TTC標準
Standard

ＪＴ－Ｘ７８０．１

粗粒度 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 coarse-grained CORBA managed object interfaces

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

目　次

＜参考＞

１．国際勧告等との関連

本標準は、ITU-T 勧告 X.780.1 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：X.780(2001), Q.816(2001), Q.816.1(2001)

（2）本標準の作成

網管理専門委員会

１．範囲

ITU-T M.3010 (2000) で定義されている TMN アーキテクチャは、分散処理からの概念を導入し、複数の管理プロトコルの使用を含む。ITU-T Q.816 および X.780 は、TMN 管理プロトコルのうちの１つとして共通オブジェクト・リクエスト・ブローカ・アーキテクチャ(CORBA)を適用するためにこのアーキテクチャ内にフレームワークを定義する。

本標準は、ITU-T Q.816.1 と共に、独自のフレームワーク仕様書で指定されたものよりむしろ、管理システムと被管理システム間の僅かに異なる相互作用形式のサポートを可能とするために、フレームワークに仕様書を加える。この相互作用スタイルはある利点を持っている。その主要なものとしては、個々のアクセスを望む扱いやすい資源のために管理システムがオブジェクト指向ソフトウェアアドレスを検索する必要性を解除する。

これらのソフトウェアアドレスは、大規模システム上で何百万もの番号を割り当てられている可能性がある。更に、その相互作用スタイルは、被管理システム上で組み立てられる、被管理システム供給者が好むような方法を幾らか変更する。

本標準の範囲は、オリジナルの TMN CORBA フレームワークと同じである。そのフレームワーク及び拡張内容は、CORBA が使用し得るあらゆる TMN 内のインタフェースをカバーする。しかしながら、ここに定義された全ての能力及びサービスが全ての TMN インタフェースで要求されるとは限らない。この事は、管理システムとネットワークエレメント間と同様に、抽象的概念（外部と内部管理者）の全てのレベルにおける管理システム間のインタフェースのために、フレームワークが利用され得る事を暗示している。

1.1 目的

本標準の目的は、より広い範囲のアプリケーションで利用できるように、TMN CORBA フレームワークを拡張することである。その拡張は、管理システムと被管理システムの間の僅かに異なる相互作用モードを多くの場合に好ましくなり得るようにできる。従って、本標準は様々なグループによってネットワーク管理インタフェースを指定するために利用される事を意図している。

1.2 アプリケーション

CORBA TMN フレームワーク勧告で取られたアプローチは、管理可能なネットワーク資源を、CORBA を利用してアクセス可能なソフトウェアオブジェクトとしてモデル化する事である。CORBA インタフェース定義言語（IDL）で書かれた情報モデルは、オブジェクトインタフェースを記述する。

CORBA は、位置に関わらず、あるソフトウェアオブジェクトが他のソフトウェアオブジェクトと対話できるようにするためにロケーション非依存性を提供する。ソフトウェアオブジェクトは相互作用可能なオブジェクトリファレンス（IOR）として CORBA が参照するものを利用するのにアクセスされる。

オリジナルの CORBA TMN フレームワークは、それらのユニークな IOR と共に、各々の管理可能な資源を独立した CORBA オブジェクトとしてモデル化する。このアプローチは、各オブジェクトがどのような場所にある事をも柔軟に可能にする。しかしながら、そのフレームワークは、管理システムが、手元にそれらがアクセスしたいオブジェクトごとの IOR を持っていることを必要とする。これは、電気通信産業の多くの企業や管理者が回避するように努力してきた負担である。更に、そのフレームワークは、膨大な数の IOR（幾つかの被管理システムの供給者が回避したがる）をサポートする事を、被管理システムに要求する事も出来た。本標準は ITU-T Q.816.1 に加え、膨大な数の IOR を必要とする事を回避するために、TMN CORBA フレームワークがどのようにして拡張されることになっているか定義する。

個々の管理可能な資源がユニークな IOR でアドレス付け可能なアプローチを利用する、CORBA に基づいたインタフェースは細粒度インタフェースとして知られるようになってきた。代わりに、IOR が個々の管理

