNotificationSetType  correlatedNotifications,
        in AdditionalTextType             additionalText,
```

```
                in AdditionalInformationSetType    additionalInfo,
                in ProbableCauseType               probableCause,
                in SpecificProblemSetType          specificProblems,
                in PerceivedSeverityType   perceivedSeverity,
                in BooleanTypeOpt                  backedUpStatus,
                in NameType                        backUpObject,
                in TrendIndicationTypeOpt          trendIndication,
                in ThresholdInfoType               thresholdInfo,
                in AttributeChangeSetType          stateChangeDefinition,
                in AttributeSetType                monitoredAttributes,
                in ProposedRepairActionSetType     proposedRepairActions,
                in BooleanTypeOpt                  alarmEffectOnService,
                in BooleanTypeOpt                  alarmingResumed,
                in SuspectObjectSetType            suspectObjectList
                );
…
     }; // end of Notifications interface
```

他の 14 の通知操作は上記のものに類似している。 定義された 15 の通知の名前は次のとおり:

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

他のインタフェースでは要求されない(それらは ITU-T X.733 では要求されない)通知識別子の使用を、本 CORBA フレームワークは要求する。 例証のため、以下にネットワーク要素/EMS インタフェースから EMS/NMS インタフェースへの障害通知識別子のマッピングが行われる際の 4 つの可能性のあるケースを示す:

1) ネットワーク要素は常に通知識別子を使用し、管理オブジェクトは両方のインタフェースにおいて表現される。 この場合、EMSはNMSへ (通知識別子を備えた)障害を渡す。

2) ネットワーク要素は通知識別子を使用せず、管理オブジェクトは両方のインタフェースにおいて表現される。 この場合、EMSは内部カウンタを使用し、この値を通知識別子として含み、NMSへ障害を渡す。

3) ネットワーク要素は時に通知識別子を使用し、管理オブジェクトは両方のインタフェースで表現される。 通知識別子が要求されるので、提供されない場合はEMSは値を定義しなければならない。 通知識別子の値が、相関性が重要となる時間全体に渡って特定の管理オブジェクト・インスタンスのすべての通知に対してユニークでなければならないので、EMSで値を定義することは困難であり得る[1]。 したがって、EMSは、現在の障害で使用されておらず、それに続く後の障害でも使用されない値を選ばなければならない。 通知識別値を選ぶためのアルゴリズムは (値を) 生成するシステム(この場合、ネットワーク要素)によって所有されるため、これを実行する際には特別な注意が必要である。

   1つの可能性のある解決法においては、EMSがすべてのアラームに対して通知識別子に見合う値を供給する場合がある。 これはまた、各々の警報に関連付けられた通知リストの更新を要し、結果的にEMSがネットワーク要素通知識別子値のEMS通知識別子値への完全なマッピングを維持する。

   別の可能性のある解決法では、EMSおよびネットワーク要素は通知識別子番号の異なる部分集合をサポートする点で合意することがあり得る。

もう一つの方法として、EMSはそれ特有の番号を供給し潜在的な衝突を無視し得る、従いそれのまれな発生を許容する。

4) 警報は、1つのネットワーク要素/EMSインタフェースオブジェクトから異なるEMS/NMSインタフェースオブジェクトへマッピングされる。 上の項目と同様に、EMSは、EMS/NMS管理オブジェクトにはユニークな通知識別子値を供給しなければならない。 関連通知リストも更新される必要がある。

5.4 データ型定義

付属資料 A の中のインタフェース定義に先行する部分として、多くのデータ構造および型定義がある。これらのうちのほとんどは通知の中で使用される。 これらは構文を単純化するために、ITU-T X.721 の中の ASN.1 モジュールにわずかな変更を加えて引き出された。 可能な限り、入/出力パラメータや例外のような現代のオブジェクト指向の概念を採用し、これらの型に反映させている。

注意すべきデータ型の１つとして時間型がある。 これらのガイドラインは、CORBA Time Service に対して定義された世界時コードを採用している。 このデータ型は、1582 年 10 月 15 日 0 時以降経過した、何百ナノ秒を数える大きな整数から成る。 世界的な時間を説明するために、時間は差を符号付 short 整数を使用したグリニッジ時間帯の時間と相対的に表現されている。 これは、これらのガイドラインに基づいたシステムが現地のタイムゾーンを知る必要があることを意味している。 このアプローチにより、時間が整数として表わされるので、時間を比較することを容易にしている。 整数表現とよりよく知られたフォーマット間の変換に対する標準ライブラリは、広範囲で利用可能となるであろう。

5.5 例外

付属資料 A での IDL モジュールは、管理オブジェクト操作による使用のためのいくつかの例外を定義する。 下に定義されるように、これらはいくつかの操作で上げられる場合がある。 さらに、標準的な CORBA 例外のうちのいくつかは任意の操作で上げられる場合がある。 例えば、 “CORBA:NO_PERMISSION”例外はセキュリティ違反を示すために上げられる場合がある。 定義された例外は以下のとおり:

```
valuetype ApplicationErrorInfoType {
          public UIDType          error;
          public Istring          details;
};
valuetype CreateErrorInfoType : ApplicationErrorInfoType {
          public MOSetTyperelatedObjects;
          public AttributeSetType attributeList;
};
valuetype DeleteErrorInfoType : ApplicationErrorInfoType {
          public MOSetTyperelatedObjects;
          public AttributeSetType attributeList;
};
valuetype CreateErrorInfoType : ApplicationErrorInfoType {
          public MONameSetType    relatedObjects;
          public AttributeSetType attributeList;
};
valuetype DeleteErrorInfoType : ApplicationErrorInfoType {
          public MONameSetType    relatedObjects;
          public AttributeSetType attributeList;
};

valuetype PackageErrorInfoType : CreateErrorInfoType {
          public StringSetType    packages;
};
exception ApplicationError { ApplicationErrorInfoType info; };
exception CreateError { CreateErrorInfoType info; };
exception DeleteError { DeleteErrorInfoType info; };
```

5.5.1 ApplicationError **例外**

操作が被管理システムでのアプリケーションレベルの条件により完了できない場合、*ApplicationError* 例外が上げられる。例外とともに返された情報は、特定の条件に対する識別子や追加の詳細あるいは説明を備えた文字列を含む。

特定のエラー条件に対する少数の識別子はフレームワークによって定義される。 可能な場合は常に、これらは使用されるべきである。 しかしながら、情報モデルは追加のエラー条件コードを定義するか、あるいは自身に対する例外を生成する場合がある。

アプリケーションエラー例外とともに返されたデータは値型であり、それはそれが拡張される場合があることを意味する。 すなわち、あるエラー条件コードに対して返された実際のデータ型は基底アプリケーションエラー情報型の拡張である場合がある。 エラーコードが基底型の中にあるので、クライアントコードはそれを検査することができ、その値がサブクラス中に戻された一つである場合、クライアントは値型を制限（キャスト）し、追加情報にアクセスすることができる。

ApplicationError 例外は、すべての管理オブジェクトおよび管理オブジェクトファクトリ操作の *raises* 節に含まれている。 アプリケーションエラー例外に対する少数のエラーコード値はフレームワークに対して定義された。 各々は、以下の副節の中で議論する。

5.5.1.1 invalidParameter

ある操作パラメータの値が要求された操作に対して有効でない場合、*invalidParameter* のエラーコードを備えたアプリケーションエラー例外が上げられる。 誤ったパラメータの名前は詳細フィールドへ返される。

5.5.1.2 resourceLimit

操作が、メモリの不足のような被管理システム上のある一時的なエラーにより完了できない場合、*resourceLimit* のエラーコードを備えたアプリケーションエラー例外が上げられる。説明を含んだ文字列は詳細フィールドへ返される。

5.5.1.3 downstreamError

操作が被管理システムから下流のエラーにより完了できない場合、*downstreamError* のエラーコードを備えたアプリケーションエラー例外が上げられる。 この例は、EMS が NE と通信できないために、操作が完了できない場合である。

5.5.2 *CreateError* **例外**

ファクトリ生成操作上でエラーが生じる場合、*CreateError* 例外が上げられる。 それはあらゆる管理オブジェクトファクトリ生成操作の *raises* 節に含まれる。

この例外とともに返されたデータは、一般的な *ApplicationError* のデータを拡張し、関連するオブジェクトのリストおよびオブジェクト（もしそれが生成されていたならば）が有する属性値を追加する。このフレームワークによってこの例外に対して定義された特定のエラーコードを以下に示す。 可能な限り、実装はこれらを使用すべきである。 情報モデルは新しい値を加えるか、あるいは特別の場合に対する新しい例外を定義する場合がある。

5.5.2.1 invalidNameBinding

この状況で生成操作に含まれるネームバインディングがオブジェクトの生成をサポートしない場合、*invalidNameBinding* と等しいエラーコードを備えた生成エラー例外が上げられる。

5.5.2.2 duplicateName

生成操作に含まれる名前が重複である場合、*duplicateName* に相当するエラーコードを備えた生成エラー例外が上げられる。

5.5.2.3 unsupportedPackages

1 つ以上の要求されたパッケージが実装によってサポートされない場合、*unsupportedPackages* と等しいエラーコードを備えた生成エラー例外が上げられる。 このエラーコードが使用される場合、返されたデータ構造が現実に *PackagesErrorInfoType* 構造であり、それは *CreateErrorInfoType* 構造を拡張することを注意のこと。 *PackagesErrorInfoType* 構造は、この場合サポートされていないパッケージのリストを含む。

5.5.2.4 incompatiblePackages

要求されたパッケージのうちのいくつかが、お互いに、あるいはオブジェクト生成のための資源と互換性をもたない場合、*incompatiblePackages* と等しいエラー・コードを備えた生成エラー例外が上げられる。 このエラー・コードが使用される場合、返されたデータ構造が現実に *PackagesErrorInfoType* 構造であり、それは *CreateErrorInfoType* 構造を拡張することを注意のこと。 *PackagesErrorInfoType* 構造は、この場合互換性のないパッケージのリストを含む。

5.5.3 *DeleteError* 例外

削除操作上でエラーが生じる場合、*DeleteError* 例外が上げられる。 それは、すべての管理オブジェクトによって継承されるような基底 *ManagedObject* インタフェース上の破壊操作の *raises* 節に含まれる。

この例外とともに返されたデータは、一般的な *ApplicationError* のデータを拡張し、関連するオブジェクトのリストおよびオブジェクト（削除が試みられた場合）が有する属性値を追加する。このフレームワークによってこの例外に対して定義された特定のエラー・コードを以下に示す。 可能な場合は、実装はこれらを使用すべきである。情報モデルは新しい値を加えるか、あるいは特別の場合に対する新しい例外を定義する場合がある。

5.5.3.1 notDeletable

削除ポリシによって破壊すべきでない管理オブジェクトに対する破壊操作を起動しようとした場合、*notDeletable* と等しい定数値を備えた削除エラー例外が上げられる。 (destroy()管理オブジェクト操作がフレームワークの他の部分での使用のために定義されることを注意のこと。それを直接呼び出す管理システムが、被管理システム上のデータを破壊する危険がある。)

さらに、クライアントが *notDeletable* の削除ポリシーを備えたオブジェクトを削除しようとした場合、Terminator Service はこの例外を上げる。

5.5.3.2 containsObjects

下位オブジェクトおよび *deleteOnlyIfNoContainedObjects* の削除ポリシーを有する管理オブジェクトを削除しようとした場合、*containsObjects* と等しい定数値を備えた削除エラー例外が上げられる。

管理オブジェクトはこの条件の検知に対する責任はなく、Terminator Service が責任を持つ。

5.6 マクロ定義

付属資料 A の以下のインタフェースは、いくつかのマクロの定義である。 これらのマクロは、通知がどのオブジェクトによってサポートされるかを識別するために省略表記を提供する。 情報を受理するような CORBA IDL の制限された能力故に、これらのマクロは有用であろうと感じられていた。

MandatoryNotification マクロは、オブジェクトによってサポートされなければならない通知を識別する。また、*ConditionalNotification* マクロは、特別のパッケージをサポートする場合に管理オブジェクトによって発せられる通知を識別する。 両方のマクロは、操作(操作は通知を伝えるために使用されることを想起すること)の名前、および操作が定義されるインタフェースの範囲指定された名前を識別する引数を取る。*ConditionalNotification* マクロは、さらに 3 番目のパラメータ、通知が属するパッケージの名前を受理する。

通知マクロは何にも拡大しない。 不運にも、IDL はあまりに制限されているためにこの情報を把握する方法を提供することができない。 コメントは生成されても、コンパイラによって直ちに廃棄される。HTML を生成するために使われるフォーマット化されたコメントは、関連する IDL の構築を要求するため、使用できなくなる。

将来CORBA Component Model が解決法を提供すると望まれたが、実装はこれらのガイドラインに間に合うように利用可能とはならない。CORBA Component Modelと整合性の取れたIDLを生成するマクロを修正することが将来あり得る。 しかしながら、今のところ、通知がどのオブジェクトのクラスによって発せられるかについての情報は、これらのマクロによって把握される。

5.7 定数定義

インタフェース仕様書は、常に同じことを意味するために万人により合意された値を有する定数を多く含んでいる。例えば、あるフィールドにおける“1”は信号損失を意味し、“2”はフレームの損失を意味するなどは万人に合意されている。ITU-T X.721 は例外でなく、多くの定数を定義している。 これらは付属資料 B の中の IDL 形式で再定義される。 あらかじめ定義された定数を伝えるために使用されるメカニズムについての詳細に関しては、節 6.11 を参照のこと。

６．情報モデル化ガイドライン

この節は、CORBA 基準の TMN 情報モデルの開発ガイドラインを示す。GDMO の中で指定された既存モデルの変換ガイドラインは、次の節で提供される。

6.1 モジュール

IDL モジュールは、インタフェース、型定義、例外、そして他の IDL 構成物を一まとめにするために用いられる。モジュールはまた、名前空間の描写を提供する；モジュール内で、識別子はユニークでなければならないが、他のモジュールで再使用されるかもしれない。ほとんど全ての場合にモジュールは、ある情報モデルを指定するために使用される構成要素を、グループ化するために使用されるものとする。モジュールは、他のモジュールの中で入れ子となり、そしてモジュールは複数ファイルにまたがってもよい。これらのガイドラインで指定される IDL は、“itut_x780”と名づけられた単一のモジュールの中に含まれる。例：

```
module itut_x780 {
…
}; // end of module itut_x780
```

このモジュールは定数定義のために、サブモジュールを持つ。

6.2 インタフェース

CORBA ネットワーク管理インタフェースによってアクセス可能な各々の entity は、そのために定義される IDL インタフェースを持つものとする。インタフェースは、単一のソフトウェアオブジェクトによって提供されるとみなされる属性とメソッドの集合を一まとめにする。インタフェースは、他のインタフェースから能力を継承してもよく、そして、enitity をモデル化するように定義されたインタフェースは、この勧告の中で定義されている ManagedObject と名付けられたインタフェースを（直接的、或いは間接的に）継承しなければならない。例：

```
interface Equipment : ManagedObject {
…
}; // end of interface Equipment
```

そのようなインタフェースは、“管理オブジェクトインタフェース”と称される。これらのインタフェースをサポートするオブジェクトは、“管理オブジェクト”である。この勧告で定義される ManagedObject インタフェースは、全ての管理オブジェクトインタフェースによって継承される能力の集合を持っているので、各々の管理オブジェクトは、TMN CORBA フレームワークに存在するために、機能の基本集合を実装しなければならない。

情報モデラが直面する一つの問題は、多重継承に対する CORBA の制限された支持である。あるインタフェースが、操作または属性を複数の上位クラスから継承してよいのは、それらが同じ上位クラスから代わる代わる操作や属性を継承する場合だけである。これは“ダイヤモンド”継承として知られ、下記の図 3 のように示される。

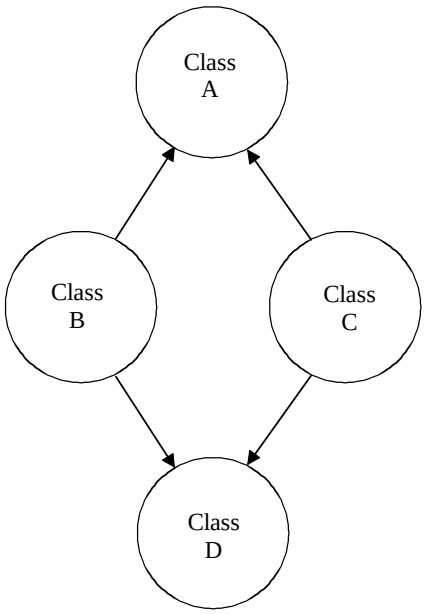

図 3/JT-X780　ダイヤモンド継承

(ITU-T X.780)

もし情報モデラが、一つの共通な上位クラスを共有できない二つの異なるクラスから同じ能力を継承しなければならない事に直面した場合、モデラはクラスを修正して、能力を継承することができる仮想上位クラスを生成しなければならない。例えば、上位に“A”が存在しない場合で、“B”と“C”から“D”を作成する時、モデラは、“B”と“C”よって継承される共通な能力をもつ、新しい仮想クラス（“A”）を生成するように上位クラスを修正しなければならない。

6.3　属性

属性は、属性の値にアクセスするために使用される操作として、インタフェースの中にモデル化される。操作の名前は、入力と出力の型に加えて、操作の型と同様に属性の名前も示す。（CORBA IDL は操作に加えて、属性をサポートする。しかし、この時操作だけが、ユーザ定義の例外をあげることを許されている。これから見られるように、ユーザ定義の例外は属性にアクセスするということに必要とされる。この理由により、操作は単に属性を定義するというよりも、むしろ属性にアクセスすることに定義される。CORBA の将来のバージョンでは、属性にアクセスすることに対して、ユーザ定義の例外を許可するように予定されている。そして、これらのガイドラインはそれを利用するために変更されるであろう。）

6.3.1　読み取り可能な属性

管理オブジェクトは、各々の読み取り可能な属性のために、それらのインタフェースに“<属性名>Get”と名付けられる操作を持たなければならない。この操作によって返される型は、属性の型を反映する。例:

```
AdministrativeStateType administrativeStateGet()
        raises (ApplicationError);
```

稀ではあるが、設定可能だが読み取り不可能という属性は、そのインタフェースに読み取りの操作を定義してはならない。

莫大な量のデータを返すかもしれない属性の取得操作は、クライアント・システムが情報のリターン・フ

ローを制御できるように、イテレータを定義しなければならない。イテレータの使用例は、ITU-T Q.816 を参照。

6.3.2 設定可能な属性

管理オブジェクトは、各々の設定可能な属性に対して、“<属性名>Set”と名付けられる操作を持たなければならない。操作の戻りの型は void でなければならず、入力パラメータは属性の型を反映しなければならない。例:

```
void administrativeStateSet (in AdministrativeStateType adminState)
        raises (ApplicationError);
```

設定不可能な属性は、そのインタフェースにこのような操作を持ってはならない。

6.3.3 値の集合属性

多くの管理オブジェクトの属性は、値の集合を含んでもよい。これらの場合、これまでに定義された操作は、ここでもサポートされるべきである（属性が読み取り可能でかつ、または書き込み可能な場合）。CORBA は明示的に、値の設定に対する複雑な型を定義していないので、これらの操作に対する入力または戻りの型が CORBA シーケンスとなる。これらの属性に対して返された値は重複した値を含んではならない。また、値の順序は重要ではない。さらに、これらの属性への値の追加または削除をサポートすることは必要であろう。これらの操作は“<属性名>Add”そして“<属性名>Remove”として名付けられなければならない。これらの操作の戻りの型は void にすべきである。また、各々への入力パラメータは属性の型を反映した順序であるべきである。例：

```
void supportedByObjectsAdd (in ManagedObjectSetType objects)
        raises (ApplicationError);
void supportedByObjectsRemove (in ManagedObjectSetType objects)
        raises (ApplicationError);
```

6.3.4 例外

属性アクセス操作はまた、例外を上げてもよい。次の例外が属性アクセス操作にて上げられると定義される。:

1) *ApplicationError*。この例外は、属性アクセス操作を含む、すべての管理オブジェクト操作のraises句に含まれるものとする。それは、範囲外の値や管理対象システムなどに対する資源限界などのような、多くの条件を示すために使用されてもよい。

2) 条件付きのパッケージ例外。属性が条件付きのパッケージの一部である場合、その条件付きのパッケージのために定義された例外は、属性アクセス操作のraises句に含まれるものとする。属性にアクセスしようとした時に、その属性が属しているパッケージがインスタンスによってサポートされていない場合に、上げられる。これ以降の6.6節の条件付のパッケージで、さらに参照すること。

これらに加えて、実装ではまた、標準の CORBA 例外の中の何を上げてもよい。例外を上げる操作は、属性の値を修正しないものとする。例外を上げる属性アクセス操作の例は次のとおりである:

```
void supportedByObjectsRemove (in ManagedObjectSetType objects)
        raises (ApplicationError);
```

6.3.5 標準の属性

管理オブジェクトは資源をモデル化し、しばしば、管理オブジェクトの間に共通性がある。これは、オブジェクトのクラス間の継承関係を利用して、ときどき表わされるが、継承関係が存在しない場合にも、オブジェクトの間に共通性があるであろう。この良い例は、相似な属性である。多くの管理オブジェクトは相似な属性を持っている。管理インタフェースの実装をより容易にするために、これらのガイドラインは、可能な場合は常に、属性のために使用されるべきいくつかの標準のデータ型を定義する。すなわち、モデラは、新しい型を定義する代わりに、これらの型定義を使用するように試みるべきである。さらに、属性の名前および操作にアクセスするための操作の名前も使用されるべきである。実際、新しいモデルを定義する場合、可能な場合は常に、既存モデルから属性の型および名前を再使用するのが適切な方法である。標準の属性は表 1 に定義される:

表 1/JT-X780　標準属性

(ITU-T X.780)

データ型	属性名	アクセスメソッド
AdministrativeStateType	administrativeState	administrativeStateGet()
AvailabilityStatusSetType	availabilityStatus	availabilityStatusGet()
BackedUpStatusType	backedUpStatus	backedUpStatusGet()
ControlStatusSetType	controlStatus	controlStatusGet()
SourceIndicatorType	creationSource (注　参照)	creationSourceGet()
DeletePolicyType	deletePolicy (注　参照)	deletePolicyGet()
ExternalTimeType	externalTime	externalTimeGet()
NameType	name (注　参照)	nameGet()
ObjectClassType	objectClass (注　参照)	objectClassGet()
OperationalStateType	operationalState	operationalStateGet()
StringSetType	ackages (注　参照)	packagesGet()
ProceduralStatusSetType	proceduralStatus	proceduralStatusGet()
StandbyStatusType	standbyStatus	standbyStatusGet()
SystemLabelType	systemLabel	systemLabelGet()
UnknownStatusType	unknownStatus	unknownStatusGet()
UsageStateType	usageState	usageStateGet()
注 – これらの属性は、全ての管理オブジェクトに継承される。		

6.4 アクション

属性に加えて、多くの管理オブジェクトが actions －属性へのアクセスとは別の目的のメソッドーを持つであろう。これらの操作のためのパラメータと戻りの型は単にアクションの要求を満たすために定義される。操作の名前は、操作の目的を反映すべきである。次の例外がアクション・操作で上がるものとして定義される:

1)　*ApplicationError*。この例外は、アクション操作を含むすべての管理オブジェクト操作の raises 句に含まれるものとする。それは、範囲外の値や管理対象システムなどに対する資源限界などのような、多くの条件を示すために使用されてもよい。

2)　条件付きのパッケージ例外。アクションが条件付きのパッケージの一部である場合、その条件付きのパッケージのために定義された例外は、アクション・操作の raises句に含まれるものとする。アクションを実行しようとした時に、そのアクションが属しているパッケージがインスタンスによってサポートされていない場合に上げられる。これ以降の6.6節の条件付きのパッケージで、さらに参照すること。

これらに加えて、実装ではまた、標準の CORBA 例外の中の何を上げてもよい。アクションに特有の他の例外は、他のエラー条件のために定義されてもよいし、定義されるべきである。代わりに、情報モデルは *ApplicationError* 例外のために定義されたエラー・コード・ポイントを拡張してもよい。

大量のデータを返すアクションは、クライアント・システムが情報のリターン・フローを制御できるように、イテレータを定義しなければならない。イテレータの使用の例については、ITU-T Q.816 を参照すること。

6.5 通知

ほとんどの管理オブジェクトが、ある条件の下で通知を発すると予想される。TMN CORBA フレームワークでは、通知が通知サービスの助けを借りて、管理するシステムに管理オブジェクトからのメソッド呼び出しによって伝えられる。したがって、通知操作は、管理オブジェクトのインタフェースではなく、実際には管理するシステムの CORBA インタフェースのために定義される。これらのガイドラインでは多くの標準の通知を定義する。しかし、新しい通知を定義しなければならない場合、それは情報モデルのモジュール内の“Notification”と名付けられるインタフェースに対する操作として定義されるべきである。その操作の名前は、通知の名前でなければならない。操作へのパラメータは、通知の中で報告されるデータを反映しなければならない。通知操作の戻りの型は void にならなければならなく、また、それは“in”パラメータしか持ってはならない。通知操作定義の前に“oneway”キーワードが使用されてはならないことに注意すること。これらのガイドラインに沿った通知は確認される。すなわち、管理オブジェクトがチャネルへ通知を送る時、その通知を受信したことは、チャネルによって管理オブジェクトに戻って確認される。同様に、チャネルが各受信者のもとへ通知を送るので、確認はチャネルによって得られる。ITU-T Q.816 の中で指定されたサービスの品質保証は、チャネル自体の信頼性を定義する。したがって、受信者への通知の配信は保証することができる。

どの管理オブジェクトがどの通知を発するかを文書で記述する方法も必要である。IDL ファイルの中のコメントによって、単にこれを記述するのではなく、マクロ文が使用される。実際に、これらのガイドラインは二つのマクロを定義しており、一つは通知が必須な場合で、もう一方は通知が条件付きのパッケージの一部である場合である。マクロは、管理オブジェクトのインタフェースの中に使用されるように意図されていて、以下のように定義される:

```
MANDATORY_NOTIFICATION(<interface name>,
        <notification operation name>);
CONDITIONAL_NOTIFICATION(<interface name>,
        <notification operation name>, <package name>);
```

例:

```
interface Equipment : ManagedObject {
…
MANDATORY_NOTIFICATION(itut_x780::Notifications, objectCreation);
CONDITIONAL_NOTIFICATION(itut_x780::Notifications,
        equipmentAlarm, equipmentAlarmPackage);
…
}; // end of Equipment interface
```

条件付きの通知マクロの中で使用されるパッケージの名前は、他のところに使用されるのと同じものである。詳細に関しては、節 6.6 のパッケージを参照すること。現実には、CORBA IDL に適切な代案がないので、実際にはマクロは何も展開されない。したがって、マクロは文書化の目的のためであり、現実にはコード生

成に帰着しない。さらなる研究項目としては、オブジェクトにサポートされる通知を識別する IDL を生成するためにマクロを修正していく事である。CORBA コンポーネントモデル仕様のリリースは、そのモデルと一致する方法で、この事を行う機会を提供する。一つの通知だけが各マクロにリストされてもよい。これはマクロの可能な将来の修正をより単純にすることである。

6.6 条件付きのパッケージ

これらの情報モデル化ガイドラインは、管理オブジェクトのクラスのために定義された全ての能力が、全てのインスタンスによってサポートされる必要はない、という考えを支持する。実際、能力のグループは、すべての能力がサポートされるか、あるいはどの能力もサポートされないかのどちらかになるように、定義することができる。これらの能力のグループは package と称される。IDL の中でパッケージを表わすことに対する選択は限定されている。各パッケージに対して個別のインタフェースを定義することは、あまりにも多くのインタフェースを発生させる。したがって、代わりにここに記述されたアプローチが使用される。
条件付きのパッケージの一部である各々の操作は、パッケージのために定義された例外を上げてもよい。例外の名前は“NO<パッケージ名>”とする。例:

```
exception NOadministrativeStatePackage {};
…
AdmininstrativeStateType administrativeStateGet()
        raises (NOadministrativeStatePackage);
```

条件付きのパッケージの一部として発せられる通知は、上に記述されるような CONDITIONAL_NOTIFICATION 文で表される。

能力がパッケージの中に含まれている時に関係のある規則は、サポートされるべきである。また、それらが含まれていない時は、管理オブジェクトのインタフェースと関係するコメントの中に置かれる。操作は、その raises 句の中の多数の NO<パッケージ名>例外のリストにより、一つ以上の条件付きのパッケージに含まれてよい。例外は、パッケージのどれも存在しない場合のみ、上げられるであろう。そして、パッケージ例外の中のどれでも上げられるであろう。操作が必須な場合、操作はその raises 句の中にパッケージ例外をリストしてはならない。通知は、複数のパッケージを CONDITIONAL_NOTIFICATION マクロの中にリストしてもよい。

6.7 振舞い

CORBA IDL は、オブジェクトの振舞いを捕らえる正式な手段を欠いている。今後、情報モデルが UML で文書化され、ユースケースおよびオブジェクト関係図を含むようになる可能性はある。しかしながら、IDL はコメントに限定されている。したがって、必要或いは有用な場合は、コメントをオブジェクトの振舞いについて記述するために使用しなければならない。

この勧告にある IDL は多くのコメントを含んでいる。それらは、より容易に判読できるように IDL を HTML に変換する為に使用されるコンパイラが、解析できるフォーマットになっている。フォーマットされたコメントは、 /** で始まり */ で終わり、そして次の IDL 構成部分に関係している。IDL から HTML へのコンパイラによって追加フォーマットに変換されるキーワード(“@”シンボルが頭に付くもの)と同様に、HTML のフォーマット・タグは、これらのコメントによって許されている。HTML ブラウザにて IDL を見るのは便利であるが、上に記述されたマクロの使用がこれによって影響を受ける事に注意すること。マクロの展開が HTML への変換の一部として実行されるので、あらかじめ展開されたマクロの情報が失われてしまう。したがって、各管理オブジェクトによりサポートされる通知を識別するために使用されるマクロは展開されているであろう。

6.8 ネームバインディング情報

包含はネットワーク管理で非常に重要な関係である。TMN CORBA 基準のフレームワークでは、包含が名前によって表わされる。これは、不運にも、恐らく存在することができる包含関係に対して制限を一切かけない。例えば、ネットワーク・オブジェクトが接続オブジェクトによって含まれることを防ぐものは何もない。はっきりと、可能な包含関係を意識したものだけに制限する方法が望ましい。しかしながら、これらの制限は情報をモデラの制御の下で、広げられてしまうに違いない。

これらの要求を満たすために、これらのガイドラインは、CORBA 基準の TMN 情報モデルを指定する IDL モジュールが管理オブジェクトクラスの可能な包含関係を定義する情報をさらに含むことを必要とする。この包含関係情報は、managed object name binding information として参照される。(不運にも、CORBA 名前サービスに保持されるネームバインディング情報と、容易に混同されるかもしれない。これら２つは同じではない。)

管理オブジェクトのネームバインディング情報は、次の約束事を使用して、CORBA IDL の中で表わされる:

1) 各情報モデルのIDLモジュールは、管理オブジェクトのネームバインディング情報のために“NameBindings”と名付けられたサブモジュールを含むものとする。

2) このネームバインディングモジュールの内では、サブモジュールが各々許可された包含関係のために定義されるものとする。

3) 各ネームバインディングサブモジュールは、これらの7つの定数に値を割り当てるものとする。;

```
const string                superiorClass
const boolean               superiorSubclassesAllowed
const string                subordinateClass
const boolean               subordinateSubclassesAllowed
const boolean               managersMayCreate
const DeletePolicyType      deletePolicy
const string                kind
```

superiorClass 定数は、上位の(含んでいる)オブジェクトの範囲指定されたクラス名を含んでいる。オブジェクトが管理されるシステム上の“最上位の”オブジェクトかもしれない場合、すなわち、それがローカルなルート名前コンテキストの直下に含まれているかもしれない場合、superiorClass ネームバインディングの値は空の文字列であるものとする。superiorSubclassesAllowed 定数は、このネームバインディングを使用して、上位クラスの型のサブクラスがアクセス可能な場合に真の値を持つ Boolean フィールドである。subordinateClass 定数は、下位のオブジェクト(作成されるオブジェクト)の範囲指定されたクラス名を含んでいる。subordinateSubclassesAllowed 定数は、下位のオブジェクトのサブクラスが、このネームバインディングを使用して作成されるかどうかを示す。managersMayCreate フラグは、オブジェクト生成がこのネームバインディングを使用して、管理インタフェースを超えてサポートされるかどうか示す。このフラグに偽をセットする値は、たとえ下位オブジェクトが管理されるシステムによってのみ作成されても、それが包含関係情報をすべてIDLに文書化することができることである。deletePolicy定数は、オブジェクトが生成される場合に管理オブジェクトの deletePolicy 属性に割り当てられる値を含んでいる。kind 定数は、オブジェクトが生成される場合にオブジェクトのために、CORBAネームバインディングの中の kind フィールドに割り当てられる値を含んでいる。

ネームバインディングで kind フィールドのために選ばれる値は、一般的には範囲指定されない下位クラスの名前になるであろう。(範囲指定されないクラスの名前は、一般的に名前の長さを縮小するために使用されるであろう。) kind フィールドの主な目的は、名前の衝突が発生しないように名前空間を分けることである。既存のインタフェースの新バージョン用のネームバインディングモジュールは、より古いインタフェースのために使用される kind 値を再使用してもよい。例えば、EquipmentとEquipmentR1のインタフェース用のネームバインディングモジュールは、両方とも値“Equipment”を使用するかもしれない。しかしながら、さもなければ、インタフェースの各クラスに対するユニークな値を使用することが、恐らく最も安全であろう。

4) ネームバインディングサブモジュールの名前は、<下位クラス>_<上位クラス>となるべきであり、ここで<下位クラス>はモジュール内で、subordinateClass定数に割り当てられる値であり、<上位クラス>はsuperiorClass定数に割り当てられる値である。もし、同じ親モジュールの中で二つのネームバインディングモジュールが同じsuperiorClassとsubordinateClassの値を共有するが、他

の値が異なる場合は、モジュールのうちの一つの名前は、二つの間の違いを表す言葉が追加されるものとする。例えば:“Equipment_Equipment”および“Equipment_Equipment_NotDeleteabe”である。

管理オブジェクトネームバインディングのいくつかの例:

```
module itut_m3120 {
…
  /** The following module contains name binding information */
  module NameBindings {
         /** This name binding module allows Equipment objects to be
         created under Managed Element objects.
         */
         module Equipment_ManagedElement {
                  const string       superiorClass = "itut_m3120::ManagedElement";
                  const boolean      superiorSubclassesAllowed = TRUE;
                  const string       subordinateClass = "itut_m3120::Equipment";
                  const boolean      subordinateSubclassesAllowed = TRUE;
                  const boolean      managersMayCreate = TRUE;
                  const DeletePolicyType deletePolicy =
                                     itut_x780::DeleteOnlyIfNoContainedObjects;
                  const string       kind = "Equipment";
         }; // end of Equipment_ManagedElement name binding module
         /** This name binding module allows Equipment objects to be
         created under other Equipment objects.
         */
         module Equipment_Equipment {
                  const string       superiorClass = "itut_m3120::Equipment";
                  const boolean      superiorSubclassesAllowed = TRUE;
                  const string       subordinateClass = "itut_m3120::Equipment";
                  const boolean      subordinateSubclassesAllowed = TRUE;
                  const boolean      managersMayCreate = TRUE;
                  const DeletePolicyType deletePolicy =
                                     itut_x780::DeleteOnlyIfNoContainedObjects;
                  const string       kind = "Equipment";
         }; // end of Equipment_Equipment name binding module
  }; end of name binding module
}; end of itut_m3120 module
```

deletePolicy 定数は、enum 型であり CORBA IDL の定数定義の規則に沿っている。もし、この型が他のモジュールに定義されているならば、定数に割り当てられた値は、そのモジュールに範囲指定されなければならない。DeletePolicyType はモジュール itut_x780 に定義されている。また、例の IDL モジュールは itut_m3120 である。従って、DeleteOnlyIfNoContained の値は、文字列“itut_x780::”をその前に挿入することにより範囲指定しなければならない。DeletePolicyType という型自体も範囲指定されなければならない。これは、モジュールの初めの typedef 文で処理するができる。

6.9 ファクトリ

TMN CORBA 基準のフレームワークは、オブジェクトを削除するためにサービスを定義するが、オブジェクトはクラス固有の factory で生成される。ファクトリは生成するために使用されるオブジェクトとは異なるが、通常は関連するインタフェースを備えたオブジェクトである。管理オブジェクトの各クラスはまた、ファクトリクラスを持つであろう。これが実行されると、その結果ファクトリ生成操作は、強く分類され、それらが作成するオブジェクトのクラスに特有になるかもしれない。この結果は、管理オブジェクトインタフェースを定義する IDL モジュールがオブジェクトを生成するために使用されるファクトリのためのインタフェースも含むということである。ファクトリ IDL インタフェースの名前は“<管理オブジェクトクラス名>Factory”であるものとする。

この勧告は、各ファクトリのインタフェースが継承しなければならない基本的な管理オブジェクトファクトリのインタフェースを定義する。ファクトリはそれらが生成するオブジェクトと同じ継承階層に従わない。ファクトリは、ManagedObjectFactory のインタフェースを継承するだけである。ファクトリインタフェース定義の例は次のとおりである。:

```
interface EquipmentFactory : ManagedObjectFactory {
…
}; // end of EquipmentFactory interface
```

ファクトリはオブジェクトのサブクラスを生成することができないので、新しいファクトリは各サブクラスのために定義されなければならない。
マネージャがインスタンスを生成することを可能にするネームバインディングモジュールがないとしても、すべてのインスタンス化可能なクラスは、そのために定義されるファクトリを持つものとする。これはマネージャがインスタンスを生成することを可能にするネームバインディングモジュールの将来の定義を考慮に入れてある。

6.9.1 生成操作

各ファクトリインタフェースは、クライアントがオブジェクトを生成するために使用するための単一の操作を定義するものとする。この操作の名前は“create”であるべきで、それは、ファクトリによって作成されるオブジェクトの型への参照を返すものとする。全ての生成操作に対する最初の 4 つのパラメータは、常に同じである。これらの後に、管理オブジェクトのために定義された各書き込み可能な属性、あるいは生成によって設定される属性が続く。(生成によって設定される属性は、“set”操作を持っていないオブジェクトとのためのもので、生成操作で指定される値である。) これらのパラメータの名前は属性の名前と同じである。(これは、属性アクセス操作の名前から、終わりの“Get”あるいは“Set”を引いたものである。)
各生成操作はまた、ファクトリによって生成されるオブジェクトの全ての上位クラスのどの書き込み可能な属性または生成によって設定される属性の値に、パラメータを設定することを許容しなければならない。ここに、equipment factory の生成操作の例がある:

```
Equipment create(
        in NameBindingType nameBinding,        // module name containing NB info.
        in ManagedObject superiorObject,// Reference to containing object.
        inout string name,           // In/out, may be null if auto-create.
        in StringSetType packages,   // List of packages requested.
        …                                    // Writeable and set-by-create values
                                             // for Equipment superclass attributes.
        …                                    // Writeable and set-by-create values
                                             // for Equipment attributes.
);
```

6.9.1.1 ネームバインディング

ネームバインディングパラメータは、6.8 節に記述されるように、管理オブジェクトネームバインディングを含んでいるモジュールの名前を伝える。値の例は、“itut_m3120::NameBindings::Equipment_Equipment”であろう。これを与えられて、ファクトリは、値が有効なネームバインディングの識別子かどうかチェックすることができる。(ファクトリは、システムがコンパイルされる場合に、有効なネームバインディング情報でハード・コードされる、あるいは、それはランタイムで CORBA インタフェース・リポジトリの情報にアクセスするかもしれない。) ネームバインディング情報を見つけることができない場合、ファクトリは引き数として“nameBinding”を返して、invalidParameter ApplicationError の例外を上げるものとする。(これは invalidParameter へ設定される

errorコードと“nameBinding”へ設定されるdeltails文字列を備えたApplicationError例外である。） ネームバインディング情報を見つけることができるが、不完全な場合、 ファクトリは invalidNameBinding CreateErrorの例外を上げるものとする。

ファクトリは、さらにネームバインディングモジュールの中で指定された下位クラスの型が、生成するオブジェクトの型と一致するかどうかをチェックしなければならない。それが一致しない場合、ファクトリは作成するオブジェクトの型が、下位クラスの定数値のサブクラスかどうかチェックすることができる。それがサブクラスである場合、および subordinateSubclassesAllowed 定数が真の場合、ファクトリはオブジェクトを生成することができる。そうでなければ、invalidNameBinding CreateError例外を上げることにより、リクエストを拒絶することになる。

最後に、ネームバインディングモジュールの中で managersMayCreate 定数が偽の場合、ファクトリはまた invalidNameBinding CreateError 例外を上げることにより、リクエストを拒絶するであろう。(ファクトリは、この値をチェックしない、そして管理インタフェースを横切って公開されない管理対象システムによって内部的に使用される第二の生成操作を持つであろう。） falseにセットされる managersMayCreateの値をもつネームバインディングモジュールは、たとえオブジェクトが管理対象システム自身のみによって生成されるとしても、GDMOで可能なように、IDLの包含情報の全てを捕らえること可能とする。

ネームバインディングモジュール内の他の情報は、オブジェクトを生成する時のファクトリとCORBA名前サービスのネームバインディングに使用される。deletePolicy 定数は、新しい管理オブジェクトの同じ名前の属性に割り当てられるであろう。kind 定数値は、ファクトリが CORBA 名前サービスの中で、管理オブジェクトのネームバインディングを生成する時に、使用されるであろう。

6.9.1.2 上位オブジェクト

create 操作中の第二のパラメータは、上位オブジェクト(新しいオブジェクトはその下に作成されることになっている)への参照である。標準の CORBA の能力を使用して、ファクトリはネームバインディングモジュールの中に定義された superiorClass 定数の中に指定された型と一致するかどうか確定するために、上位オブジェクトのクラスを調べる。それが一致しない場合、ファクトリは、供給された参照がsuperiorClass 定数の中で指定された型のサブクラスであるかどうか次にチェックしなければならない。それがそうである場合、そしてネームバインディングの中の superiorSubclassesAllowed 定数が true の場合、ファクトリはオブジェクトを生成し始めてよい。そうでなければ、ファクトリは invalidNameBinding CreateError 例外を上げることにより要求を拒絶し、その詳細の中に“superiorObject”を返してもよい。

もし、ネームバインディングモジュールの中の superiorClass 定数が空の文字列である場合、下位クラスのオブジェクトは、上位オブジェクト(親)なしで生成されるであろう。また、それらの名前は、ローカル・ルート名前コンテキストと直接接するであろう。通常、これらのオブジェクトは管理対象システムによって生成される。しかし、これらの場合では上位オブジェクトの参照はヌルであろう。

6.9.1.3 名前

第3のパラメータは新しいオブジェクトに割り当てられる名前である。この文字列は、新しいオブジェクトのためにCORBA名前サービスの中で生成されるCORBAネームバインディングの IDフィールドになるであろう。これは上位のオブジェクトの名前に関連のあることになる。パラメータが inout の場合は、ファクトリが自動名前付けをサポートしなければならない。この場合は、クライアントは名前にヌル文字を提供してもよい、そしてファクトリは適切な文字を選択し、選択した値を戻すであろう。もしクライアントが文字列を提供する場合は、ファクトリはこの値を代わりに使用すべきである(そして、out 値としてそれを返す)。もしパラメータが in だけの場合、自動名前付けはサポートされず、クライアントは名前を提供しなければならない。もし名前を提供しない場合は、ファクトリは badName CreateError 例外を上げるものとする。

ファクトリは、もし提供された名前が重複していたならば、duplicateName CreateError 例外を上げる。(これは ID と kind の両フィールドが、上位オブジェクトによって包含される既存のオブジェクトと一致することを意味する。)

6.9.1.4 パッケージ

パッケージ属性は重要である。それは、インスタンスがサポートしなければならないパッケージだけをファクトリに伝えるのではなく、生成操作上でそれが無視しなければならないパラメータの値もファクトリに伝える。パッケージは強固に型定義されているので、たとえ属性が条件付きのパッケージの一部であっても、生成メソッドはオブジェクトの書き込み可能な、または生成によって設定可能な各々の属性のためのパラメータを含んでいる。たとえ、ファクトリがとにかくパッケージにてオブジェクトをインスタンス化しても、ファクトリは、クライアントによって要求されていないパッケージ内の任意の属性の値を無視しなければならない。(ファクトリがクライアントによって要求されていないパッケージでオブジェクトをインスタンス化した場合、ファクトリは初期の値を選ばなければならない。) これによって、クライアントは、パッケージ内の属性のために望まない値を提供しなければならないことから解放される。代わりに、クライアントはどんな値も提出することができる。効率のために、クライアントによって要求されないパッケージでの属性のために提出される値は short であるべきである。

もしクライアントがパッケージ・パラメータ中の無効なパッケージ名を提供する場合は、ファクトリは unsupportedPackage CreateError 例外を上げて、引数としてパッケージの名前を返すものとする。もしクライアントがインスタンスの生成を要求するが、同じインスタンスに共存してはならないパッケージを指定する場合は、incompatiblePackages CreateError 例外をまた上げてもよい。

6.9.1.5 上位クラス・パラメータ

これらの最初の 4 つのパラメータに続くものは、ファクトリによって生成されるオブジェクトの型の、あらゆる上位クラスのための、書き込み可能な属性と生成によって設定される属性の各々のパラメータであろう。

6.9.1.6 オブジェクトクラス・パラメータ

最後に、上位クラス・パラメータに続くものは、ファクトリによって生成される管理オブジェクトクラスのための、書き込み可能な属性と生成によって設定される属性の各々のパラメータである。

6.9.2 ファクトリ・ファインダ

ファクトリを見つけるタスクを緩和するために、ITU-T Q.816 はファクトリファインダ・インタフェースを定義する。(ファクトリ・ファインダは CORBA アプリケーションの共通のデザインパターンである。) このことにより、クライアントは、管理対象システム上に存在するすべてのファクトリについての情報を持った、ブローカとの対話により、容易にファクトリを見つけることができる。

6.10 管理オブジェクトクラス Value Type

これらのガイドラインに従った個々の管理オブジェクトクラスは、単一の valuetype の中の対象属性の部分集合もしくは全てを返すベース管理オブジェクトクラスからの操作を継承する。(CORBA 2.3 は、値型(参照の代わりに値によって渡されるオブジェクト)の概念を導入する。)管理オブジェクトの実装はこの特徴をサポートしなければならないだけでなく、管理対象を記述した IDL は、管理対象によってサポートされた各々の属性用としてパブリック属性の値型を含まねばならない。これらのガイドラインはベース ManagedObjectValueType を定義する。また、管理オブジェクトのために定義された値型は、最終的にこのベー

ス値型の派生になる。管理オブジェクトのために定義された値型は、通常管理オブジェクトインタフェースの継承パターンに続くべきである。しかし、CORBA の値型が単に単一の継承をサポートするので、これは必ずしも可能になるとは限らない。しかしながら、これは重大な制限とはならない。多重継承を用いてインタフェース用に定義された値型は、上位の値型の一つから単一に継承され、他の属性は手修正するのみである。例として、ベース管理対象クラスから直接継承する Equipment 管理対象は、特に UserLabelType 型を返す userLabelGet という属性アクセス関数を保持する。Equipment 管理オブジェクトの値型を定義した IDL は以下のように記述する。

```
valuetype EquipmentValueType : ManagedObjectValueType {
   public UserLabelType         userLabel;
   …                                    // other attributes
};
```

値型の名前は“ValueType”で追加されたインタフェースの名前である。値型の中のパブリック属性名は、“Get”なしの、属性にアクセスする管理対象インタフェース上のメソッド名となる。このしきたりは値型中のすべての属性に適用する。属性のタイプは属性アクセス機能によって返されたタイプと同じである。

Equipment オブジェクトに対する属性価値を検索したいクライアント側のコードは、以下のようになる。:

```
ManagedObjectValueType         moValue;
EquipmentValueType   eqValue;
Equipment                      eq;
eq = …    // code that sets eq to a CORBA proxy representing an
          // equipment object.
moValue = eq.getAttributes();
eqValue = (EquipmentValueType) moValue; // cast return to proper type
System.out.println("User Label = " + eqValue.userLabel); // print label
```

IDL がオブジェクト指向のプログラミング言語へコンパイルされる時、インタフェース(この場合 Equipment)および値型(ManagedObjectValueType と EquipmentValueType)の両方は、クラスに翻訳される。インタフェースについては、クラスは実際にプロキシとなる。メソッドがそれらに起動される場合、それらは、サーバにリクエストを送信するため ORB を利用する。しかしながら、値型から翻訳されたクラスはプロキシではなく、それらは単にローカルのオブジェクトとなる。

クライアントが属性を得るために Equipment Proxy 上の呼び出しを起動する時、サーバからの応答は EquipmentValueType になる。ORB が受ける際には、サーバからの属性値を伴った EquipmentValueType オブジェクトのローカルのインスタンスを作成する。ベース管理オブジェクトインタフェース上で定義された attributesGet() メソッドへのリターン・タイプが、ManagedObjectValueType であるので、EquipmentValueType インスタンスへの参照はタイプ ManagedObjectValueType の参照として次に引き渡される。EquipmentValueType が ManagedObjectValueType に派生するので、これは動作する。しかしながら、EquipmentValueType に特有の属性にアクセスするために、クライアントは、EquipmentValueType をタイプするためにそれをキャストすることにより、参照を限定する必要がある。

ORB によって行われている舞台裏の処理が少し複雑になっているため、選択子としては属性値を維持する為に CORBA の any 型のリストを使用する。しかしながら、このアプローチはより多くの処理を必要とする。Any 型は、プログラマにとっても複雑になる。上記の例において示されるように、値型の使用は現実に全く単純である。

6.11 定数

ネットワーク管理システムは、予め決められた意味で情報を交換する能力を要求する。例えば、probable cause として“1”の状態変化通知は、信号損失による可能性を示しているが、“2”はフレーム損失を意味

する等。あるフィールド中でインタフェースをまたがる列挙型あるいは整数値の集合を定義することは容易であるが、パラレルに動作するような複数のグループに拡張する機構とするのは若干の手間が必要となる。このためにこれらのガイドラインによって用いられる機構は“Universal Indentifier(UID)”と呼ばれる。UID は 2 つのフィールドを備えたデータ構造である。1 番目はあるフィールドのために定義された定数を含んでいる IDL モジュールの scoped された名前を含むと意味されたストリングである。第 2 は値を含んでいる、short サイズ(16 ビット)の符号付整数である。例えば、probable cause フィールドに信号損失の値を送るために、システムは、“itut_x780::ProbableCauseConst”に等しい moduleName 文字列と、29 という整数値からなる UID 構造を構築する。(付属資料 B は、これらのガイドラインのために定義された定数を含んでいる。それでは、29 の値を備えた lossOfSignal と命名される定数を含んでいる“ProbableCauseConst”と命名されるモジュールがある。)

これがこのフレームワーク内に使用される一定の値のためのただ一つのフォーマットであることに注意する必要がある。“ローカルの”値は使用されていない。

情報モデル用定数を定義する場合以下の決まりに従うものとする:

1) 定数の値は特定のフィールド用に定義された定数の集合毎に、個別のモジュールに定義される。これらのサブモジュールは、情報モデルを定義した別な構造体を含んだトップレベルのモジュールに含まれる。

2) モジュールの名前は“Const”で追加されたフィールドの名前であるものとする。例えば、probableCause フィールド(タイプ UIDType として定義された)に対する値は“ProbableCauseConst”と命名されるモジュール内に含まれている。

3) サブモジュール内に定義された定数は、const shortタイプにする必要がある。例えば:

```
const short lossOfSignal = 29;
```

4) 定数は、メインのIDLファイルの長さや複雑さを減らすために分割されたファイルの中で維持されるかもしれない。定数が個別のファイルにあっても、サブモジュールは、メインファイル中のモジュールと同じ名前を備えたIDLモジュール・ステートメント内にあるものとする。メインファイルは、コンパイル実行中に定数をincludeするためにファイル先頭において、プレ・コンパイラであるinclude文を記述する。

5) サブモジュールは、さらにそのモジュールのscopedされた名前を含んでいる“moduleName”と命名されるストリングの定数を含む。例えば:

```
module itut_x780 {
        …
        module ProbableCauseConst {
                const string moduleName = ″itut_x780::ProbableCauseConst″;
                …
        }; // end of module ProbableCauseConst
        …
}; // end of module itut_x780
```

これは実際にプログラマが、ハード・コーディングするモジュール・ストリング名というよりも、定数によりモジュール名称を参照する為の単なる形式となる。

他の情報モデルが probable cause に対する値を拡張するかもしれないことに注意する必要がある。例えば、probable cause フィールドとしてモジュール“itut_m3120::ProbableCauseConst”は追加可能な値がある。これらのモジュールは値 29 をさらに再使用することができる。モジュール名が異なるので、UID はまだユニークになる。

6.12 登録

CORBA IDL は、モジュール内の識別子がすべてユニークでなければならない。これは、モジュール名がユニークな限りその内容がすべてユニークに指定されるだろうということを意味する。CORBA IDL は、CORBA ORBs に用いられるインタフェース情報の中心ディレクトリである CORBA インタフェース・リポジトリ中で登

録される時、モジュール識別子のためのユニークな接頭辞(prefix)を定義する為に用いるかもしれない IDL コンパイラの pragma statement もまた定義する。このフレームワークは、IDL ドキュメントが、含まれるモジュールの為の接頭辞として組織の Internet ドメイン名を用いた pragma prefix statement を含むことを必要とする。
これは、個々の individual construct を登録する必要性を排除する。

6.13 CORBA/IDL 仕様の版数

CORBA を使用する場合、管理インタフェースは、IDL を用いて定義された 1 つ以上のオブジェクト・インタフェースとして指定される。必然的に、管理インタフェースは変更される。管理インタフェースへ新しい CORBA オブジェクト・インタフェースを加えることは容易である。新しい CORBA インタフェースの場合には単に IDL で定義を行ない、個別の管理インタフェースをサポートするオブジェクト・インタフェースを識別する specification に追加する必要がある。

しかしながら、既存の CORBA オブジェクト・インタフェースの更新はもう少し手続きを必要とする。これらのガイドラインは以前のものに対する互換性に優先事項を置く。したがって、次の規則は既存の管理オブジェクトインタフェースの拡張に適用される。オブジェクトのビジネス目的の変更に関係しない基底クラスへの拡張にのみ、これらの規則は適用されることに注意が必要である。すなわち、新しいクラスは古いクラスと同じ資源をモデル化し、それはいくつかの能力の追加を行なっているだけとなる。

1) 新しいオブジェクト・インタフェースの名前は、既存のインタフェースに文字“R”と1から始まる数字を追加する。インタフェースを更新する場合には、数字をインクリメントする。したがって、管理オブジェクト“Equipment”のためのインタフェースの拡張は“EquipmentR1”と命名されるインタフェースとなる。

2) 新しいインタフェースは、既存のインタフェースと同じモジュール名の内に定義されるものとする。(CORBAモジュールは実際には単に名前スペースで、複数のファイルに渡っているかもしれない。)

3) 新しいインタフェースは既存のインタフェースを継承するものとする。

4) 既存のインタフェースから継承された能力は新しいインタフェースの中で削除したり、修正することができない。操作定義を修正しなければならない場合、新しい操作として定義しなければならない。新しい操作の名前は、既存の操作名に文字“R”と1から始まる数字を追加して定義する。ゆくゆくの変更の場合には、数字をインクリメントする。

5) Name Bindingの中で用いられるkindフィールドの値は、オブジェクト作成時に参照するname bindingモジュールの中の定数によって決定される。既存のインタフェースに有効などんな名前bindingsも新しいインタフェースに対して有効である。すなわち、subordinateSubclassesAllowedのモジュールの値が偽でも、Equipmentオブジェクトのname binding モジュールはEquipmentR1オブジェクトに対しても有効であるものとする。

6) 新しいインタフェースへの参照は最もSpecificタイプであるべきである。(それらがそうでない場合、新しい能力はアクセスすることができない。)さらに、新しいクラスのオブジェクトによって報告されたobjectClass属性の値は、最もspecificタイプであるべきである。CORBAは、IORに含まれていた情報に基づいた参照の実際のクラスを決定するためのある手段を提供する。

例えば、次のオブジェクト・インタフェースでは:

```
interface Foo {
        void action(in int A, in int B);
}
```

そのアクションはこのように拡張されるかもしれない:

```
interface FooR1: Foo {
        void actionR1(in int A, in int B, in int C);
}
```

古いアクションはまだ有効な操作になる。

一定の定義を含めて、他の既存の IDL 定義が改訂される場合、“R”およびインクリメントされた数を備えた名前を追加する同様のアプローチは使用される、定義および valuetype 定義をタイプする。

７．GDMO の翻訳

本節は、GDMO を使用して記述した既存の情報モデルから IDL 情報モデルを作成するためのガイドラインを規定する。以降の各節では、GDMO テンプレートの各々がどのように CORBA　IDL に翻訳されるべきかを記述している。

7.1　管理オブジェクトのクラス

GDMO 仕様の各管理オブジェクトクラスは、管理オブジェクトインタフェースに翻訳されなければならない。GDMO のトップクラスから生じた管理オブジェクトクラスの翻訳は、*ManagedObject* CORBA IDL を継承しなければならない。トップから直接的には生じないクラスの翻訳は、それがどのクラスの翻訳であろうと継承しなければならない。全ての管理オブジェクトインタフェースは、*ManagedObject* インタフェースを直接的または間接的に継承しなければならない。多重継承は CORBA　IDL の規則に従うことができる。しかし、留意すべきは、これらの規則が CMIP と異なることである。特に、CORBA では、属性やオペレーションが交替で同一の共通ソースから継承しない限り、多重ソースから継承することができない。仮に、CMIP からの多重継承解釈が、CORBA の規則に一致しないとすると、トランスレータはある上位クラスから継承を選択し、他のクラスからの他の能力を手動で付加しなければならない。他のオプションとしては、共通ソースからの相反する能力を継承するよう相反する上位クラスを変更しなければならない。これは、勿論、以上の上位クラスの再定義を必要とする。

潜在的な上位クラスからの継承不能性もまた、潜在的な上位クラスあるいは任意の上位クラスが変更される場合、手仕事が必要となる可能性があることを意味している。一層深刻な問題は、CORBA 多様性が継承に基づいていることである。サブクラスがクラスから継承されない場合、それはクラスに対して多様性ではない。不幸なことに、これが CORBA の限界であり、このガイドラインでは扱わない。

必須および条件パッケージでの属性、動作、そして通知は、以下のガイドラインに従ってインタフェースのオペレーションに翻訳される。インタフェースの前のコメントは、条件パッケージの能力がこのパッケージに関する“PRESENT IF”に基づくインスタンスによってサポートされなければならない諸条件を記述しなければならない。留意すべきは、CORBA では、上位クラスに存在する能力の再定義を許容していないことである。従って、能力が上位クラスで条件として定義される場合、これはサブクラスでは必須として再定義できない。（以下に記述する通り、能力が *NO<package_name>*例外を生じる時、能力は条件を表す）。サブクラスでは例外を除去することができない。最良の代案は、サブクラスが例外を生じないことを示すコメントである。別の代案は、継承を放棄し、これを行いながら必須にするように手動で能力を付加することであった。しかし、これは多様性や手動更新の問題に繋がる可能性があった。）

個別のインタフェースの登録は不要である。

7.2 パッケージ

不幸にして、IDL は１つのパッケージをインタフェースに翻訳して、パッケージを定義する手段以外を提供することはない。しかし、これは多くのエキストラインタフェースを生じ、CORBA インタフェースの複雑さを増すことになる。その代わり、このガイドラインには、能力群向け条件サポートのコンセプトが盛り込まれている。

上述のように、GDMO パッケージが管理オブジェクト・クラスに含まれている時はいつでも、このクラスの IDL インタフェースへの解釈は、パッケージのテンプレート各々の解釈を含んでいる。条件パッケージの１部である GDMO の属性は、各々がこのパッケージ向けに定義した例外を含むレイズ節を生じるアクセス・オペレーションに翻訳されなければならない。条件パッケージの１部である GDMO の動作はまた、このパッケージ向けに定義された例外を含むレイズ節を備えたオペレーションに翻訳されなければならない。条件パッケージの１部である GDMO の通知は、*CONDITIONAL_NOTIFICATION* マクロ・ステートメントに翻訳されなければならない。

GDMO オブジェクトの条件パッケージステートメント内の *present if* 節はオブジェクトの IDL 翻訳に先行するコメントに翻訳されなければならない。

異なる条件パッケージに同一能力がある時、CMIP からの翻訳にも問題がある。GDMO オブジェクト内の条項ならば条件つきのパッケージステートメントはオブジェクトの IDL 翻訳の前に、コメントに解釈される。以上の規則は以下のように規定される：

1) 任意のソースもしくは、他のソースでは条件で必須の場合、これは翻訳されたクラスでも必須でなければならない。
2) 能力が多重条件パッケージの１部である場合、翻訳されたオペレーションは各パッケージに対して例外を含む。例外が生じるのはパッケージが全く無い場合のみで、その時、いずれかの例外が生じてよい。
3) 同一の条件パッケージが多重上位クラスから取り込まれる場合、パッケージが新しいクラスに含まれる条件は、上位クラスの論理"OR"条件である。
4) 多重パッケージの１部である通知は、正に単一のマクロ・ステートメントに翻訳される。任意のパッケージが必須の場合、*MANDATORY_NOTIFICATION* マクロ・ステートメントが使用される。そうでなければ、*CONDITIONAL_NOTIFICATION* マクロ・ステートメントが使用され、全てのパッケージの例外がリストアップされる。

単一オブジェクトに含まれる多重条件パッケージに GDMO テンプレートが発生する場合、モデラは能力を必須にするかあるいは能力向けに新しい条件パッケージを定義することを考えてもよい。
パッケージだけを表す例外の使用は条件パッケージをサポートすることに留意すべきである。仮に、GDMO クラスに多重命令パッケージがあれば、これらのパッケージは翻訳されたインタフェースでは区別することができなくなる。

パッケージの定義に伴う振舞いステートメントは、パッケージを含む GDMO オブジェクトから翻訳された IDL オブジェクトのインタフェース定義でコメントに翻訳されければならない。
パッケージの登録は不要である。

7.3 属性

上記の通り、GDMO の管理オブジェクトのクラスは、クラスの定義に含めなければならないパッケージをリストアップしている。そして、パッケージは当該パッケージを構成する属性、動作および通知をリストアップしている。管理オブジェクト・クラスを翻訳する時、含まれたパッケージの中の各テンプレートは、管理対象オブジェクトインタフェース上で翻訳され、これらの大部分は属性の定義を含む。

GET 能力をサポートする属性には、これらに関して定義された＜Attribute Name＞Get オペレーションを付けなければならない。オペレーションのリターン型は、属性の ASN1 シンタックスの翻訳でなければならない。

REPLACE 能力をサポートする属性には、これらに関して定義された＜Attribute Name＞Set オペレーションを付けなければならない。オペレーションの入力パラメータの型は、属性の ASN1 シンタックスのの翻訳でなければならない。

ADD 能力をサポートする属性には、これらに関して定義された＜Attribute Name＞Add オペレーションを付けなければならない。

REMOVE 能力をサポートする属性には、これらに関して定義された＜Attribute Name＞Remove オペレーションを付けなければならない。これらのオペレーションの入力パラメータの型は、属性の ASN1 シンタックスから翻訳された IDL シーケンスでなければならない。

set-by-create 能力をサポートする属性は、ファクトリ作成方法での初期値を受け入れなければならないが、*SET* オペレーションは付けてはならない。（ファクトリ作成方法も単に読み出せるのではなく設定可能な属性についての値を受け入れることになる。）

デフォルト値は、インタフェースの定数として定義される。定数の識別子は＜Attribute Name＞Default でなければならない。インタフェースにも属性をデフォルトに設定するためのオペレーションが付く可能性がある、あるいはクライアントはデフォルト定数で *SET* オペレーションを使用できる。

Set-to-default オペレーションは、＜Attribute Name＞SetDefault と名称付けされなければならず、これはいかなるパラメータをも受け入れてはならず、*void* を返さなければならない。CORBA IDL は simple 型およびエミュレート型のみについて定義された定数を受け入れるが、属性の型が複雑な場合、これについては全くデフォルトを定義しないことができる。これらのケースでは、set-to-default オペレーションが定義されなければならず、set-to-default オペレーションと関連するコメントがデフォルト値を記述しなければならない。若干の他の属性に関する GDMO 能力を IDL で再作成することはできない。*DERIVED-FROM* 節を持つ GDMO 属性はインタフェース仕様に手動で付加する他の属性の能力を備えることになる。（derived-from 属性のシンタックスが使用されることになる。）マッチング規則は多重オブジェクトオペレーション・サービス制約された言語で定義されるが、この言語は ITU-T Q.816 で定義された TMN CORBA サービスの１部である。これらのマッチング規則は、単に属性の基本型に左右される。属性ごとのマッチング規則はない。初期値、許容値および要求値はサポートされない。これはしばしば各属性に関して IDL 型を定義するのに意味のあることとなる。たとえ属性が単一型でも、これの型の定義に IDL *typedef* ステートメントが使用される可能性がある。属性に関する型定義に先行するコメントは、振舞いステートメントの翻訳を差し挟むのに最良の場所である。そうでなければ、振舞いステートメントは、オブジェクト・インタフェースの属性アクセスオペレーションに先行するコメントに翻訳される可能性がある。

これらのガイドラインで定義される標準的な属性は、使用できるところならどこでも使用されなければならない。6.3.5 節参照のこと。

属性の登録は不要である。

7.4 属性グループ

これらのガイドラインは属性グループのコンセプトをサポートしていない。GDMO の属性グループには同等の翻訳が全くない。

7.5 動作

動作は IDL オペレーションに翻訳されなければならない。入力パラメータ、出力パラメータおよびオペレーション向けリターン型は、動作の入力と出力 ASN1 シンタックスから翻訳されなければならない。すなわち、入力シンタックスは、IDL *in* パラメータに翻訳されなければならず、一方、出力シンタックスは、*out* パラメータとリターン値の混合に翻訳されなければならない。IDL *inout*(in/out)パラメータは、適切なところで使用される可能性がある。例外もノーマルおよびエラー値のリターン結合よりもエラー条件に関するリターン値のために定義されなければならない。

unconformed モードがある GDMO 動作（*MODE UNCONFIRMED* 節が無いもの）は、リターン型に先行する IDL キーワード *oneway* があるメソッドに翻訳されてもよい。しかし、これらのオペレーションには、*void* のリターン型がなければならず、*out* ないしは *inout* パラメータがあってはならない。*oneway* キーワードの無い IDL オペレーションが受付られる。

7.6 通知

これらのガイドラインで定義するのは、大部分の GDMO 情報モデルで使用される通知である ITU-T X 721 に見られる 15 の通知の IDL と同等物である。GDMO パッケージの通知は典型的に、パッケージを含む各インタフェースで通知マクロ・ステートメントに単純に翻訳されることになる。通知が必須パッケージの一部である場合、*MANDATORY_NOTIFICATION* ステートメントが使用され、それが条件パッケージの一部である場合、*CONDITIONAL_NOTIFICATION* ステートメントが使用される。
通知ステートメント内での通知フィールドに対するオブジェクト属性のマッピングはサポートされない。ある特別なマッピングが必要な場合、これはコメント付きの文書でなければならない。通知への回答はサポートされない。

新しい通知を定義しなければならない場合、これは新しい情報モデルのモジュール内で“notification”と名付けられたインタフェースでのオペレーションとして定義されなければならない。（このことは、このインタフェースが itu_x780:: Notofication インタフェースから継続しなければならないことを前提条件とするものではない。）オペレーション名は、通知の名称でなければならない。オペレーションへのパラメータは、通知の情報シンタックスから翻訳されなければならない。通知オペレーションのリターン型は void でなければならず、*in* パラメータでなければならない。ITU-T Q.816 はどのようにしてデータを構造化された通知に入れ込むかについての情報を提供する。属性 IDs が不要であることに留意すべきである。代りに、パラメータは名称やデータ型によって識別される。この時、標的のインタフェース名や通知オペレーションは通知マクロ・ステートメント内で使用され得る。この通知（”R1”,等を添え）には新しい名称が与えられ、管理システムが旧版や拡張版の両方を含むアラームを受信するための multiply-inherited インタフェースを作成することができる。

通知が拡張される必要がある場合、新しいオペレーションを定義することで拡張されなければならない。新しいオペレーションは、旧いものと同じパラメータを持たなければならない。例えば、以下の IDL は”newType” 型の”newData”と名づけられたパラメータを付加して装置のアラームを拡張している。

```
module newModule {
…
        interface Notifications {
                void equipmentAlarmR1 (
                        in ExternalTimeType             eventTime,
                        … (other equipmentAlarm parameters)
                        in SuspectObjectSetType         suspectObjectList,
                        in newType                      newData);
        }
}
```

7.7 振舞い

GDMO 振舞いテンプレートは、各振舞いが関連する IDL 構成の直前のフォーマットされた IDL コメントに翻訳されなければならない。属性の振舞いは、属性型のためのタイプ定義に先行する IDL コメントに翻訳されなけらばならない。パッケージの振舞いは、コメント向けに定義された例外に先行する IDL コメントに翻訳されなければならない。

7.8 ネームバインディング

各 GDMO のネームバインディングは、6.8 節で定義した IDL ネームバインディングモジュールに翻訳されなければならない。IDL ネームバインディングでの様々な構成は以下のように翻訳されなければならない：

ネームバインディングの上位クラス名称は、ネームバインディングモジュール内の *superiorClass* 定数の値に割り当てられなければならない。GDMO 上位クラス節に、*AND SUBCLASSES* 修飾子がある場合、IDL ネームバインディング定数 *superiorSubclasses Allowed* は *true* でなければならない。さもなければ、*false* でなければならない。

ネームバインディングの従属クラスは、ネームバインディングモジュールの *subordinateClass* 定数の値に割り当てられなければならない。GDMO 従属クラス節に AND SUBCLASSES 修飾子がある場合、IDL ネームバインディング定数 *subordinateSubclasesAllowed* の値は、*true* でなければならない。さもなければ、*false* でなければならない。

GDMO ネームバインディングに *CREATE* 節がある場合、IDL ネームバインディング定数 *managersMayCreate* が *true* でなければならず、無い場合は *false* でなければならない。

GDMO ネームバインディングに DELETE 節が全く無い場合、IDL ネームバインディング定数 deletePolicy の値は notDeletable でなければならない。DELETE 節に全く修飾子が無いかあるいは ONLY-IF-NO-CONTAINED-OBJECTS 修飾子がある場合、deletePolicy の値は deleteOnly1NoContainedObjects でなければならない。DELETE 節に CONTAINED-OBJECTS 修飾子がある場合、deletePolicy の値は deleteContainedObjects でなければならない。

ネームバインディング create 節に *WITH-AUTOMATIC-INSTANCE-NAMING* 修飾子がある場合、管理オブジェクトのファクトリ create オペレーションは、*inout* として名称パラメータを定義しなければならず、クライアントが無効の名称を出しても良いことを示すコメントを盛り込まなければならない。もしそうならば、ファクトリは名称を選んで、それを返す。

参照オブジェクトから属性値の部分をコピーしてオブジェクトを作成することは、強く型付けされたファクトリのメソッドでは不可能である、その理由は、ファクトリにはどの値をコピーし、オペレーション・パラメータのどれを使用すべきか全く言えないからである。全ての値をレファレンスからコピーする強く型付けされたオペレーションを定義することができるが、これの有用性は限られている。属性の部分的なリスト同様レファレンス・オブジェクトを受け入れた弱く型付けされたオペレーションもファクトリで定義できるが、これを実行する困難さから利益にならないと思われる。従って、*WITH-REFERENCE-OBJECT* 修飾子のネームバインディング create 節への翻訳はサポートされない。

create 節のパラメータは *CreateError* 例外に翻訳されなければならない。これはエラーID に対して新しい

値を定義することを要求してもよい。ネームバインディングで作成したオブジェクトにどの CreateError 例外エラーIDs を適用するかを通知するネームバインディング IDL モジュールにコメントが設定されなければならない。Create 節パラメータを *CreateError* 例外に翻訳できない場合、別の手段として、あまり望ましくないが、代案として新しいファクトリを定義し、パラメータを当該ファクトリで create オペレーションの例外に翻訳することがある。*CreateError* 例外の汎用性を考えれば、この必要は稀でなければならない。（詳細については以下のパラメータを参照のこと。）

7.9 パラメータ

GDMO パラメータは GDMO 情報モデルに拡張性を提供する。サブクラスを定義する時、通知、動作（リクエスト、レスポンスおよび障害）および特定のエラーのエリアでの既存の仕様を増加させるために、パラメータ・テンプレートが使用される。全ての通知と多くの動作の GDMO 定義は、（必要な場合）サブクラスで定義される拡張フィールドを備えている。specific error の場合には、class-specific エラーが、CMIP で一般“processing failure”エラーを増加させるために使用される。この情報のフォーマットはしばしば名称-値ペアのリストであり、ここで名称は値のデータ型を定義する。

GDMO パラメータを IDL に翻訳することは、多くのオブジェクト・クラスとともに役に立つとわかっている、現在定義されている拡張を、“normal”で強く型付けされた部分のモデルにする良い機会を与える。例えば、アラーム向けに定義された３つの GDMO パラメータは、IDL で定義された通知に含まれた。（３つのパラメータとは、”Alarm Effect On Service”, “Suspect Object List”および”Alarming Resumed”である。）

拡張性のセマンティックを明示するため、GDMO パラメータ・テンプレートでは幾つかのキーワードが使用されている。これらのキーワードに基づくパラメータ・テンプレートで利用できる様々な拡張能力の解釈は以下で議論される。

7.9.1 ACTION-INFO と ACTION-REPLY

フレームワークで推奨された強い型付けを保ったままでは、テンプレートにキーワード”ACTION-INFO”がある GDMO パラメータは、拡張フィールドとしては翻訳されない。その代わり、動作を規定し、メソッドの通常の”in”パラメータとして拡張を付加する既存のインタフェースから新しいインタフェースがサブクラスにされる。IDL パラメータの名称はパラメータから採られなければならず、パラメータのデータ型は GDMO パラメータシンタックスから翻訳されなければならない。“ACTION-REPLY”パラメータもオペレーションで”out”パラメータに翻訳されなければならない。

上記のメソッドは、次にパラメータが既に存在する IDL オペレーションに付加されることがサポートされないことを前提条件としている。その代わり、情報モデラは、オブジェクト・インタフェースをサブクラスにしたり、新しいメソッドを定義したりするようなインタフェースを拡張するため、CORBA によって提供されるもっと普通のアプローチを使用することができる。この場合、追加の *in* かつ／または *out* パラメータあるいは追加の例外を持たせる

これについてのガイドラインについては 6.13 節参照）

7.9.2 EVENT-INFO と EVENT-REPLY

“EVENT-INFO”パラメータが既に定義されている場合、これらのパラメータは通知を送るのに使用される IDL パラメータの規則的な”in”パラメータに翻訳される。これらのガイドラインは通知へのレスポンスをサポートしていないので、” EVENT-REPLY”パラメータ向けの解釈は全く無い。

このフレームワークは既に一連の通知を定義しているので、EVENT-INFO パラメータの解釈は、通知オペレーションの１つの再定義を意味する。7.6 節参照

しかし、大部分のケースでは、既存の通知定義の再使用が好まれることになる。GDMO 拡張がアラーム情

報について事前に定義されている場合、この拡張は翻訳された通知 IDL 仕様に含まれなければならない。しかし、フレームワーク通知 IDL も弱く型付けされた名称－値ペアリストである“additional information”フィールドをサポートする。これは情報をこれらの事前に定義された通知に付加するのに使用することができる。通知 event 型は不変であることになる。特定のパラメータ向けの拡張を使用する必要がある管理オブジェクトのインタフェースは、コメント内のこのパラメータの使用に注意すべきである。不幸にして、どの通知がどのオブジェクトでサポートされるのかを明示するには、上記のマクロの使用を除いては他のメカニズムは全く無いが、このことはパラメータの明示をもサポートしないことではない。同じ通知型を使用することの利点は、マネージャが通知を受け取り、新しい通知型を記録することに係わりを持たなくできることである。異なるマネージャやエージェントのせいで拡張が理解されない場合、追加情報が切り捨てられる。

追加情報向けの拡張仕様は以下に記述されている。フレームワークで定義された通知には、CMIP 通知で”additionalInformation”フィールドとそっくりの”additionalInformation”と呼ばれるフィールドが含まれている。通知での”additionalInformation”フィールドの IDL シンタックスは、”AdditionalInformationSetType” 型である。

```
struct ManagementExtensionType {
        UIDType id;             // identifies the type of info
        any     info;           // type will depend on id
};
typedef sequence <ManagementExtensionType> AdditionalInformationSetType;
```

EVENT-INFO キーワードを持つパラメータは、各パラメータ向けのユニーク識別子(UID)を定義して翻訳されるこれの詳細については 6.11 節参照。要するに、モデラは 値型“short”の定数が定義されている”AdditionalInformationConst”と呼ばれるサブモジュールを定義する。これらの定数の名称は GDMO パラメータの名称である。各定数値もおそらくパラメータの登録の最後の数に基づく GDMO から導かれる。さもなければ、当該モジュールの定数に対して唯一の整数が選択されなければならない。この定義にも”additionalInformation”フィールドで UID に伴う値のデータ型を示すコメントを含めなければならない。一例として、Alarm Effect On Service パラメータがこのフレームワークでのアラームで使用された Alarm Info データ・ストラクチャの正常数にならなかった場合、これは次のように翻訳される可能性がある：

```
module itut_m3100 {
…
 module AdditionalInformationConst {
 /** Alarm effect on service parameters are accompanied by a boolean
 value in the "any" field indicating if service has been affected. */
 const short alarmEffectOnService = 1;
 …
 }; // end of module AdditionalInformationConst
}; // end of module itut_m3100
```

管理オブジェクトの IDL インタフェースはこの時、通常これがサポートする通知を識別できるが、コメントは通知に含まれることになるパラメータを示さなければならない。

7.9.3 Context-Keyword

context-keyword パラメータは、CMIP PDU 中の名称付きのフィールドで渡されることになっている情報を識別する。この名称がつけられたフィールドは通常、データ構造のシーケンスであり、そのデータ構造は、識別子、および識別子に依存する値を保持する”any”データ型から構成される。CMIP では、これらの context-keyword パラメータは Action パラメータまたは通知の中で渡されてもよい。Action への context-keyword パラメータの翻訳は、強い型付けを優先するためこのフレームワークでサポートされていな

い。その代わり、Action への追加情報は、規定のオペレーション・パラメータに翻訳されなければならない。（上記 ACTION-INFO パラメータを参照のこと。）

（上で説明済みの）拡張を除き、通知については、フィールドが弱い型と定義される場合、拡張フィールドと同じアプローチが使用できる。しかし、大部分の GDMO 規格ではこのアプローチは使用されていない。EVENT-INFO キーワードと context-keyword との相違点は、EVENT-INFO キーワードは１つ以上のパラメータを付加できる拡張性をもっている点にある。拡張の推薦アプローチは EVENT-INFO の使用であり、全ての標準はこのキーワードを使用するパラメータを定めている。

7.9.4 SPECIFIC-ERROR

“SPECIFIC-ERROR”パラメータは、CMIP プロセッスイング障害メッセージに戻され、オペレーションの異常な結果が示される。これらのパラメータの解釈には2つのオプションがある。先ず、パラメータは specific-error パラメータ定義されるオペレーションから生じる IDL 例外に翻訳される。例外の名称は、GDMO パラメータ名称から採用すべきであり、例外と共に戻ってきたデータ型は、GDMO パラメータ・シンタックスから得なければならない。Specific- error パラメータが GDMO テンプレートの異なる種類に定義できるので、動作の specific- error パラメータは動作から生じる例外に翻訳されなければならない。属性の specific- error パラメータは属性アクセスオペレーションから生じる例外に翻訳されなければならない。また、ネームバインディングの”Create”節での specific- error パラメータは、ファクトリ・インタフェースで create オペレーションの例外に翻訳されなければならない。通知へのレスポンスが許容されないので、このフレームワークでサポートされる通知に関する specific- error パラメータは全く無い。

specific- error パラメータを specific-error パラメータを翻訳する２つ目のオプションは、パラメータをフレームワークで定義する標準的例外の１つへの新しいコード･ポイントに翻訳することである。フレームワークは３つの標準的例外を定義する：ファクトリ create オペレーションで生じる *CreateError* 例外、管理オブジェクト delete オペレーションで生じる *DeleteError* 例外および他の全ての管理オブジェクト･オペレーションで生じる *ApplicationError* 例外。*ApplicationError* 例外は、特定のアプリケーション・エラーとテキスト説明を識別する唯一の識別子を返す。Create と delete エラー例外は、含まれている関係オブジェクトのリストおよび、オブジェクトが試みられたオブジェクトの属性を付加することで、情報を拡張する。関係するオブジェクトのリストは、例えば、目標のオブジェクトを消去可能になる前に消去すべきいくつかのオブジェクトを示してもよい。属性にはエラーに関係するオブジェクトの状態情報があってもよい。

Specific-error をこれらの標準的な例外の１つが使用するコード・ポイントに翻訳することを、可能であればいつでも行わなければならない。例外で戻ってきたデータ型は値型なので、これらは特定のコード・ポイントに拡張することができる。理由は、delete オペレーションが、*DeleteError* 例外コード・ポイントに翻訳されなければならない GDMO ネームバインディング delete 節に現れる基本の管理オブジェクト・インタフェース specific-error パラメータから、継承されるからである。これは上記の EVENT-INFO についても同様に行われる。基本的に、モデラは UID 定数向けに delete エラーサブモジュールを定義する。定数の定義にはどのデータが配置されるのかの”relatedObjects”やこの識別子を持つエラーを伴う”attributeList”フィールドにあるかを示すコメントを含まなければならない。また、モデラがコード・ポイントのために戻った標準値型を拡張した場合、マネージング・システムが型を狭め、追加情報にアクセスできるように、コメントは戻った現在のデータ型を記録しなければならない。事実、フレームワークは標準 delete エラー値型を拡張する幾つかの delete エラー・コード・ポイントを含んでいる。

最後に、管理オブジェクトの IDL インタフェースについての消去エラー値を示すコメントは、オブジェクトの消去が不正確に行なわれる時に例外で上げてもよい。