可能な資源に割り当てられない場合のインタフェースは粗粒度インタフェースとして知られている。

本標準は、粗粒度インタフェース上で管理可能な資源をモデル化するために、僅かに異なるアプローチを定義するので、インタフェースモデル仕様書は、細粒度アプローチと粗粒度アプローチとの違いを僅かなものにするだろう。

1.3 標準のロードマップ

本標準は以下の様に構成されている。

１節	導入、ロードマップとアップデート
２節	リファレンス
３節, ４節	本標準全体にわたって使用される定義および略語
５節	協定
６節	粗粒度インタフェースのサポートとして割り付けられなければならない設計上の考慮がフレームワークに付与される。
７節	TMN CORBA フレームワークと粗粒度必要条件概観
８節	管理オブジェクトにアクセスするためのファサードインタフェースの提供。この節は、粗粒度インタフェースで実装されるに違いない、モデル特有のインタフェースをカバーする。
９節	粗粒度 CORBA インタフェースを定義するガイドライン
10 節	細粒度 CORBA インタフェース仕様を粗粒度インタフェース仕様に変換するためのガイドライン
11 節	遵守と適合のガイドライン
付属資料 A	粗粒度でモデル化するガイドライン仕様のための IDL モジュール。この付属資料は標準である。

２．標準リファレンス

以下の ITU-T 勧告及び他のリファレンスは、このテキスト内のリファレンスを通して、本標準の条件を構成している条件を含んでいる。出版時に示された版数は有効だった。全ての勧告及び他のリファレンスは修正に従う。従って、本標準の利用者は、下にリストされた勧告および他のリファレンスの中の最新版の適用可能性を調査する事が推奨される。現在有効な ITU-T 勧告のリストは定期的に出版される。

[1] ITU-T X.780 (2001), *TMN guidelines for defining CORBA managed objects.*

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

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

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

３．定義

3.1 ITU-T X.701 からの定義

本標準で使用される以下の専門用語は、システム管理概観で定義される。（ITU-T X.701）

－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専門用語は、開放型分散管理アーキテクチャで定義される。（ITU-T X.703）－通知

3.3 追加定義

3.3.1ファサード：全てが同じクラスにあるひと組の管理オブジェクトへのアクセスを提供するために定義されるオブジェクトインタフェース。

４．略語

本標準は以下の略語を使用する。

CMIP 共通管理情報プロトコル
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
COS共通オブジェクトサービス
DN 識別名
EMS エレメント管理システム
GDMO 管理オブジェクトの定義のためのガイドライン
ID 識別子
IDL インタフェース定義言語
IIOPインターネット相互運用プロトコル
IOR 相互運用可能なオブジェクトリファレンス
ITU-T 国際電気通信連合－電気通信標準化セクタ
MO 管理オブジェクト
NE ネットワークエレメント
NMS ネットワーク管理システム
OAM&P 操作，運用，保守，供給
OID オブジェクト識別子
OMG オブジェクト管理グループ
ORB オブジェクトリクエストブローカ
OSI 開放型システム間相互接続
PDUプロトコルデータユニット
POAポータブルオブジェクトアダプタ
QoS サービス品質
RDN 相対識別名
TMN 電気通信管理網
UID 国際識別子
UML 統一型モデル化言語
UTC協定世界標準時

５．協定

5.1　協定

幾つかの協定は、読者にテキストの目的を知らしめるために、本標準の中で後述される。本標準の殆どが標準である間、（管理であれ、被管理であれ）管理システムによって満たされる必須要求を簡潔に述べるパラグラフの前に、括弧で囲まれた太字”R“が先行し、要求の主題を示す短い名称および数が続く。例えば、

(R)EXAMPLE-1　例　必須要求。

管理システムによってオプションとして実装されるかもしれない要求は、「R」の代わりに「O」が先行している。例えば、

(O)OPTION-1　例　オプション要求。

要求状態は、遵守と適合のプロフィールを作成するために使用される。
CORBA IDL の多くの例は、本標準に含まれている。また TMN 特有のサービスとサポートするデータタイプを規定する IDL は、付属資料 A に含まれている。IDL は 9 ポイントクーリエ書体で書かれる。

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

5.2　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 を利用する利点は、ORB に付随したツールによって、IDL がプログラムコードに“コンパイル”できる事にある。これは、実際に、ネットワーク管理アプリケーションが共同で作業することを可能にするために必要な、幾つかのコードの開発を自動化する。付属資料 A は実装者が抽出し、コンパイルしたいと思うであろうコードを含んでいる。付属資料 A は標準化され、本標準に従ってシステムを実装する開発者によって利用されるべきである。本標準内の IDL はその正確さを保証するために、二つのコンパイラによってチェックされている。ITU-T Q.816 で指定されているバージョンの CORBA をサポートするコンパイラが使われなければならない。

付属資料 A はコンパイルされるようなプレーンテキストファイルにカット＆ペーストできるように単純なフォーマットで作成されている。以下は、その方法である。

1)カット＆ペーストは、本標準の Microsoft Word 版から行う事でよりよく実行できるようである。Adobe Acrobat のファイルからのカット＆ペーストはページのヘッダやフッターを含んでしまうため、コンパイルする事が出来ない。

2)付属資料 A は全て、“ /*　この IDL コードは…”という行から始まり最後まで、IDL コンパイラによって見つかるディレクトリの中の“ itut_x780_1.idl ”という名称のファイルに入れられるべきである。

3)付属資料 A の中の埋め込まれた標題は、削除する必要はない。それらの標題は、IDL コメントの中に入れられており、コンパイラに無視される。

4)特別なシーケンス“ /** ”で始まるコメントはコンパイラによって判別され、IDL から HTML に変換される。これらのコメントはしばしば、コンパイラにとって特別のフォーマット指示を行っている。IDL と共に働くであろうそれらのコメントは、ファイル間の早急なナビゲーションを行うためのリンクを保持する HTML ファイルの生成として、HTML を生成する事を必要とするかもしれない。

5)付属資料 A は Tab の空白を８空白間隔として作成されている。またほとんどどんなテキストエディターも動作できる Hard line feeds で作成されている。

６．粗粒度のインタフェースデザインの考察

この節は、粗粒度インタフェースのサポートが加えられるようにフレームワークによって割り当てられなければならない、幾つかの設計考察を識別する。

6.1 粗粒度のオブジェクトの生成と削除

管理資源の粗粒度表現は生成、削除出来なければならない。粗粒度インタフェース上の生成操作を含む可能性は調査されるべきである。

6.2 属性

フレームワークは粗粒度インタフェースを通して、アクセスされる管理資源を関連属性としてサポートしなければならない。

6.3 通知

フレームワークは、粗粒度インタフェースを通してアクセスされる管理資源からの事象通知をサポートしなければならない。

6.4 全ての管理資源への粗粒度のアクセス

フレームワークは、粗粒度インタフェースを通して、全ての管理資源へのアクセスする事を管理システムに許可するための実装を要求し、また、可能でなければならない。

6.5 例外

粗粒度インタフェースは、操作の発動において管理資源が例外の発行を可能としなければならない。これらの例外は、各操作のために明示的に明確にされなければならない。

6.6 全ての操作のサポート

粗粒度インタフェースは管理資源に適用可能な全ての操作をサポートしなければならない。

6.7 規定マッピング

細粒度情報モデルと粗粒度情報モデル間のマッピングが規定されなければならない。そのマッピングは、アルゴリズム的に実行することができなければならない。もしも、IDL 情報モデルが GDMO 情報モデルの翻訳により開発されるならば、翻訳中に手作業で実行されたどんな最適化も、細粒度モデルと粗粒度モデルの双方に現われなければならない。

6.8 複数のオブジェクトからの属性検索

単一の強く型付けされた操作の中で、同じ種別の複数の管理オブジェクトのインスタンスの属性を検索する必要がある。同じファサードを使用して、これらのオブジェクトがすべてアクセスされるので、この検索は、ファサードによって実行された単一の操作で遂行することができる。