```
module itut_m3100 {
…
```

```
  module DeleteErrorConst {
  /** Network TTP Terminates Trail delete errors are raised when an
  attempt is made to delete a TTP before the trail has been deleted.
  It includes a reference to the Trail in the "relatedObjects" field. */
  const short networkTTPTerminatesTrail = 54;
  …
  }; // end of module DeleteErrorConst
};  // end of module itut_m3100
```

7.10　ASN.1 データ型

GDMO は、ASN.1 言語を使用して、属性のシンタックスおよびオペレーションと通知パラメータを定義するので、GDMO テンプレートを IDL に翻訳するときには、これらのシンタックスの定義も翻訳しなければならない。この節は、ASN.1 シンタックスを CORBA　IDL に翻訳する際のガイドラインを定める。

7.10.1　基本データ

CORBA　IDL は、ASN.1 基本データを翻訳することができる、次の基本データを定義する。任意の、boolean, char, （二倍精度浮動小数点数を意味する）double、（計数値を意味する）enum、固定された、（単精度浮動小数点数を意味する）浮動、（大きな整数を意味する）長い、（オブジェクト・レフェランスを意味する）オブジェクト、オクテット、（小さな整数を意味する）短い、文字列、（”wide”文字を意味する）wchar および（”wide”文字の文字列を意味する）wstring。

このフレームワークは、すべての文字列についてstring型を使用し[1]、文字列が拡張された国際的文字を含むことができる場合には"Istring"と呼ばれる*typedef*を定義する。Istringとは、wstring、つまり、”wide”文字列のtypedefである。これらの文字列は、“幅の広い”（16 ビット）文字から構成されている。

さらに、CORBA タイム・サービスは、このフレームワークが使用する"UtcT"と呼ばれるタイム（時間）を定義する。

[1] ITUT-T注 : GIOPバージョン 1.1 およびこれ以上でサポートされるcordsetネゴシエーションが使用される時、ストリングが国際文字を扱えるので、wstringの代わりにstrinngに対してIstringの代わりのtypedef使用の貢献が要請される。Wstring型は、しばしばUnicodeに結び付けられるプログラミング言語wstringに対してCORBA言語構築でマッピングされる。

7.10.2 **シーケンス**

CORBA　IDL は、ASN.1 *sequence* 型に類似した *struct* キーワードを使用するデータ構造の定義をサポートする。

7.10.3 Sequence of

CORBA　IDL は、ASN.1 *sequence of* 型とほぼ同様の方法で、基本および複合データ・シーケンスの定義をサポートする。

7.10.4 Set of

CORBA　IDL は、ASN.1 とは異なり、複合集合型の定義をサポートしない。ただし、集合は、IDL シーケンスに翻訳される。タイプの名称の末尾を"SetType"とする規約にしたがうものとする。設定値を操作するときには、重複は消去されなければならず、順序は消去されなければならない。

7.10.5 **選択**

CORBA　IDL は、ASN.1 の選択型と同じ目的に使用される、識別されたユニオンの定義をサポートする。

CORBA ベースの TMN 標準命令の実行を簡略化するため、このフレームワークは、識別されたユニオンの利用を控え目にすることを推奨する。ASN.1 から CORBA　IDL に翻訳するときには、大抵の場合、翻訳された型は、意味を損なうことなく簡略化することができる。例えば、文字列と空白の間の選択は、簡単に文字列に翻訳することができる。この可能性を明確にするため、文字列が空白でもある可能性があるというコメントを追加することができる。sequence of（または set of）と空白の間の選択も、同様に、単純にシーケンスに翻訳することができる。

7.10.6 **オブジェクト識別子（OID）**

このフレームワークは、ASN.1 OIDs の代用として指定された"Universal Identifier"（UID）と呼ばれる型を定義する。

7.10.7 **オブジェクトインスタンス**

フレームワークは、ASN.1 オブジェクトインスタンス型の 2 種類の翻訳の可能性をサポートする。管理オブジェクトはそれぞれ名称をもっているので、CORBANaming Service によって定義された名称型を使用することができる。（このフレームワークは、*NameType* と呼ばれる、CORBANaming Service の名称の *typedef* を定義する）。また、CORBA オブジェクト・レフェランスを使用する事が可能である。管理オブジェクト・インタフェースはすべて、*ManagedObject* インタフェースを受け継がなければならないので、オブジェクトに対する一般的名称が要求される場合には、ManagedObject 型を参照するのが望ましい。モデラは、例えば *Equipment* など、管理オブジェクトの等級に固有な型を使用する事が可能である。そうすれば、より典型的なモデルを製作することができる。

7.10.8 **ビット・ストリング**

この節は、GDMO 仕様を CORBA 管理オブジェクトに翻訳するときに使用する、或いは新しい CORBA 管理オブジェクトの仕様の中で使用されるシンタックスを指定するときに使用する ASN.1 BIT STRING の 2 種類のマッピングを定義する。

ASN.1 ビット・ストリングは、異なる方法で使用されるので、2 種類のマッピングが必要である。

```
BitStringType ::=
```

```
BIT STRING |
BIT STRING{NamedBitList}
```

2 種類のマッピングは、下記の通りである。

・ASN.1 BIT STRING は、IDL オクテット・シーケンスにしたがってマッピングする。IDL には、意味タグは付けられていない。

・ASNI.1 BIT STRING は、状態表現、値を操作するローカル補助機能および関連する意味タグ定数を使用して、IDL 値型にしたがってマッピングされる。

7.10.8.1　ASN.1 BIT STRING の simple な表現

ASN.1 BIT STRING の状態表現は、＜オクテット＞シーケンスとして定義される BitString の IDL typedef にしたがってマッピングされる。下記の宣言は、simple なビット・ストリングの表現に使用される"itut_x780.idl"に含まれている。

typedef sequence<octet> BitString:

BitString の解釈は、BIT STRING の BER エンコーディングにしたがって定義される。オクテット・シーケンスは、最初のオクテットの後にオクテットがまったく続かないこともあり、一つまたは複数のオクテットが続くこともある。bitstring のビットは、最初のビットから後続ビットまでを次のオクテットのビット 8 から 1 に位置付け、その後、順番に各オクテットのビット 8 から 1 を続け、最終オクテットではビット 8 から始めて、必要なだけビットを続けなければならない（"first bit"と"trailing bit"の概念は、ITU-T X.208 ｜ ISO/IEC 8824 に明記されている）。最初のオクテットは、最下位のビットとしてビット 1 の、符号のない 2 進整数として、最終オクテットの未使用のビット数を符号化しなければならない。この数値は、ゼロから 7 までの数値とする。bitstring が空の場合には、後続のオクテットはないので、最初のオクテットはゼロとしなければならない。

この simple な表現の場合には、GIOP を使用して、ビット・ストリングの値を IDL の”any”に移行すると、repository ID が送信され、受け手（例えば、orb ユーザー）は、オクテット・シーケンスを、符号化されたビット・ストリングとして解釈できるようになる。

ASN.1 ビット・ストリング定数は、単一引用符が消去された X.208 指定"bstring"フォームの変形フォームを使用して、オクテット・シーケンスとして表示される。"hstring"フォームを使用して定義された ASN.1 定数は、関連する IDL 定数については、この変更された"bstring"フォームに翻訳しなければならない。

表 2/JT-X780　Simple な ASN.1 BIT STRING マッピングの例

（ITU-T X.780）

ASN.1	IDL
CCDScan ::= BIT STRING scan CCDScan ::= '1001101001000011101 10'B	typedef ITUT_X780::BitString CCDScanType; const string scan = "1001101001000011101 10B" ;
G3FacsimilePage ::= BIT STRING -- a seq of bits conforming to Rec. T.4 image G3FacsimilePage ::= '1001101001000011101 10'B trailer BIT STRING ::= '01'H	typedef ITUT_X780::BitString G3FacsimilePageType; //string constants generated for BIT STRING constants const string image = "1001101001000011101 10B"; const string trailer = "00000001B";

7.10.8.2　ASN.1 BIT STRING の Valuetype 表現

値型の状態表現は、ITUT_X780::BitString である。BitStringValType の定数の表現は、最も simple な BitString 型と同じである。ヘルパメソッド（オブジェクト機能が、ローカル・プログラミング言語にどのマップを呼ぶか）は、定義された値型に含まれている。

```
exception InvalidLength { long length } ;
valuetype BitStringValue {
  public BitString bitStringVal;
  factory initValue (in unsigned long number_of_bits);
  factory InitFromBitString (in BitString desiredValue);
  // local operations
  short getBit (in unsigned long position)
        raises (InvalidLength);
  void setBit (in unsigned long position, in short new_bit_value)
        raises (InvalidLength);
  unsigned long length ();
  string asString ();    // produces a string with binary values ("1001011B")
  // input a string with binary values ("1001011B")
  void setFromString (in string string_value)
        raises (InvalidString);
};
```

IDL 定数は、指名ビット位置に関連する意味タグについては、BitStringValue から受け継いだ valuetype で生成される。指名ビットは、ビット・ストリングのオフセット値と等しい値の、符号のない長い型の IDL 定数としてマッピングされる。定数の名称は、導き出された値タイプの定義の範囲内で識別子が衝突する場合には、JIDM 規則によって明確にされた任意の名称となる（例えば、n のときに、同じ状況で識別子"a"を初めて使用する場合には、"a.n"）。

```
const unsigned long <bitname> = <offset>;
```

表 3/JT-X780　Bit String ValueType へのマッピング例

(ITU-T X.780)

ASN.1	IDL
T0 ::= BIT STRING	valuetype T0Type : ITUT_X780::BitStringValue {} //could have used typedef to BitString for simpler mapping

MessageFlag ::= BIT STRING {
posResp (0), negResp (1),
doNotForward (2) }
	valuetype MessageFlagType : ITUT_X780::BitStringValue {
	const unsigned long posResp = 0;
	const unsigned long negResp = 1;
	const unsigned long doNotForward = 2;
	}

a INTEGER ::= 1
T1::= INTEGER { a(2) }
T2 ::= BIT STRING { a(3), b(a) }
	const long a = 1;
	typedef long T1Type;
	const T1Type a_1 = 2;
	valuetype T2Type : ITUT_X780::BitStringValue {
	const unsigned long a = 3;
	const unsigned long b = a;
	}

注記 - ある翻訳の場合で、valuetype 宣言の範囲内であっても、定数の名称を明確にすることが必要になることがある。ネーミングの衝突を回避するには、(T1 を例として取り上げて説明した) JIDM 衝突規則を valuetype の範囲内で使用する事が可能である。

８．CORBA IDL 仕様に関するスタイル慣用句

本節ではインタフェース仕様に使用される CORBA IDL 仕様に関するスタイル慣用句のセットを定義する。スタイル慣用句により、CORBA/IDL 仕様に一貫性のあるスタイルをもたらす。これはエディタによる付加作業を要するかもしれないが、この特別な努力は CORBA/IDL 仕様の信頼性を高める価値がある。スタイル慣用句が必ずしも作者側の利益でなく読者側の利益 であるということが、真相を正しく保つために重要である。

8.1 一貫性のある字下げの使用

本節では、IDL モジュールにおいて使用される字下げスタイルを示す。例として、CORBA セキュリティサービス非拒絶モジュールからの引用を以下示す。

```
enum EvidenceType {
  SecProofofCreation,
  SecProofofReceipt,
  SecProofofApproval,
  SecProofofRetrieval,
  SecProofofOrigin,
  SecProofofDelivery,
  SecNoEvidence // used when request-only token desired
};
interface NRPolicy {
  void get_NR_policy_info (
    out Security::ExtensibleFamily NR_policy_id,
    out unsigned long       policy_version,
    out Security::TimeT      policy_effective_time,
    out Security::TimeT      policy_expiry_time,
    out EvidenceDescriptorListType supported_evidence_types,
    out MechanismDescriptorListType supported_mechanisms
  );
};
```

8.2 識別子に関する一貫性のあるケースの使用

ASN.1 のように幾つかの言語はケースルールを強要するが、他はデファクトルールを持つ。これらのルールは、読者が異なる型の識別子を容易に区別できるので読みやすさが向上する。IDL は、ケースを強要しないため以下ルールを提案している。

- 操作、パラメータ、属性、メンバ及び定数は、最初の組込み単語を除き組込み単語の先頭大文字化をしなければならない。
- その他全ての識別子は、最初の組込み単語の先頭大文字化をしなければならない。

```
module CarModule {
  struct EngineType {
    PistonType piston;
    RodType  pistonRod;
  };
  typedef string KeyType;
  enum WontStartReasonType {
    BatteryIsDead,
    NoGas
  };
  exception WontStart {
    WontStartReasonType reasonEngineWontStart;
  };
  interface FordRanger {
    void startEngine(
      in KeyType key
    )
    raises (
      WontStart;
    );
    attribute EngineType engine;
  };
};
```

8.3 IMPORT に関して JIDM アプローチに従う

他モジュールからインポートした型のモジュール開始において、局所型定義(typedef)を生成する。本明示はインポートするモジュールのエクスポートモジュールからの独立する型をリストする。（注－局所識別子の名前はエクスポートモジュールにおけるものと同一である必要はない） typedef の本使用は、インポートされたインタフェース又は値型（valuetype）定義に使用してはならない。その代わりに、全範囲の名称をこれらに使用すべきである。

```
module ImportingModule {
  // Imports
  typedef ExportingModule::SomeType             SomeType;
  typedef ExportingModule::SomeOtherType        SomeOtherType;
  typedef ExportingModule::SomethingElse SomethingElseType;
  ...
};
```

8.4 OPTIONAL と CHOICE に関する JIDM アプローチの使用

列挙及び数値(整数／実数)型に関して、JIDMでの前述のとおりASN OPTIONAL 及びCHOICEのIDLへのマッピングを使用する。

Open Group and Open-Network Management Forum Joint Inter-domain Management (JIDM) group's Inter-Domain Management: Specification Translation [3]. 一例を以下に示す。

```
// Choice
enum CarChoiceType {
  Ford,
  Cheverolet,
  Chrysler
};
union CarType switch (CarChoiceType) {
  case Ford:    FordType        fordValue;
  case Cheverolet:      ChevroletType           chevroletValue;
  case Chrysler:        ChryslertType           chryslerValue;
}
// Optional
union SunRoofTypeOpt switch(boolean) {case TRUE: SunRoofType the_value};
```

文字列、順序列及びオブジェクト参照に関して、空(null)値は値が存在しない場合のオプションケースを表すために通常使用できる。’空値’と’存在しない’ に意味上の違いがある場合には、上記方法を使用しても良い。

構造体と unions に関して、上記方法が使用できる。又は存在しないオプション値を表すため構造体内の空値を使用してもよい。例えば、２文字列、２空値からなる構造体は存在しないオプション値を表すことができる。値がオプションの場合、コメントをつけてオプションのマークが必須である。

通常、ガイドラインは常識として使用される必要がある。結果の変換は、明白性と有用性に関して評価されなければならない。もし変換が非常に複雑である場合、モデル設計者はそれの簡易化を試みても良い。

8.5 一貫性型接尾辞の使用

接尾辞 Type は全ての IDL 型に付加される。これにより型識別子とメンバーは、IDL がケースに影響されないので、衝突なしに同一名を使用できる。加えて、本慣用句は、他の識別子から型識別子を分離することにより読みやすくする。

8.6 順序型に関する一貫性型接尾辞の使用

順序型(順序有、二重許容)に関して、唯一型から区別するために接尾辞”SeqType”を使用する。

8.7 セット型に関する一貫性接尾辞の使用

セット（順序無、二重不可）に関して、唯一型から区別するために接尾辞”SetType”を使用する。

8.8 オプション型に関する一貫性接尾辞の使用

オプション型に関して、非オプション型から区別するために接尾辞” TypeOpt”を使用する。

8.9 一貫性の様式における操作パラメータの整列

パラメータの一貫した順序付けは、読みやすさを向上させる。パラメータは in、inout、out の順に整列させる。

8.10 非全域識別子空間の想定

名前衝突や再利用促進のため、全ての識別子は部分的な範囲に限定しなければならない（例：モジュール及びインタフェース）。

8.11 モジュールレベル定義

全ての型定義はモジュールレベルにおいて行われなければならない。下位コンテキストにおける入れ子の型定義は再利用の困難性と二重定義になりやすい。

8.12 例外とリターンコードの使用

例外はエラー等の例外条件に使用されなければならない。正常リターンはリターンコードと出力パラメータにより処理されなければならない。

8.13 明示 vs．暗黙　操作

操作は、明示機能を行うべきである。フラッグ等を用いて操作の振る舞いを暗黙的に変更することは混乱の元である。各振る舞いは分離された明示操作に織り込まなければならない。

8.14 例外の大数生成の禁止

例外の大数発生はインタフェース定義理解を困難にする。必要に応じ新規エラーコードの定義を行いカテゴリー毎に例外のグループ化したり、標準例外(*ApplicationError*, *CreateError*, and *DeleteError*) を使用すること。

９．遵守と適合

本節では、本ガイドラインと（本標準適合の）実装機能に対する遵守請求を行う他の標準化文書が合致すべき基準を定義する。

9.1 標準化文書遵守

本ガイドラインに対し遵守請求するいかなる仕様も：

1) 5.1節に記述され、付属資料 AのCORBA IDLで定義されている管理対象（*ManagedObject*）インタフェースから、リソースをモデル化する全てのインタフェースを得なければならない（直接的又は間接的）。

2) 活性化可能の各管理対象(MO)クラスに関して、5.2節に記述され付属資料 A CORBA IDLで定義されている管理対象工場（*ManagedObjectFactory*）インタフェースから（直接的／間接的に）得られる工場インタフェースを定義しなければならない。

3) 適切と思われる個所には付属資料 B CORBA IDLにおいて定義されている定数を使用しなければならない。

4) 適切と思われる個所には5.3節で記述され付属資料 A CORBA IDLにおいて定義されている通知(notifications)を使用しなければならない。

5) 6節に記述されているCORBA TMN管理対象定義に関する慣例を守らなければならない。

6) 8節に記述されているIDL慣例を守らなければならない。

7) 本標準に定義されている通知が適用不可の場合、"Notifications"インタフェース上の方法により通知の定義を行うこと。

8)　条件付パッケージのパーツである属性とアクションの識別に関するNO<package name> exception を定義し利用しなければならない。

9)　管理対象によりサポートされる通知の識別に関して本標準に定義されているマクロを使用しなければならない。

10)　適切と思われる個所には6.3.5節にある一般属性型に関する定義を使用しなければならない。

11)　許容可能な収容関係を識別するためのモジュールを結合するIDL名を定義しなければならない。

12)　他の一般属性が使用されているモジュールを参照するその準拠を宣言しなければならない。

13)　GDMOからの変換によるIDLモデルの場合７節で定義されているGDMO IDL対応規則に従わなければならない。

9.2　システム適合

本標準に対し適合請求する実装は：

1) 5.1 節に記述の管理対象インタフェースの全能力をサポートしなければならない。

2)　6.9 節に記述の生成・操作・振る舞いをサポートしなければならない。

9.3　適合宣言ガイドライン

本ガイドラインのユーザは適合宣言を書くときに注意を要する。IDL モジュールは名前空間として使用されるので、OMG の IDL 規則で許されているように、それらはファイルを横断してスプリットできる。すなわち、モジュールが拡張されたとき、名前は変わらない。その代わり新規 IDL ファイルを簡単に追加できる。簡単にいうと、適合宣言におけるモジュール名は IDL インタフェースのセットを識別するのに十分ではない。適合宣言は、IDL の正しい版の識別を確実にするため、文書と発行年次を明確にしなければならない。

付属資料 A　The Object Model CORBA IDL Module

```
/* This IDL code is meant to be stored in a file named "itut_x780.idl"
located in the search path used by IDL compilers on your system. */
#ifndef ITUT_X780_IDL
#define ITUT_X780_IDL
#include <CosNaming.idl>
#include <CosTime.idl>
#include <itut_x780Const.idl>
#pragma prefix "itu.int"
/* Most comments in this file are formatted to be parsed by an IDL-to-HTML
converter such as idldoc or orbacus hidl. */
```

// MODULE itut_x780