７．フレームワークと要件概観

6 節では、フレームワークに追加され粗粒度インタフェースをサポートするために、解決されるべき設計考察を概説した。7 節および本標準の残りの部分では、これらの問題に取り組むためにフレームワークがど

のように拡張されるかについての詳細を提供する。本標準が、粗粒度インタフェースのための情報モデルの開発のためにガイドラインを定義している一方、ITU-T Q.816.1 では、粗粒度インタフェースのためのフレームワークサポートサービスに注目する。最初に、現在のフレームワークの簡潔な概観を示し、次いでその拡張の概観を示す。

7.1 フレームワーク概観

CORBA ベースの TMN インタフェース用のフレームワークは、能力の集合体である。フレームワークの中央の 1 片は、1 セットの OMG 共通オブジェクトサービスである。フレームワークは、ネットワーク管理インタフェース内での OMG 共通オブジェクトサービスの役割を定義し、それらのサービスの使用のための協定を定義する。フレームワークはさらに、OMG 共通オブジェクトサービスとして標準化されていないが、フレームワークに一致するネットワーク管理インタフェースで標準であると期待されるサポートサービスを定義する。

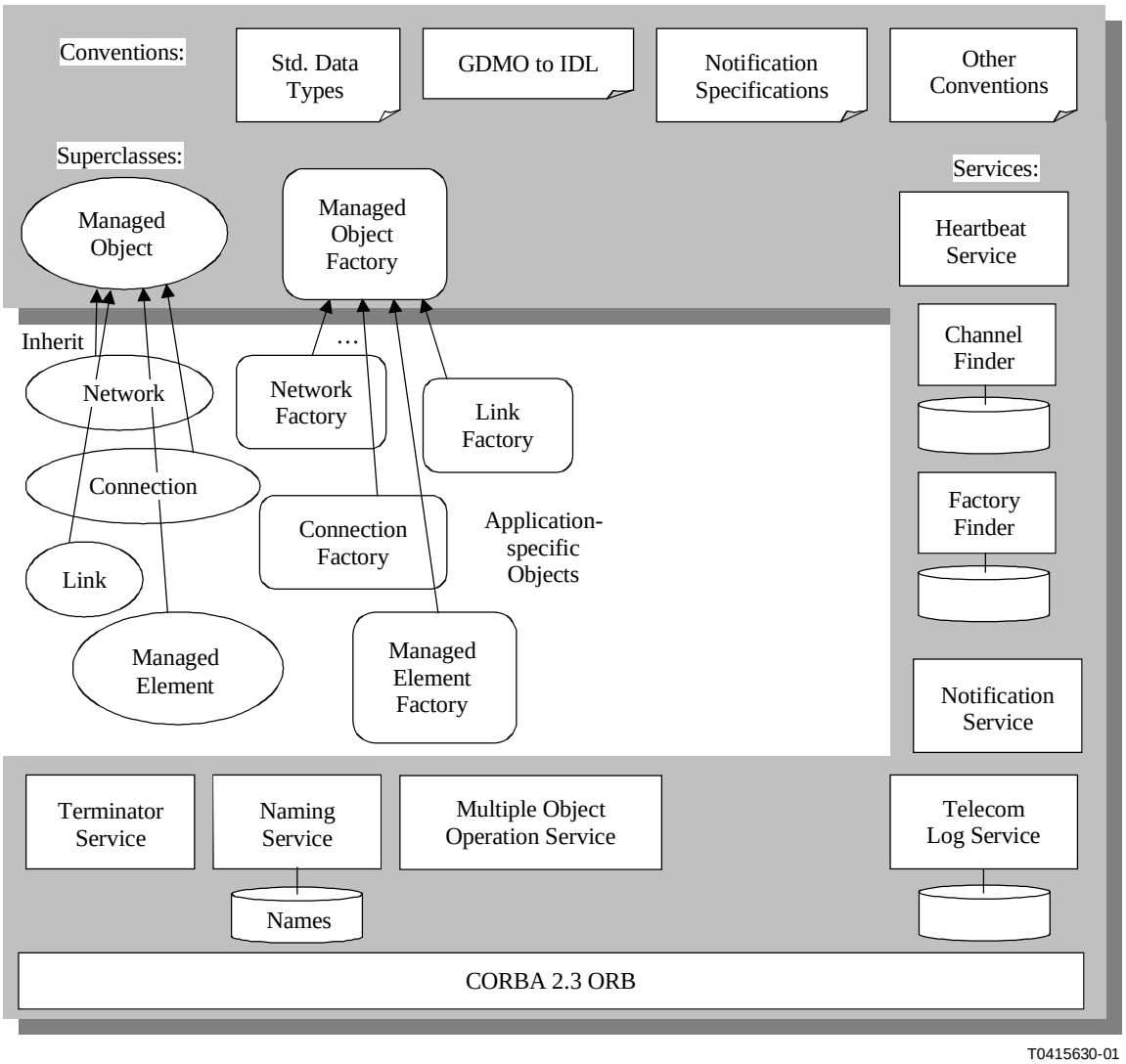

図 1/JT-X780.1　フレームワーク概観

(ITU-T X.780.1)

フレームワークは、図 1 に図示される。図中、灰色でフレームワークを示す。中位には、フレームワークにサポートされるアプリケーション固有のオブジェクトがある。底部に沿って、CORBA ORB を表わす箱がある。その上に、フレームワークを構成するサービスを表わす名前を備えた無数の箱がある。(いくつかはさ

らに、機能を実行するために保守しなければならないデータベースを描くアイコンを持っている。)これらのサービスは ORB バージョンの要求条件と共に、ITU-T Q.816 に定義される。図の上部に沿って、2 つの上位クラスがアイコンで示される。一つは管理対象オブジェクト、もう一つは管理対象オブジェクトファクトリである。このフレームワークにサポートされた管理対象オブジェクトおよび管理対象オブジェクトファクトリは、結局それぞれ継承するに違いない。さらに図の上端に示されたアイコンは、協定をモデル化する標準のオブジェクトを表わす。これらの協定および上位クラスは ITU-T X.780 に定義される。

7.2 粗粒度の拡張概観

7.2 節では、粗粒度インタフェースをサポートするのに必要なフレームワークの拡張の概観を提供する。

7.2.1 ファサードデザインパターン(The facade design pattern)

粗粒度インタフェースをサポートするために必要とされるフレームワークへの最も重要な変更は、管理オブジェクトがアクセスされる方法である。システムによりサポートされる IOR 数が増えない一方で、被管理システム上の管理オブジェクト数は増やせるに違いない。管理オブジェクトへのアクセスは強い型付けが残っているとしても、望ましいことである。これは、”facade(ファサード)”パターンとしてここに参照されるデザインパターンの利用に結びつく。ファサードは見せかけの正面または表玄関と見なすことができる。ファサードデザインパターンの利用により、被管理システムは、システム上の管理オブジェクトの各タイプにつき少なくとも 1 つから通常高々数個の少数のファサードインタフェースをサポートするであろう。管理システムは、実際にはシステム上の管理オブジェクト種別のためのファサード上の操作を呼び出すことにより、管理オブジェクトに対する操作を呼び出すであろう。ファサードデザインパターンでは、管理オブジェクトが CORBA インタフェースを露出する必要は無いし、従って個々の IOR を持たないかもしれない。これは、ファサードアプローチをサポートする被管理システムが細粒度管理オブジェクトインタフェースを実装する必要が無いことを意味している。

ファサードを、管理オブジェクトとしてではなく、管理するシステムが管理オブジェクトにアクセスすることができる中間オブジェクトと見なすことが最も良い。ファサードオブジェクトは CORBA インタフェースを持っており、CORBA を用いてアクセスできる。管理オブジェクトはしかしながら、CORBA インタフェースを持っていないかもしれないし、CORBA を用いて直接アクセスできないかもしれない。ファサード自身は管理可能なネットワーク資源を表現しない。その目的は、管理可能な資源を表わすオブジェクトを用いて相互接続を可能にすることである。全てのファサードオブジェクトは被管理システムにより自動的に生成され、管理オブジェクトがファサードを通じてアクセス可能な間存在する。同種の管理オブジェクト用の複数のファサードが、粗粒度インタフェース上に存在し得る。しかし管理オブジェクトは 1 つのファサードのみを通してアクセスできるものとしなければならない(図 2 参照)。