```
/** This module provides the fundamental capabilities for implementing network
management interfaces and defines the "managed object" interface. The
interfaces below are modeled after the managed object specifications
found in the ITU-T CMIP specification document X.721. */
module itut_x780 {
```

// IMPORTED TYPES

```
        // Types imported from CosNaming
        typedef CosNaming::Name NameType;
        // Types imported from CosTime
        typedef TimeBase::UtcT UtcT;
```

// FORWARD DECLARATIONS AND TYPEDEFS

```
        /** International strings are strings of wide (16 bit unicode)
        characters. */
        typedef wstring Istring;
        /** Istring Sets are just sets of Istrings */
        typedef sequence <Istring> IstringSetType;
        /** Additional Text Type is often used in notifications to convey a
        text explanation for the notification.
        */
        typedef Istring AdditionalTextType;
```

```
/** Avalibility Type is used in a sequence to indicate the
availability of a resource. Zero or more of these conditions may be
indicated.
*/
typedef short AvailabilityStatusType;
const AvailabilityStatusType inTest = 0;
const AvailabilityStatusType failed = 1;
const AvailabilityStatusType powerOff = 2;
const AvailabilityStatusType offLine = 3;
const AvailabilityStatusType offDuty = 4;
const AvailabilityStatusType dependency = 5;
const AvailabilityStatusType degraded = 6;
const AvailabilityStatusType notInstalled = 7;
const AvailabilityStatusType logFull = 8;
/** Availability status is used to indicate the availability of a
resource. It is represented as a sequence of integers because several
of the conditions may exist at once.
*/
typedef sequence<AvailabilityStatusType> AvailabilityStatusSetType;
/** Backed Up Status Type is used to indicate if an object has a back
up. */
typedef boolean BackedUpStatusType;
/** BitStrings are used to hold strings of bits. They may be of any
length. */
typedef sequence<octet> BitString;
/** Control Status Type is used in a sequence to indicate the
control status of a resource. Zero or more of these may be indicated.
*/
typedef short ControlStatusType;
const ControlStatusType subjectToTest = 0;
const ControlStatusType partOfServicesLocked = 1;
const ControlStatusType reservedForTest = 2;
const ControlStatusType suspended = 3;
/** Control status set is used to indicate the control status of a
resource. It is represented as a sequence of integers because several
of the conditions may exist at once.
*/
typedef sequence<ControlStatusType> ControlStatusSetType;
/** Generalized time is a basic ASN.1 type. It is usually represented
as a string in computing languages but it has certain, parseable
formats. The 3 possible forms are: <ol><li>
Local time only. "YYYYMMDDHHMMSS.fff", where the optional fff is
accurate to three decimal places, <li>
Universal time (UTC time) only. "YYYYMMDDHHMMSS.fffZ", and <li>
Difference between local and UTC times. "YYYYMMDDHHMMSS.fff+-HHMM".
</ol>
The options for representing this in IDL seem to be either a string or
the UtcT structure from the CORBA Time Service. UtcT makes it a little
easier to compare times from different zones, but requires managed
systems to know their time zones. UtcT was picked.
*/
typedef UtcT GeneralizedTimeType;
/** External Time is generalized time. */
typedef GeneralizedTimeType ExternalTimeType;
/** Forward declaration. CORBA uses object references
of type "object" to identify objects. These are used instead of ASN.1
object instances. For network management interfaces, all objects will
inherit from the "ManagedObject" interface. */
interface ManagedObject;
/** MO Set is a set of ManagedObject references. */

typedef sequence <ManagedObject> MOSetType;
/** MO Seq is a sequence of ManagedObject references. */
typedef sequence <ManagedObject> MOSeqType;
/** A set of names is definded as a sequence of names. */
typedef sequence <NameType> NameSetType;
/** Notification IDs are long integers. */
typedef long NotifIDType;
/** This defines a set of notification IDs. */
typedef sequence <long> NotifIDSetType;
```

```
/** Procedural Status Type is used in a sequence to indicate the
procedural status of a resource. Zero or more of these may be
indicated.
*/
typedef short ProceduralStatusType;
const ProceduralStatusType initializationRequired = 0;
const ProceduralStatusType notInitialized = 1;
const ProceduralStatusType initializing = 2;
const ProceduralStatusType reporting = 3;
const ProceduralStatusType terminating = 4;
/** Procedural Status Set is used to indicate the procedural status of
a resource. It is represented as a sequence of integers because
several of the conditions may exist at once.
*/
typedef sequence<ProceduralStatusType> ProceduralStatusSetType;
/** ScopedName is just a string. */
typedef string ScopedNameType;
/** Scoped Name Sets are simply sets of Scoped Names. */
typedef sequence <ScopedNameType> ScopedNameSetType;
/** In CORBA, strings containing scoped names are used to identify
object classes (actually, "interfaces"). */
typedef ScopedNameType ObjectClassType;
/** Object Class Set is a set of object classes */
typedef sequence <ObjectClassType> ObjectClassSetType;
/** Name Binding Modules are identified with scoped names. */
typedef ScopedNameType NameBindingType;
/** StartTimeType is used to specify a time when something starts.
It is often paired with a StopTimeType to control the activation of
some function.
*/
typedef GeneralizedTimeType StartTimeType;
/** String sets are sets of strings. */
typedef sequence <string> StringSetType;
/** System Labels are strings used to identify systems. */
typedef string SystemLabelType;
/** Unknown status is used to indicate if the status of a resource is
not known. A value of true indicates the status is unknown. */
typedef boolean UnknownStatusType;
```

// ENUMERATED TYPES

```
/* The following state objects are used in many interfaces and parallel
the state objects in CMIP standards. */
/** Administrative State is read/write. A "locked" object is usually
one that may not be changed or one which is not providing service.
Setting the Admininstrative State of an object to "shuttingDown" begins
the shutdown process for that object. */
enum AdministrativeStateType {locked, unlocked, shuttingDown};
/** Operational State is read only. It simply reports the current
capability of the object to provide service. */
enum OperationalStateType {disabled, enabled};
/** Usage state is read only. If "idle," the resource is completely
unused. If "busy," the total capacity of the resource is in use.
"Active" is in between. */
enum UsageStateType {idle, active, busy};
/** Delete Policy indicates if an object can be deleted and if so if
any contained objects should automatically be deleted. Since objects
must not be orphaned, if an object has a delete policy of
"deleteOnlyIfNoContainedObjects" the object must not be deleted if it
has contained objects. A value of "deleteContainedObjects" means if
the object is deleted its contained objects should also be deleted. */
enum DeletePolicyType {notDeletable, deleteOnlyIfNoContainedObjects,
                deleteContainedObjects};
/** PerceivedSeverity reports the severity of an alarm. "Indeterminate"
is used when it is not possible to assign one of the other values */
enum PerceivedSeverityType {indeterminate, critical, major, minor,
                                warning, cleared};
/** Source Indicator is used in many notifications. It identifies
whether the notification is a result of a management operation or
something that occurred on the managed system. */
```

```
enum SourceIndicatorType {resourceOperation, managementOperation,
                                unknown};
/** The standby status attribute is single-valued and read-only.
The value is only meaningful when the back-up relationship role exists.
If "hot standby" the resource is not providing service, but is
operating in synchronism with another resource that is to be backed-up.
If "cold standby" the resource is to back-up another resource, but is
not synchronized with that resource. If "providing service" the back-up
resource is providing service and is backing up another resource.
*/
enum StandbyStatusType {hotStandby, coldStandby, providingService};
/** Stop times are used to specify when some function should cease.
There are normally two choices, the function runs continually (in
which case no actual time is specified) or the function ends at
a specified time.
*/
enum StopTimeChoice {specific, continual};
/** Threshold indication describes if the threshold crossed was in the
up or down direction. */
enum ThresholdIndicationType {up, down};
/** TrendIndication values indicate if some observed condition is
getting better, worse, or not changing. */
enum TrendIndicationType {lessSevere, noChange, moreSevere};
```

// STRUCTURES AND UNIONS

```
/* The structures defined below are used to pass values that may be
optionally included. For some types of values, like strings, lists,
and pointers, it is easy to tell if the value is included. For others,
like enumerations, numbers, and structures, it is not. */
/** AdministrativeStateTypeOpt is an optional type. If the
discriminator is true the value is present, otherwise the value is
null. */
union AdministrativeStateTypeOpt switch (boolean) {
        case TRUE:      AdministrativeStateType value;
};
/** BooleanTypeOpt is an optional type. If the discriminator is
true the value is present, otherwise the value is null. */
union BooleanTypeOpt switch (boolean) {
        case TRUE:      boolean value;
};
/** FloatTypeOpt is an optional type. If the discriminator is
true the value is present, otherwise the value is null. */
union FloatTypeOpt switch (boolean) {
        case TRUE:      float   value;
};
/** LongTypeOpt is an optional type. If the discriminator is
true the value is present, otherwise the value is null. */
union LongTypeOpt switch (boolean) {
        case TRUE:      long    value;
};
/** OperationalStateTypeOpt is an optional type. If the discriminator
is true the value is present, otherwise the value is null. */
union OperationalStateTypeOpt switch (boolean) {
        case TRUE:      OperationalStateType    value;
};
/** ShortTypeOpt is an optional type. If the discriminator is
true the value is present, otherwise the value is null. */
union ShortTypeOpt switch (boolean) {
        case TRUE:      short   value;
};
/** TrendIndicationTypeOpt is an optional type. If the discriminator
is true the value is present, otherwise the value is null. */
union TrendIndicationTypeOpt switch (boolean) {
        case TRUE:      TrendIndicationType     value;
};
/** UnsignedShortTypeOpt is an optional type. If the discriminator is
the value is present, otherwise the value is null. */
union UnsignedShortTypeOpt switch (boolean) {
        case TRUE:      unsigned short value;
};
```

```
/** UsageStateTypeOpt is an optional type. If the discriminator is
true the value is present, otherwise the value is null. */
union UsageStateTypeOpt switch (boolean) {
        case TRUE:      UsageStateType  value;
};
/** Many times interface specifications need to define standard values
to be passed across the interface. Also, often the scheme used to
define these values needs to be extensible as new interfaces are
subclassed, so enumerations don't work well. CMIP uses OIDs, strings
of numbers that are often appended, in standards. To serve this
purpose, the Unique ID is used. It consists of two parts, a string
containing a scoped module name, and an integer value defined as a
constant within that module. These UIDs, and the ObjectClass type
defined above, replace ASN.1 OIDs. It is expected that each module
will contain a constant string named "moduleName" that contains the
name of the module for error-free use by the programmer. A null module
name will indicate a null value for the UID. <p>
Code to interpret a UID might look like the following code snippet:<br>
<code><pre>
UIDType pc;     // probable cause
...
if (pc.moduleName ==
        itut_x780::ProbableCauseConst::moduleName) //string compare
        switch (pc.value) {
        case itut_x780::ProbableCauseConst::adapterError:
        ...
        case
        itut_x780::ProbableCauseConst::applicationSubsystemFailure:
        ...
        case itut_x780::ProbableCauseConst::bandwidthReduced:
        ...
        }
else if (pc.moduleName == MyLocal::ProbableCauseConst::moduleName)
        switch (pc.value) {
        ...
        }
</pre></code>
@member moduleName      The scoped module name where values are
                        defined.
@member value           The value defined as a constant within the
                        module.
*/
struct UIDType {
        string moduleName;      // module where value is defined
        short value;            // constant within the module
};
typedef sequence <UIDType> UIDSetType;
/** Management Extension is a structure for flexibly reporting
information. It is typically used in the Additional Information field
of notifications.
@see <a href="#AdditionalInformationSetType">
AdditionalInformationSetType </a>
@member id      identifies the type of information
@member any     contains the actual information, type will depend on
                the value of the id member.
*/
struct ManagementExtensionType {
        UIDType id;             // identifies the type of info
        any     info;           // type will depend on id
};
/** Additional Information is a flexible way to report information that
does not fit into the structure of a notification. It contains a
sequence of a structure called "Management Extension". */
typedef sequence <ManagementExtensionType>
                        AdditionalInformationSetType;
/** An Attribute Value structure is used in a notification to report
the value of any attribute. The string used for the attribute's name
is the same as the name of the data member in the value object defined
for the object. In other words, it is the name of an attribute accessor
method minus the "get" or "set".
@member attributeName   the name of the attribute
```

```
@member value           contains the value of the attribute, type will
                        depend on the attributeName.
*/
struct AttributeValueType {
        string  attributeName;
        any     value;          // type will depend on the attribute
};
/** Attribute Value Sets are used to report attributes generically,
in a batch mode. */
typedef sequence <AttributeValueType> AttributeSetType;
/** An Attribute Value Change structure is used in a notification to
report an attribute that has been changed.
@see <a href="#AttributeValueType">AttributeValueType</a>
@member attributeName   the name of the attribute
@member oldValuethe old value, type will depend on the
                        attributeName
@member newValuethe new value, type will depend on the
                        attributeName.
*/
struct AttributeValueChangeType {
        string          attributeName;
        any             oldValue;       // type depends on attribute
        any             newValue;       // type depends on attribute
};
/** An Attribute Change Set is used to report the attributes that have
been changed in an attribute value change notification. */
typedef sequence <AttributeValueChangeType> AttributeChangeSetType;
/** A Correlated Notification is identified by the object that emitted
the notification and the notification ID. Both are included in case
the Notification IDs are not unique across objects.
@member source  Reference to object that emitted the correlated
                notification. If null, the correlated notifications
                are from the same source as the notification containing
                this data structure.
@member notifIDs IDs of the correlated notifications. Notification
                identifiers must be chosen to be unique across all
                notifications from a particular managed object
                throughout the time that correlation is significant.
*/
struct CorrelatedNotificationType {
        NameTypesource;
        NotifIDSetType  notifIDs;
};
/** Correlated Notification sets are sets of Correlated Notification
structures. */
typedef sequence <CorrelatedNotificationType>
                CorrelatedNotificationSetType;
/** ProbableCause, in CMIP standards, may be either an integer or GDMO
OID, a dot-notation string. The UID type is used instead. */
typedef UIDType ProbableCauseType;
/** Proposed Repair Actions are sets of unique identifiers. */
typedef UIDSetType ProposedRepairActionSetType;
/** Security Alarm Causes are unique identifiers. */
typedef UIDType SecurityAlarmCauseType;
/** Security Alarm Detector can indicate either a mechanism or a
specific object. According to X.721 a choice is made between one or
the other, though it is not clear why. (Actually, X.721 adds a third
choice for an AE-title which has no equivalent here.) Unless otherwise
indicated, then, at most one of the members will be non-null. Two
nulls may be sent if the managed system does not support this property.
@member mechanism       the scheme or function detecting the alarm, may
                        be null
@member obj             the object detecting the alarm, may be null
*/
struct SecurityAlarmDetectorType {
        UIDType         mechanism;      // may be null
        NameTypeobj;            // may be null
};
```

```
/** Service User
@member id      the id of the service user
@member details details about the service user, type will depend on id
*/
struct ServiceUserType {
        UIDType id;
        any     details;// value will depend on id
};
/** Service Providers share the same representation as Service Users.
*/
typedef ServiceUserType ServiceProviderType;
/** Specific Problems are sets of unique identifiers. */
typedef UIDSetType SpecificProblemSetType;
/** A Stop Time Type is used to indicate when some function should
cease. In the specific case, an actual time is given. In the
continual case, the function runs continually and no value is
carried in this union.
*/
union StopTimeType switch (StopTimeChoice) {
        case specific:  GeneralizedTimeType time;
        /* case continual carries NULL value */
};
/** A SuspectObject identifies an object that may be the cause of a
failure. It is usually a component of a SuspectObjectList.
@member objectClass             Object class of the suspect object
@member suspectObjectInstance   Object instance of the suspect object
@member failureProbability      Optional failure responsibility
                                probability from 1 to 100
*/
struct SuspectObjectType {
        ObjectClassType         objectClass;
        ManagedObject           suspectObjectInstance;
        UnsignedShortTypeOpt    failureProbability;
};
/** Suspect Object Lists are used to identify objects that may be the
cause of a failure.
*/
typedef sequence<SuspectObjectType> SuspectObjectSetType;
/** Threshold Level Indication describes multi-level threshold
crossings. Up is the only permitted choice for a counter. In ASN.1,
if indication is "up", low value is optional.
@member indication      indicates up or down direction of crossing.
@member low             the low observed value.
@member high            the high observed value.
*/
struct ThresholdLevelIndType {
        ThresholdIndicationType         indication;
        FloatTypeOpt                    low;    // observed value
        float                           high;   // observed value
};
/** Threshold Level Ind Type Opt is an optional type. If the
discriminator is true the value is present, otherwise the value is
null. */
union ThresholdLevelIndTypeOpt switch (boolean) {
        case TRUE:      ThresholdLevelIndType   value;
};
/** Threshold Information indicates some guage or counter attribute
passed a set threshold. The structure differs from X.721 some to
simplify the syntax.
@member attributeID     Identifies the attribute that crossed the
                        threshold. Actually, it is an operation name
                        on an interface minus the "get" or "set". The
                        interface on which the operation is defined is
                        included elsewhere in the notification as
                        ObjectClass. A Null value indicates the entire
                        structure is null.
@member observedValue   Attributes that are of type integer will be
                        converted to floats.
@member thresholdlevel  This parameter is for multi-level threhsolds.
                        Optional.
@member armTime         May be null(0). */
```

```
struct ThresholdInfoType {
        string                          attributeID;
        float                           observedValue;
        ThresholdLevelIndTypeOpt        thresholdLevel;
        ExternalTimeType        armTime;
};
```

// EXCEPTIONS

```
/** Application error info types are passed back in managed object
exceptions.
@member error   A unique identifier identifying the problem.
@member details A text message with additional information about the
problem.
*/
valuetype ApplicationErrorInfoType {
        public UIDType          error;
        public Istring          details;
};
/** Create error info types are passed back in managed object create
exceptions. They extend application error info types.
@member relatedObjects  objects that have some relationship to the
                        object to be created that somehow prevented the
                        creation.
@member attributeList   the values that would have been assigned to the
                        created object. These may hold some key to why
                        the object could not be created.
*/
valuetype CreateErrorInfoType : ApplicationErrorInfoType {
        public MOSetTyperelatedObjects;
        public AttributeSetType attributeList;
        };
/** Delete error info types are passed back in managed object delete
exceptions. They extend application error info types.
@member relatedObjects  objects that have some relationship to the
                        object to be deleted that somehow prevented the
                        deletion.
@member attributeList   the attribute values assigned to the object to
                        be deleted. These may hold some key to why the
                        object could not be deleted.
*/
valuetype DeleteErrorInfoType : ApplicationErrorInfoType {
        public MOSetTyperelatedObjects;
        public AttributeSetType attributeList;
};
/** A package error info type is a special create error. It will be
passed back in a managed object create exception as a create error. If
the UID error code matches the package error info type, the client
application may narrow the value type from create error info type to
package error info type to access the additional information.
@member packagesthe list of requested packages that conflicted
                        or could not be supported.
*/
valuetype PackageErrorInfoType : CreateErrorInfoType {
        public StringSetType    packages;
};
/** Application error exceptions may be raised on any managed object
operation to identify a problem preventing the operation from being
completed. */
exception ApplicationError { ApplicationErrorInfoType info; };
/** Create error exceptions may be raised on any managed object create
operation to identify a problem preventing the object from being
completed. */
exception CreateError { CreateErrorInfoType info; };
/** Delete error exceptions may be raised by a managed object in
response to an attempt to delete the object. They may also be raised
by the terminator service. */
exception DeleteError { DeleteErrorInfoType info; };
/** Invalid length exceptions are raised when an invalid length is
supplied on an operation invocation. */
exception InvalidLength { long length; };
```

```
/** Invalid string exceptions are raised when an invalid string is
supplied on an operation invocation. */
exception InvalidString {};
```

// VALUE TYPES