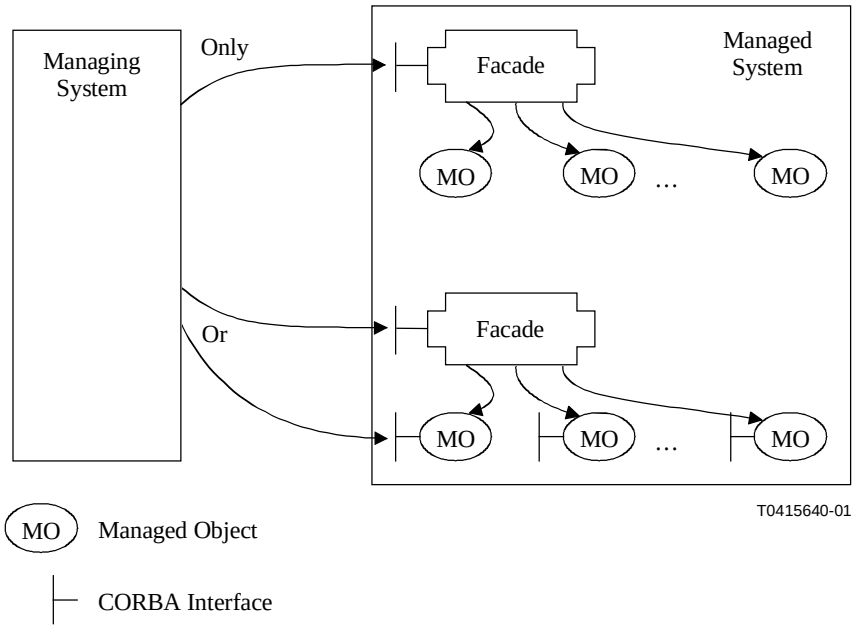

図 2/JT-X780.1　ファサードロール

(ITU-T X.780.1)

図 2 は、粗粒度アプローチをサポートする被管理システムにアクセス中の管理システムを示している。被管理システムは、管理システムが管理オブジェクトの 2 つの異なるセットにアクセスできる 2 つのファサードインタフェースを持っている。図の上側の管理オブジェクトは、ファサードを通してのみアクセスされうる。図の下側の管理オブジェクトはまた、直接 CORBA インタフェースをサポートし、ファサード経由でまたは直接アクセスされうる。直接の CORBA アクセスはオプションであるが、ファサードアプローチをサポートする被管理システムは、管理オブジェクトインスタンスそれぞれのためのファサードインタフェースを提供しなければならない。

ファサードは、管理オブジェクトの操作を呼び出す CORBA インタフェース、あるいは他のある実装に特有な手段を使用し得る。被管理システムは、実際、個々のオブジェクトの内部として管理オブジェクトをさらに実装する必要はない。しかしながら、このフレームワークに基づくインタフェースの実装により、管理オブジェクトがオブジェクトとして内部に実装されるような錯覚を与えるかもしれない。

ファサードを通じて管理オブジェクト上で操作が呼び出されたとき、ファサードは実際の管理オブジェクトまたはエンティティの操作を呼び出さなければならない。なぜなら数ある管理オブジェクトが単一のファサードを通じてアクセスされ、ファサードはどの管理オブジェクトが実際の操作対象であるか知っていなければならないからである。これは、管理オブジェクト向けの全ファサード操作の第一パラメータとして対象となる管理オブジェクトの名前を含むという申し合わせの採用により扱われるであろう。

管理オブジェクトはもはやユニークな IOR を持たないかもしれない一方、それらはまだユニークな名前を持ち、管理可能な資源を表す個々のエンティティとみなすこともできる。

7.2.2　管理オブジェクト名の拡張

上述されるように、たとえ管理オブジェクトが個々の CORBA インタフェースを持たないことがあっても、ファサードを通じてアクセスされた管理オブジェクトはまだ名前を持つであろう。管理オブジェクトの名前に基づいて、どのファサードを使用すべきかを管理システムが決定できることは重要である。もしこれができない場合、被管理システムを尋ねなければならないか、あるいは全ての管理オブジェクト名にファサード IOR を関連付けなければならない。名前だけに基づいて管理オブジェクトのファサードを決定する能力をサポートするために、ファサードを通じてアクセス可能な管理オブジェクトの名前は、ファサードを通じてア

クセスできない管理オブジェクトの名前よりもわずかに拡張される。末尾に”Object”(または ITU-TQ.816.1 による拡張で<empty>) を備えた *ID* 文字列を常に持つ名前コンポーネントでは、オブジェクトをアクセスし得るファサードに割り当てられたファサード識別子の値に *kind* 文字列が設定される。ファサードを通じてアクセスできない管理オブジェクトについては、この *kind* 文字列が空である。ITU-T Q.816.1 は、ファサードを識別するためにどのように最終的な管理オブジェクト名コンポーネントに *kind* 文字列が使用されるかについて、追加の詳細を提供している。

7.2.3 ファサードアクセス可能な管理オブジェクトのためのサポートサービス

ファサードアプローチを使用するインタフェース上で提供されるフレームワークサポートサービスは、大部分は ITU-TQ.816 に定義されたものと同じになる。Factory Finder や Channel Finder のようないくつかのサービスはまったく変更を要求しない。他の Terminator や Multiple Object Operation(MOO)のようなサービスは、それらのインタフェースの変更や管理システムによって使用される方法への変更を要求しないが、(ある実装に特有な手段よりもむしろ)管理オブジェクトのファサードインタフェースを使用してそれらのサービスが管理オブジェクトにアクセスする場合、それらの実装へ少々の変更を要求し得る。ITU-T Q.816.1 は、粗粒度インタフェースをサポートするために要求されるフレームワークサポートサービスの変更の詳細を提供している。

サポートサービスへの最大の変更は、ネーミングのサポート範囲に起因する。ファサードインタフェースは、サポートサービスインタフェースがそうであるのとほとんど同じ方法で、ネーミングサービス中の名前に結び付けられる。しかしながらファサードを用いるインタフェース上で、管理オブジェクトの名前は、OMG Naming Service の IOR に結び付けられることを要求されない。その代わり、新規のサービスが、管理オブジェクト名および包含関係情報を格納する場所として導入される。この新しいサービス(Containment Service)は、ITU-T Q.816.1 に定義される。

7.2.4 ファサードモデリング

ファサードデザインパターンとこのフレームワークで使用可能なファサードの定義をサポートするために、新規の基底インタフェースが導入される。このインタフェースは、Managed Object ファサードインタフェースとして知られるであろう。Managed Object インタフェースが細粒度インタフェースで行うように、それは、粗粒度インタフェースで同じ役割を果たす。すなわちそれは、全ての管理オブジェクトファサードインタフェースがフレームワークで働くために直接的あるいは間接的に継承しなければならない、基本インタフェースである。Managed Object Facade インタフェースは、ITU-T X.780 で定義される Managed Object インタフェースに極めて似ている。Managed Object ファサードインタフェースの定義は 8 節を参照のこと。

フレームワークへの変更は、図 3 に反映される。新規の上位クラスである *ManagedObjectFacade* が、図に追加されている。また、Containment Service が追加されている。それが管理オブジェクト名のデータベースへのアクセスを提供することに注意すること。Naming Service により保守される管理オブジェクト名のデータベースは、点線で示され、それは名前や管理オブジェクトの IOR を保持する必要が無いことを示している。しかしながら Naming Service は、管理システムがファサードインタフェースの検索やサービスの参照のサポートを可能とするとするために依然要求される。最終的には、管理オブジェクトも点線で引かれ、それらが直接アクセス可能である必要が無いことを示される。