```
/** Bit string value types are used to represent bit strings with
associated semantic tags representing the bit string positions. */
valuetype BitStringValue {
        /** The state of a bit string is kept in a data member
        of type BitString (sequence of octets). The first octed
        is the count of unused bits in the last octed. */
        public BitString bitStringVal;
        /** This initializer shall set all bit positions to '0' */
        factory initValue (in unsigned long number_of_bits);
        /** This initializer shall create a new BitStringValue with
        the same length and value as the supplied BitString. */
        factory InitFromBitString (in BitString desiredValue);
        // local operations
        /** This local operation returns 0 or 1 for the bit value
        at the specified position. If the position requested is
        beyond the length of the bit string an invalid length
        exception shall be raised. */
        short getBit (in unsigned long position)
                raises (InvalidLength);
        /** This local operation is used to set the value of the
        bit at the requested position. If the position requested is
        beyond the length of the bit string an invalid length
        exception shall be raised. If the new_bit_value is 0,
        the bit shall be set to 0, otherwise it shall be set to 1. */

        void setBit (in unsigned long position, in short new_bit_value)
                raises (InvalidLength);
        /** This local operation returns the number of bits in the bit
        string. Since the first octet contains the number of unused
        bits in the last octet, and the last octet may contain unused
        bits, the value returned is
        (NumberOfOctets – 2)*8 + (8 – firstOctetVal) */
        unsigned long length ();
        /** This local operation returns the value of a bit string as a
        character string. Each bit of value 0 shall be represented as
        a '0' character, and each bit of value 1 shall be represented
        as a '1' character. A 'B' character shall be appended to the
        end of the string ("1001011B").
        string asString ();
        /** This local operation sets the value of the bit string given
        a character string. Each '0' character in the string is
        converted to a bit of value 0, and each '1' character is
        converted to a 1. If the string has any characters other than
        '0', '1', or a terminating 'B', the InvalidString exception
        shall be raised. */
        void setFromString (in string string_value)
                raises (InvalidString);
};
```

// MANAGED OBJECT INTERFACE

```
/** This valuetype object contains members for each of the attributes
accessible on this interface. */
valuetype ManagedObjectValueType {
        public NameType                 name;
        public ObjectClassType          objectClass;
        public StringSetType            packages;
        public SourceIndicatorType      creationSource;
        public DeletePolicyType         deletePolicy;
};
/** The Managed Object interface is intended to be the base interface
from which all other managed object interfaces inherit. It is a
central place to specify basic functions which all managed objects are
expected to support. */
```

```
interface ManagedObject {
        /** This method returns the fully-qualified name for the
        object. This method is used rather than having a "get*ID"
        method defined for each interface, as is done in CMIP
        specifications. This will ensure that objects have only a
        single operation to retrieve names when they are sub-classed.
        <p>
        The response is a sequence of name component structures,
        starting with the name assigned to the "local root" naming
        context under which this object is contained. The client may
        find the superiors of this object by removing components from
        the tail end of this sequence and performing a resolve
        operation on the first part of the name. */
        NameType nameGet()
                raises (ApplicationError);
        /** This method returns the scoped name of the most-specific
        class of the interface (e.g. "EquipmentR1"). */
        ObjectClassType objectClassGet()
                raises (ApplicationError);
        /** This method returns a list of all the conditional packages
        supported by this instance. */
        StringSetType packagesGet ()
                raises (ApplicationError);
        /** This method returns an indication of how the object was
        created. */
        SourceIndicatorType creationSourceGet()
                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. */
        DeletePolicyType deletePolicyGet ()
                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 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. */
        ManagedObjectValueType attributesGet (
                inout StringSetType attributeNames)
                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> */
        void destroy()
                raises (ApplicationError, DeleteError);
}; // end of ManagedObject interface
```

// MANAGED OBJECT FACTORY INTERFACE

```
/** This interface defines the generic managed object factory
interface. All Managed Object factories should inherit from this
interface. <p>
In addition to providing the means for creating objects by management
operation, the factories are assumed to take responsibility for
maintaining the integrity of the naming tree by creating name bindings
for the objects they create. <p>
Currently, this interface is null. It is included, however, as a
placeholder for capabilities that must be supported by all managed
object factories.
*/
interface ManagedObjectFactory {

}; // end of ManagedObjectFactory interface
```

// NOTIFICATIONS INTERFACE

```
/** This interface contains the definitions of notifications emitted by
many managed objects. <p>
The use of "typed" notifications is done here so that the notifications
can be documented in IDL and to support typed notifications for those
manager and managing systems that wish to use them. Note that the
OMG's Notification Service supports both structured and typed
notifications. It is not clear if implementations of the Notification
Service will support translation between them. It is expected that the
implementation agreement between the managing and managed system will
specify the use of structured or typed notifications. <p>
Notification users wishing to use typed notifications need only support
the interfaces below. Notification publishers and subscribers wishing
to use structured notifications based on the operations defined below
should follow these rules for constructing and reading the notification
structure: <ul><li>
The domain_type string in the fixed header of the structure should be
set to "telecommunications". <li>
The event_type string in the fixed header of the structure should be
set to the scoped name of the operation. For example, for the
Attribute Value Change notification defined below this field would be
"itut_x780::Notifications::attributeValueChange". <li>
The event_name string in the fixed header of the structure is not used
by this framework. It can be set to null or used for other purposes.
<li>
Optional header fields may be included to support features like Quality
of Service as appropriate. <li>
Each parameter in the operation should be placed in a name-value pair
in the filterable body portion of the notification. The fd_name string
of this pair shall be set to the name of the parameter and the type
placed in the associated fd_value will be the type specified for the
parameter. For example, each of the notifications defined below has a
parameter named "eventTime" that is an "ExternalTimeType." This
parameter would be placed in the filterable data portion of the event.
The fd_name string of this pair would be set to "eventTime" and
fd_value would contain an ExternalTimeType value. <li>
The remainder of the body of the notification (the unfilterable part)
should be null. </ul>
Unfortunately, typed notifications are mapped to notification
structures differently, so if one system wants to use typed
notifications and the other structured, the structured notification
user must be aware of how the CORBA Notification Service translates
typed notifications to structured notifications. See the specification
for details. In short, however, each of the parameters in the
operations below will be converted into a name-value pair in the
filterable data protion of the structured notification. Also, the
event_type field in the fixed header of the structured notification
will be set to the special value "%TYPED" and the domain_type field
will be an empty string. Finally, a name-value pair will be added as
the first element in the filterable data portion of the notification
with the name "operation". The value associated with this name will be
```

a string with the value set to the scoped name of the operation used to emit the notification

```
      (e.g. itut_x780::Notifications::attributeValueChange). <p>
      Also, structured notification publishers may exclude notification
      parameters that are marked "optional" or are of an optional type (a
      type name ending in "TypeOpt." This should be done for efficiency.
      This will, however, preclude the automatic conversion of structured
      notifications to typed, so managers must be capable of accepting
      structured notifications. (They do not strictly have to support typed
      notifications, but if managed systems emit typed notifications managers
      should accept them rather than translations because it will be more
      efficient.) If an "optional" parameter is included in a notification,
      the "optional" type (discriminated union) must be used. <p>
      Parameters named "operation" should be avoided in notification
      operations to support the use of typed notifications. While the
      notification channel should be able to differentiate the real parameter
      from the one added based on their positions in the filterable data
      list, it could have an impact on filtering as the default filtering
      language does not have a way to differentiate parameters based on
      position. <p>
      Because the scoped operation name is placed in either the type_name
      string (when structured notifications are used) or a filterable body
      name-value pair with the name "operation" (when typed notifications are
      used), there is no "event type" parameter explicitly included in any of
      the notification data structures. */
interface Notifications {
      /** An Attribute Value Change notification is used to report changes to
      the attributes of an object such as addition or deletion of members to
      one or more set-valued attributes and replacement of the value of one
      or more attributes.
      @param eventTime        Managed system's current time.
      @param source                   Object emitting notification.
      @param sourceClass              Actual class of source object.
      @param notificationIdentifier   A unique identifier for this
                                      notification. Must be unique for
                                      an object instance. (Optional in X.721
                                      but not here. See text for
                                      discussion of possible implications)
      @param correlatedNotifications  List of correlated notifications.
                                      Optional. Zero length sequence
                                      indicates absence of this parameter.
      @param additionalText           Text message. Optional. Zero length
                                      string indicates absence of this
                                      parameter.
      @param additionalInfo           Optional. Zero length sequence
                                      indicates absence of this parameter.
      param sourceIndicator           Cause of event. Optional. Use
                                      "unknown" if not supported.
      @param attributeChanges         Changed attributes
      */
      void attributeValueChange (
              in ExternalTimeType                   eventTime,
              in NameType                           source,
              in ObjectClassType                    sourceClass,
              in NotifIDType                        notificationIdentifier,
              in CorrelatedNotificationSetTypecorrelatedNotifications,
              in AdditionalTextType                 additionalText,
              in AdditionalInformationSetType       additionalInfo,
              in SourceIndicatorType                sourceIndicator,
              in AttributeChangeSetType             attributeChanges
      );
      /** A Communications Alarm notification is used to report when an
      object detects a communications error.
      @param eventTime        Managed system's current time.
      @param source                   Object emitting notification.
      @param sourceClass              Actual class of source object.
      @param notificationIdentifier   A unique identifier for this
                                      notification. Must be unique for
                                      an object instance. (Optional in X.721
                                      but not here. See text for
                                      discussion of possible implications)
```

```
@param correlatedNotifications  List of correlated notifications.
                                Optional. Zero length sequence
                                indicates absence of this parameter.
@param additionalText           Text message. Optional. Zero length
                                string indicates absence of this
                                parameter.
@param additionalInfo           Optional. Zero length sequence
                                indicates absence of this parameter.
@param probableCause
@param specificProblems         Optional. Zero length sequence
                                indicates absence of this parameter.
@param perceivedSeverity
@param backedUpStatus           "True" if backed up
@param backUpObject             Will be null if backedUpStatus is
                                "false"
@param trendIndication          Optional. See type for details.
@param thresholdInfo            Optional. See type for details.
@param stateChangeDefinition    Optional. Zero length sequence
                                indicates absence of this parameter.
@param monitoredAttributes      Optional. Zero length sequence
                                indicates absence of this parameter.
@param proposedRepairActions    Optional. Zero length sequence
                                indicates absence of this parameter.
@param alarmEffectOnService     True if alarm is service effecting.
@param alarmingResumed          True if alarming was just resumed,
                                possibly resulting in delayed reporting
                                of an alarm
@param suspectObjectListObjects possibly involved in failure.
*/
void communicationsAlarm (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetTypecorrelatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in ProbableCauseType                    probableCause,
        in SpecificProblemSetType               specificProblems,
        in PerceivedSeverityType        perceivedSeverity,
        in BooleanTypeOpt                       backedUpStatus,
        in NameType                             backUpObject,
        in TrendIndicationTypeOpt               trendIndication,
        in ThresholdInfoType                    thresholdInfo,
        in AttributeChangeSetType               stateChangeDefinition,
        in AttributeSetType                     monitoredAttributes,
        in ProposedRepairActionSetType          proposedRepairActions,
        in BooleanTypeOpt                       alarmEffectOnService,
        in BooleanTypeOpt                       alarmingResumed,
        in SuspectObjectSetType                 suspectObjectList
);
/** An Environmental Alarm notification is used to report a problem in
the environment.
@param eventTime        Managed system's current time.
@param source                   Object emitting notification.
@param sourceClass              Actual class of source object.
@param notificationIdentifier   A unique identifier for this
                                notification. Must be unique for
                                an object instance. (Optional in X.721
                                but not here. See text for
                                discussion of possible implications)
@param correlatedNotifications  List of correlated notifications.
                                Optional. Zero length sequence
                                indicates absence of this parameter.
@param additionalText           Text message. Optional. Zero length
                                string indicates absence of this
                                parameter.
@param additionalInfo           Optional. Zero length sequence
                                indicates absence of this parameter.
@param probableCause
```

```
@param specificProblems         Optional. Zero length sequence
                                indicates absence of this parameter.
@param perceivedSeverity
@param backedUpStatus           "True" if backed up
@param backUpObject             Will be null if backedUpStatus is
                                "false"
@param trendIndication          Optional. See type for details.
@param thresholdInfo            Optional. See type for details.
@param stateChangeDefinition    Optional. Zero length sequence
                                indicates absence of this parameter.
@param monitoredAttributes      Optional. Zero length sequence
                                indicates absence of this parameter.
@param proposedRepairActions    Optional. Zero length sequence
                                indicates absence of this parameter.
@param alarmEffectOnService     True if alarm is service effecting.
@param alarmingResumed          True if alarming was just resumed,
                                possibly resulting in delayed reporting
                                of an alarm
@param suspectObjectListObjects possibly involved in failure.
*/
void environmentalAlarm (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetTypecorrelatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in ProbableCauseType                    probableCause,
        in SpecificProblemSetType               specificProblems,
        in PerceivedSeverityType        perceivedSeverity,
        in BooleanTypeOpt                       backedUpStatus,
        in NameType                             backUpObject,
        in TrendIndicationTypeOpt               trendIndication,
        in ThresholdInfoType                    thresholdInfo,
        in AttributeChangeSetType               stateChangeDefinition,
        in AttributeSetType                     monitoredAttributes,
        in ProposedRepairActionSetType          proposedRepairActions,
        in BooleanTypeOpt                       alarmEffectOnService,
        in BooleanTypeOpt                       alarmingResumed,
        in SuspectObjectSetType                 suspectObjectList
);
/** An Equipment Alarm notification is used to report a failure in the
equipment.
@param eventTime        Managed system's current time.
@param source                   Object emitting notification.
@param sourceClass              Actual class of source object.
@param notificationIdentifier   A unique identifier for this
                                notification. Must be unique for
                                an object instance. (Optional in X.721
                                but not here. See text for
                                discussion of possible implications)
@param correlatedNotifications  List of correlated notifications.
                                Optional. Zero length sequence
                                indicates absence of this parameter.
@param additionalText           Text message. Optional. Zero length
                                string indicates absence of this
                                parameter.
@param additionalInfo           Optional. Zero length sequence
                                indicates absence of this parameter.
@param probableCause
@param specificProblems         Optional. Zero length sequence
                                indicates absence of this parameter.
@param perceivedSeverity
@param backedUpStatus           "True" if backed up
@param backUpObject             Will be null if backedUpStatus is
                                "false"
@param trendIndication          Optional. See type for details.
@param thresholdInfo            Optional. See type for details.
@param stateChangeDefinition    Optional. Zero length sequence
                                indicates absence of this parameter.
```

```
@param monitoredAttributes      Optional. Zero length sequence
                                indicates absence of this parameter.
@param proposedRepairActions    Optional. Zero length sequence
                                indicates absence of this parameter.
@param alarmEffectOnService     True if alarm is service effecting.
@param alarmingResumed          True if alarming was just resumed,
                                possibly resulting in delayed reporting
                                of an alarm
@param suspectObjectListObjects possibly involved in failure.
*/
void equipmentAlarm (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetTypecorrelatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in ProbableCauseType                    probableCause,
        in SpecificProblemSetType               specificProblems,
        in PerceivedSeverityType        perceivedSeverity,
        in BooleanTypeOpt                       backedUpStatus,
        in NameType                             backUpObject,
        in TrendIndicationTypeOpt               trendIndication,
        in ThresholdInfoType                    thresholdInfo,
        in AttributeChangeSetType               stateChangeDefinition,
        in AttributeSetType                     monitoredAttributes,
        in ProposedRepairActionSetType          proposedRepairActions,
        in BooleanTypeOpt                       alarmEffectOnService,
        in BooleanTypeOpt                       alarmingResumed,
        in SuspectObjectSetType                 suspectObjectList
);
/** An Integrity Violation notification is used to report that a
potential interruption in information flow has occurred such that
information may have been illegally modified, inserted or deleted.
@param eventTime        Managed system's current time.
@param source                   Object emitting notification.
@param sourceClass              Actual class of source object.
@param notificationIdentifier   A unique identifier for this
                                notification. Must be unique for
                                an object instance. (Optional in X.721
                                but not here. See text for
                                discussion of possible implications)
@param correlatedNotifications  List of correlated notifications.
                                Optional. Zero length sequence
                                indicates absence of this parameter.
@param additionalText           Text message. Optional. Zero length
                                string indicates absence of this
                                parameter.
@param additionalInfo           Optional. Zero length sequence
                                indicates absence of this parameter.
@param securityAlarmCause
@param securityAlarmSeverity    Clears allowed? X.721 appears to
                                restrict the "cleared" value on this
                                alarm but clears should be allowed.
@param securityAlarmDetector
@param serviceUser
@param serviceProvider
*/
void integrityViolation (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetTypecorrelatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in SecurityAlarmCauseType               securityAlarmCause,
        in PerceivedSeverityType        securityAlarmSeverity,
        in SecurityAlarmDetectorType            securityAlarmDetector,
        in ServiceUserType                      serviceUser,
```

```
        in ServiceProviderType                  serviceProvider
);
/** An Object Creation notification is used to report the creation of a
managed object to another open system. Note that the source field
should be set to the created object, not the factory.
@param eventTime        Managed system's current time.
@param source                   Object emitting notification.
@param sourceClass              Actual class of source object.
@param notificationIdentifier   A unique identifier for this
                                notification. Must be unique for
                                an object instance. (Optional in X.721
                                but not here. See text for
                                discussion of possible implications)
@param correlatedNotifications  List of correlated notifications.
                                Optional. Zero length sequence
                                indicates absence of this parameter.
@param additionalText           Text message. Optional. Zero length
                                string indicates absence of this
                                parameter.
@param additionalInfo           Optional. Zero length sequence
                                indicates absence of this parameter.
@param sourceIndicator          Cause of event. Optional. Use
                                "unknown" if not supported.
@param attributeSet             Attribute values. Optional. Zero length
                                sequence indicates absence of this
                                parameter.
*/
void objectCreation (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetTypecorrelatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in SourceIndicatorType                  sourceIndicator,
        in AttributeSetType                     attributeList
);
/** An Object Deletion notification is used to report the deletion of a
managed object. Note that the source field should be set to
the object being deleted.
@param eventTime        Managed system's current time.
@param source                   Object emitting notification.
@param sourceClass              Actual class of source object.
@param notificationIdentifier   A unique identifier for this
                                notification. Must be unique for
                                an object instance. (Optional in X.721
                                but not here. See text for
                                discussion of possible implications)
@param correlatedNotifications  List of correlated notifications.
                                Optional. Zero length sequence
                                indicates absence of this parameter.
@param additionalText           Text message. Optional. Zero length
                                string indicates absence of this
                                parameter.
@param additionalInfo           Optional. Zero length sequence
                                indicates absence of this parameter.
@param sourceIndicator          Cause of event. Optional. Use
                                "unknown" if not supported.
@param attributeSet             Attribute values. Optional. Zero length
                                sequence indicates absence of this
                                parameter.
*/
void objectDeletion (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetTypecorrelatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
```

```
        in SourceIndicatorType                  sourceIndicator,
        in AttributeSetType                     attributeList
);
/** An Operational Violation notification is used to report that the
provision of the requested service was not possible due to the
unavailability, malfunction or incorrect invocation of the service.
@param eventTime        Managed system's current time.
@param source                   Object emitting notification.
@param sourceClass              Actual class of source object.
@param notificationIdentifier   A unique identifier for this
                                notification. Must be unique for
                                an object instance. (Optional in X.721
                                but not here. See text for
                                discussion of possible implications)
@param correlatedNotifications  List of correlated notifications.
                                Optional. Zero length sequence
                                indicates absence of this parameter.
@param additionalText           Text message. Optional. Zero length
                                string indicates absence of this
                                parameter.
@param additionalInfo           Optional. Zero length sequence
                                indicates absence of this parameter.
@param securityAlarmCause
@param securityAlarmSeverity    Clears allowed? X.721 appears to
                                restrict the "cleared" value on this
                                alarm but clears should be allowed.
@param securityAlarmDetector
@param serviceUser
@param serviceProvider
*/
void operationalViolation (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetTypecorrelatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in SecurityAlarmCauseType               securityAlarmCause,
        in PerceivedSeverityType        securityAlarmSeverity,
        in SecurityAlarmDetectorType            securityAlarmDetector,
        in ServiceUserType                      serviceUser,
        in ServiceProviderType                  serviceProvider
);
/** A Physical Violation notification is used to report that a physical
resource has been violated in a way that indicates a potential security
attack.
@param eventTime        Managed system's current time.
@param source                   Object emitting notification.
@param sourceClass              Actual class of source object.
@param notificationIdentifier   A unique identifier for this
                                notification. Must be unique for
                                an object instance. (Optional in X.721
                                but not here. See text for
                                discussion of possible implications)
@param correlatedNotifications  List of correlated notifications.
                                Optional. Zero length sequence
                                indicates absence of this parameter.
@param additionalText           Text message. Optional. Zero length
                                string indicates absence of this
                                parameter.
@param additionalInfo           Optional. Zero length sequence
                                indicates absence of this parameter.
@param securityAlarmCause
@param securityAlarmSeverity    Clears allowed? X.721 appears to
                                restrict the "cleared" value on this
                                alarm but clears should be allowed.
@param securityAlarmDetector
@param serviceUser
@param serviceProvider
*/
```

```
void physicalViolation (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetTypecorrelatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in SecurityAlarmCauseType               securityAlarmCause,
        in PerceivedSeverityType        securityAlarmSeverity,
        in SecurityAlarmDetectorType            securityAlarmDetector,
        in ServiceUserType                      serviceUser,
        in ServiceProviderType                  serviceProvider
);
/** A Processing Error Alarm notification is used to report a
processing failure in a managed object.
@param eventTime        Managed system's current time.
@param source                   Object emitting notification.
@param sourceClass              Actual class of source object.
@param notificationIdentifier   A unique identifier for this
                                notification. Must be unique for
                                an object instance. (Optional in X.721
                                but not here. See text for
                                discussion of possible implications)
@param correlatedNotifications  List of correlated notifications.
                                Optional. Zero length sequence
                                indicates absence of this parameter.
@param additionalText           Text message. Optional. Zero length
                                string indicates absence of this
                                parameter.
@param additionalInfo           Optional. Zero length sequence
                                indicates absence of this parameter.
@param probableCause
@param specificProblems         Optional. Zero length sequence
                                indicates absence of this parameter.
@param perceivedSeverity
@param backedUpStatus           "True" if backed up
@param backUpObject             Will be null if backedUpStatus is
                                "false"
@param trendIndication          Optional. See type for details.
@param thresholdInfo            Optional. See type for details.
@param stateChangeDefinition    Optional. Zero length sequence
                                indicates absence of this parameter.
@param monitoredAttributes      Optional. Zero length sequence
                                indicates absence of this parameter.
@param proposedRepairActions    Optional. Zero length sequence
                                indicates absence of this parameter.
@param alarmEffectOnService     True if alarm is service effecting.
@param alarmingResumed          True if alarming was just resumed,
                                possibly resulting in delayed reporting
                                of an alarm
@param suspectObjectListObjects possibly involved in failure.
*/
void processingErrorAlarm (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetTypecorrelatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in ProbableCauseType                    probableCause,
        in SpecificProblemSetType               specificProblems,
        in PerceivedSeverityType        perceivedSeverity,
        in BooleanTypeOpt                       backedUpStatus,
        in NameType                             backUpObject,
        in TrendIndicationTypeOpt               trendIndication,
        in ThresholdInfoType                    thresholdInfo,
        in AttributeChangeSetType               stateChangeDefinition,
        in AttributeSetType                     monitoredAttributes,
        in ProposedRepairActionSetType          proposedRepairActions,
```

```
        in BooleanTypeOpt                       alarmEffectOnService,
        in BooleanTypeOpt                       alarmingResumed,
        in SuspectObjectSetType                 suspectObjectList
);
/** A Quality of Service Alarm notification is used to report a failure
in the quality of service of the managed object.
@param eventTime        Managed system's current time.
@param source                   Object emitting notification.
@param sourceClass              Actual class of source object.
@param notificationIdentifier   A unique identifier for this
                                notification. Must be unique for
                                an object instance. (Optional in X.721
                                but not here. See text for
                                discussion of possible implications)
@param correlatedNotifications  List of correlated notifications.
                                Optional. Zero length sequence
                                indicates absence of this parameter.
@param additionalText           Text message. Optional. Zero length
                                string indicates absence of this
                                parameter.
@param additionalInfo           Optional. Zero length sequence
                                indicates absence of this parameter.
@param probableCause
@param specificProblems         Optional. Zero length sequence
                                indicates absence of this parameter.
@param perceivedSeverity
@param backedUpStatus           "True" if backed up
@param backUpObject             Will be null if backedUpStatus is
                                "false"
@param trendIndication          Optional. See type for details.
@param thresholdInfo            Optional. See type for details.
@param stateChangeDefinition    Optional. Zero length sequence
                                indicates absence of this parameter.
@param monitoredAttributes      Optional. Zero length sequence
                                indicates absence of this parameter.
@param proposedRepairActions    Optional. Zero length sequence
                                indicates absence of this parameter.
@param alarmEffectOnService     True if alarm is service effecting.
@param alarmingResumed          True if alarming was just resumed,
                                possibly resulting in delayed reporting
                                of an alarm
@param suspectObjectListObjects possibly involved in failure.
*/
void qualityOfServiceAlarm (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetTypecorrelatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in ProbableCauseType                    probableCause,
        in SpecificProblemSetType               specificProblems,
        in PerceivedSeverityType        perceivedSeverity,
        in BooleanTypeOpt                       backedUpStatus,
        in NameType                             backUpObject,
        in TrendIndicationTypeOpt               trendIndication,
        in ThresholdInfoType                    thresholdInfo,
        in AttributeChangeSetType               stateChangeDefinition,
        in AttributeSetType                     monitoredAttributes,
        in ProposedRepairActionSetType          proposedRepairActions,
        in BooleanTypeOpt                       alarmEffectOnService,
        in BooleanTypeOpt                       alarmingResumed,
        in SuspectObjectSetType                 suspectObjectList
);
/** A Relationship Change notification is used to report the change in
the value of one or more relationship attributes of a managed object,
that result through either internal operation of the managed object or
via management operation.
@param eventTime        Managed system's current time.
@param source                   Object emitting notification.
```

```
@param sourceClass              Actual class of source object.
@param notificationIdentifier   A unique identifier for this
                                notification. Must be unique for
                                an object instance. (Optional in X.721
                                but not here. See text for
                                discussion of possible implications)
@param correlatedNotifications  List of correlated notifications.
                                Optional. Zero length sequence
                                indicates absence of this parameter.
@param additionalText           Text message. Optional. Zero length
                                string indicates absence of this
                                parameter.
@param additionalInfo           Optional. Zero length sequence
                                indicates absence of this parameter.
@param sourceIndicator          Cause of event. Optional. Use
                                "unknown" if not supported.
@param relationshipChanges      Changed relationship attributes
*/
void relationshipChange (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetTypecorrelatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in SourceIndicatorType                  sourceIndicator,
        in AttributeChangeSetType               relationshipChanges
);
/** A Security Violation notification is used to report that a security
attack has been detected by a security service or mechanism.
@param eventTime        Managed system's current time.
@param source                   Object emitting notification.
@param sourceClass              Actual class of source object.
@param notificationIdentifier   A unique identifier for this
                                notification. Must be unique for
                                an object instance. (Optional in X.721
                                but not here. See text for
                                discussion of possible implications)
@param correlatedNotifications  List of correlated notifications.
                                Optional. Zero length sequence
                                indicates absence of this parameter.
@param additionalText           Text message. Optional. Zero length
                                string indicates absence of this
                                parameter.
@param additionalInfo           Optional. Zero length sequence
                                indicates absence of this parameter.

@param securityAlarmCause
@param securityAlarmSeverity    Clears allowed? X.721 appears to
                                restrict the "cleared" value on this
                                alarm but clears should be allowed.
@param securityAlarmDetector
@param serviceUser
@param serviceProvider
*/
void securityViolation (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetTypecorrelatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in SecurityAlarmCauseType               securityAlarmCause,
        in PerceivedSeverityType        securityAlarmSeverity,
        in SecurityAlarmDetectorType            securityAlarmDetector,
        in ServiceUserType                      serviceUser,
        in ServiceProviderType                  serviceProvider
);
```

```
/** A State Change notification is used to report the change in the the
value of one or more state attributes of a managed object, that result
through either internal operation of the managed object or via
management operation.
@param eventTime        Managed system's current time.
@param source                   Object emitting notification.
@param sourceClass              Actual class of source object.
@param notificationIdentifier   A unique identifier for this
                                notification. Must be unique for
                                an object instance. (Optional in X.721
                                but not here. See text for
                                discussion of possible implications)
@param correlatedNotifications  List of correlated notifications.
                                Optional. Zero length sequence
                                indicates absence of this parameter.
@param additionalText           Text message. Optional. Zero length
                                string indicates absence of this
                                parameter.
@param additionalInfo           Optional. Zero length sequence
                                indicates absence of this parameter.
@param sourceIndicator          Cause of event. Optional. Use
                                "unknown" if not supported.
@param stateChanges             Changed state attributes.
*/
void stateChange (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetTypecorrelatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in SourceIndicatorType                  sourceIndicator,
        in AttributeChangeSetType               stateChanges
);
/** A Time Domain Violation notification is used to report that an
event has occurred at an unexpected or prohibited time.
@param eventTime        Managed system's current time.
@param source                   Object emitting notification.
@param sourceClass              Actual class of source object.
@param notificationIdentifier   A unique identifier for this
                                notification. Must be unique for
                                an object instance. (Optional in X.721
                                but not here. See text for
                                discussion of possible implications)
@param correlatedNotifications  List of correlated notifications.
                                Optional. Zero length sequence
                                indicates absence of this parameter.
@param additionalText           Text message. Optional. Zero length
                                string indicates absence of this
                                parameter.
@param additionalInfo           Optional. Zero length sequence
                                indicates absence of this parameter.
@param securityAlarmCause
@param securityAlarmSeverity    Clears allowed? X.721 appears to
                                restrict the "cleared" value on this
                                alarm but clears should be allowed.
@param securityAlarmDetector
@param serviceUser
@param serviceProvider
*/
void timeDomainViolation (
        in ExternalTimeType                     eventTime,
        in NameType                             source,
        in ObjectClassType                      sourceClass,
        in NotifIDType                          notificationIdentifier,
        in CorrelatedNotificationSetTypecorrelatedNotifications,
        in AdditionalTextType                   additionalText,
        in AdditionalInformationSetType         additionalInfo,
        in SecurityAlarmCauseType               securityAlarmCause,
        in PerceivedSeverityType        securityAlarmSeverity,
```

```
        in SecurityAlarmDetectorType            securityAlarmDetector,
        in ServiceUserType                      serviceUser,
        in ServiceProviderType                  serviceProvider
);
/** These constants define the names of the notifications declared
above and are provided to help reduce errors. */
const string attributeValueChangeTypeName =
        "itut_x780::Notifications::attributeValueChange";
const string communicationsAlarmTypeName =
        "itut_x780::Notifications::communicationsAlarm";
const string environmentalAlarmTypeName =
        "itut_x780::Notifications::environmentalAlarm";
const string equipmentAlarmTypeName =
        "itut_x780::Notifications::equipmentAlarm";
const string integrityViolationTypeName =
        "itut_x780::Notifications::integrityViolation";
const string objectCreationTypeName =
        "itut_x780::Notifications::objectCreation";
const string objectDeletionTypeName =
        "itut_x780::Notifications::objectDeletion";
const string operationalViolationTypeName =
        "itut_x780::Notifications::operationalViolation";
const string physicalViolationTypeName =
        "itut_x780::Notifications::physicalViolation";
const string processingErrorAlarmTypeName =
        "itut_x780::Notifications::processingErrorAlarm";
const string qualityOfServiceAlarmTypeName =
        "itut_x780::Notifications::qualityOfServiceAlarm";
const string relationshipChangeTypeName =
        "itut_x780::Notifications::relationshipChange";
const string securityViolationTypeName =
        "itut_x780::Notifications::securityViolation";
const string stateChangeTypeName =
        "itut_x780::Notifications::stateChange";
const string timeDomainViolationTypeName =
        "itut_x780::Notifications::timeDomainViolation";
/** These constants define the names of the parameters used in the
notifications declared above and are provided to help reduce errors.
*/
const string additionalInfoName = "additionalInfo";
const string additionalTextName = "additionalText";
const string alarmEffectOnServiceName = "alarmEffectOnService";
const string alarmingResumedName = "alarmingResumed";
const string attributeChangesName = "attributeChanges";
const string attributeListName = "attributeList";
const string backedUpStatusName = "backedUpStatus";
const string backUpObjectName = "backUpObject";
const string correlatedNotificationsName = "correlatedNotifications";
const string eventTimeName = "eventTime";
const string monitoredAttributesName = "monitoredAttributes";
const string notificationIdentifierName = "notificationIdentifier";
const string perceivedSeverityName = "perceivedSeverity";
const string probableCauseName = "probableCause";
const string proposedRepairActionsName = "proposedRepairActions";
const string relationshipChangesName = "relationshipChanges";
const string securityAlarmCauseName = "securityAlarmCause";
const string securityAlarmDetectorName = "securityAlarmDetector";
const string securityAlarmSeverityName = "securityAlarmSeverity";
const string serviceProviderName = "serviceProvider";
const string serviceUserName = "serviceUser";
const string sourceName = "source";
const string sourceClassName = "sourceClass";
const string sourceIndicatorName = "sourceIndicator";
const string specificProblemsName = "specificProblems";
const string stateChangeDefinitionName = "stateChangeDefinition";
const string stateChangesName = "stateChanges";
const string suspectObjectListName = "suspectObjectList";
```

```
        const string thresholdInfoName = "thresholdInfo";
        const string trendIndicationName = "trendIndication";
  }; // end of Notifications interface

}; // end of itut_x780 module
```

// MACROS

```
/* The following macros are provided for quickly and concisely defining
the notifications to be supported by an object. Example usage (within an
interface):
MANDATORY_NOTIFICATION(itut_x780::Notifications, objectCreation);
CONDITIONAL_NOTIFICATION(itut_x780::Notifications, stateChange, statePackage);
The macros simply expand into nothing, as CORBA IDL doesn't really have
anything for them to expand into that makes sense. Eventually, these
may be changed to expand into IDL supporting the CORBA Component Model.
*/
#undef MANDATORY_NOTIFICATION
#define MANDATORY_NOTIFICATION(InterfaceName, NotificationName)
#undef CONDITIONAL_NOTIFICATION
#define CONDITIONAL_NOTIFICATION(InterfaceName, NotificationName, PackageName)
#endif // end of ifndef itut_x780_IDL
```

```
/* This IDL code is intended to be stored in a file named "itut_x780Const.idl"
and located in the same directory as the file containing Annex A */
#ifndef ITUT_X780Const_IDL
#define ITUT_X780Const_IDL
#pragma prefix "itu.int"
module itut_x780 {
```

// 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 {
        const string moduleName = "itut_x780::ApplicationErrorConst";
        /** This application error exception code indicates the operation
          failed due to a problem downstream from the managed system,
          possibly a communication problem between the managed system
          and the resource */
        const short downstreamError = 1;
        /** An application error exception returining this code will return
          the name of the offending paramter in the details field. */
        const short invalidParameter = 2;
        /** This application error exception code indicates the operation
          failed due to a transient problem on the managed system. */
        const short resourceLimit = 3;
}; // end of module ApplicationErrorConst
```

// CreateErrorConst Module

```
/** This module contains the constants defined for the error code contained in
Create Error Info structures returned with Create Error exceptions.
*/
module CreateErrorConst {
        const string moduleName = "itut_x780::CreateErrorConst";

        /** This create error exception code indicates that the name included
          in the create operation is not valid. */
        const short badName = 1;
        /** This create error exception code indicates that the name included
          in the create operation is a duplicate. */
        const short duplicateName = 2;
        /** This create error exception code indicates some packages requested
          in the create operation are incompatible with each other. It must
          be included in a PackageErrorInfoType structure (subclass of
          CreateErrorInfoType). The packages list contains the names of the
          unsupported packages. */
        const short incompatiblePackages = 3;
        /** This create error exception code indicates that the name binding
          referenced in the create operation is not valid. */
        const short invalidNameBinding = 4;
        /** This create error exception code indicates a package requested in
          the create operation is not supported. It must be included in a
          PackageErrorInfoType structure (subclass of CreateErrorInfoType).
          The packages list contains the names of the unsupported packages.
          */
        const short unsupportedPackages = 5;
}; // end of module CreateErrorConst
```

// DeleteErrorConst Module

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

```
        const string moduleName = "itut_x780::DeleteErrorConst";
        /** This delete error exceptin code indicates the object has both
          subordinates and a delete policy of deleteOnlyIfNoContained. */
        const short containsObjects = 1;
        /** This delete error exception code indicates the object has a delete
          policy of notDeletable, and cannot be deleted. */
        const short notDeletable = 2;
        /** This delete error exception code indicates the object had a
          subordinate object that could not be deleted, so the superior
          object(s) could not be deleted. */
        const short undeletableContainedObject = 3;
        /** This delete error exception code indicates the object is in
          a state in which it cannot be deleted. */
        const short invalidStateForDestroy = 4;
}; // end of module DeleteErrorConst
```

// ProbableCauseConst Module

```
/** This module contains the constant values defined for the
ProbableCause UID. These values were borrowed from X.721. */
module ProbableCauseConst {
const string moduleName = "itut_x780::ProbableCauseConst";
        const short indeterminate = 0;
        const short adapterError = 1;
        const short applicationSubsystemFailure = 2;
        const short bandwidthReduced = 3;
        const short callEstablishmentError = 4;
        const short communicationsProtocolError = 5;
        const short communicationsSubsystemFailure = 6;
        const short configurationOrCustomizationError = 7;
        const short congestion = 8;
        const short corruptData = 9;
        const short cpuCyclesLimitExceeded = 10;
        const short dataSetOrModemError = 11;
        const short degradedSignal = 12;
        const short dTE_DCEInterfaceError = 13;
        const short enclosureDoorOpen = 14;
        const short equipmentMalfunction = 15;
        const short excessiveVibration = 16;
        const short fileError = 17;
        const short fireDetected = 18;
        const short floodDetected = 19;
        const short framingError = 20;
        const short heatingOrVentilationOrCoolingSystemProblem = 21;
        const short humidityUnacceptable = 22;
        const short inputOutputDeviceError = 23;
        const short inputDeviceError = 24;
        const short lANError = 25;      const short leakDetected = 26;
        const short localNodeTransmissionError = 27;
        const short lossOfFrame = 28;
        const short lossOfSignal = 29;
        const short materialSupplyExhausted = 30;
        const short multiplexerProblem = 31;
        const short outOfMemory = 32;
        const short ouputDeviceError = 33;
        const short performanceDegraded = 34;
        const short powerProblem = 35;
        const short pressureUnacceptable = 36;
        const short processorProblem = 37;
        const short pumpFailure = 38;
        const short queueSizeExceeded = 39;
        const short receiveFailure = 40;
        const short receiverFailure = 41;
        const short remoteNodeTransmissionError = 42;
        const short resourceAtOrNearingCapacity = 43;
        const short responseTimeExcessive = 44;
        const short retransmissionRateExcessive = 45;
        const short softwareError = 46;
        const short softwareProgramAbnormallyTerminated = 47;
        const short softwareProgramError = 48;
        const short storageCapacityProblem = 49;
```

```
        const short temperatureUnacceptable = 50;
        const short thresholdCrossed = 51;
        const short timingProblem = 52;
        const short toxicLeakDetected = 53;
        const short transmitFailure = 54;
        const short transmitterFailure = 55;
        const short underlyingResourceUnavailable = 56;
        const short versionMismatch = 57;
}; // end of ProbableCauseConst module

// SecurityAlarmCauseConst Module
/** This module contains the constant values defined for the
SecurityAlarmCause UID. These values were borrowed from
X.721. */
module SecurityAlarmCauseConst {
const string moduleName = "itut_x780::SecurityAlarmCauseConst";
        const short authenticationFailure = 1;
        const short breachOfConfidentiality = 2;
        const short cableTamper = 3;
        const short delayedInformation = 4;
        const short denialOfService = 5;
        const short duplicateInformation = 6;
        const short informationMissing = 7;
        const short informationModificationDetected = 8;
        const short informationOutOfSequence = 9;
        const short intrusionDetection = 10;
        const short keyExpired = 11;
        const short nonRepudiationFailure = 12;
        const short outOfHoursActivity = 13;
        const short outOfService = 14;
        const short proceduralError = 15;
        const short unauthorizedAccessAttempt = 16;
        const short unexpectedInformation = 17;
        const short unspecifiedReason = 18;
}; // end of SecurityAlarmCauseConst module

}; // end of itut_x780 module
#endif // end of ifndef ITUT_X780Const_IDL
```

付録 I Bibliography

The following Recommendations and other references contain information that was used in the development of these guidelines. As stated in the introduction, a primary design goal of these guidelines is to enable the reuse of existing network management information models, at least without significant semantic changes. These documents provide many of the details on the ITU-T's CMIP framework, and therefore define some of the functionality the CORBA object modelling guidelines must support.

[8] ITU-T X.720 (1992) | ISO/IEC 10165-1:1993, *Information technology – Open Systems Interconnection – Structure of management information: management information model*.

[9] ITU-T X.733 (1992) | ISO/IEC 10164-4:1992, *Information technology – Open Systems Interconnection – Systems Management: Alarm reporting function*.

[10] ITU-T M.3010 (2000), *Principles for a telecommunications management network*.

[11] ITU-T M.3120 (2001), *CORBA generic network and NE level information model*.

[12] ITU-T Q.821 (2000), *Stage 2 and Stage 3 description for the Q3 interface – Alarm surveillance*.