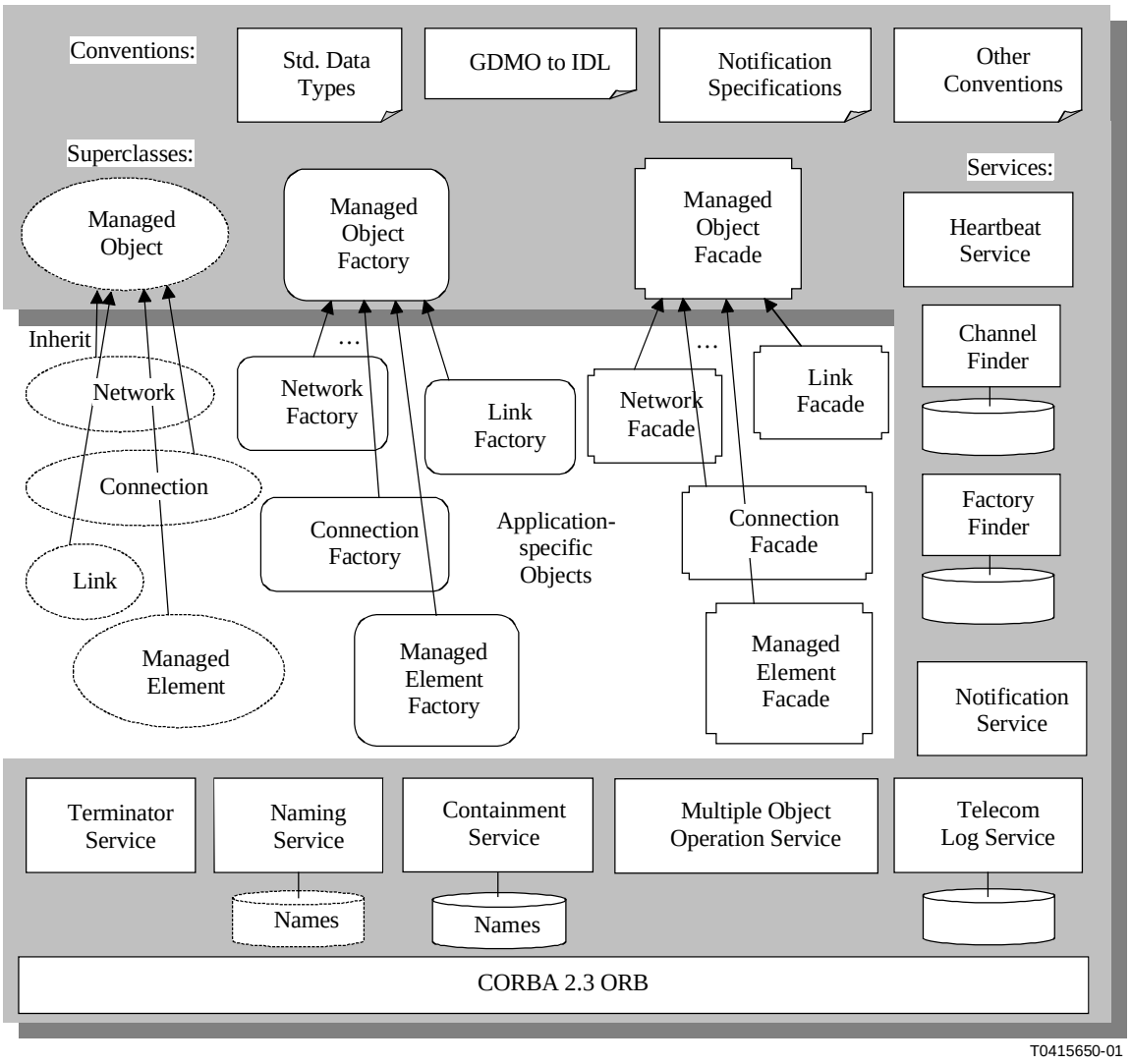

図 3/JT-X780.1　粗粒度インタフェースをサポートするために拡張されたフレームワーク
(ITU-T X.780.1)

８．管理オブジェクトのアクセスのためのファサード・インタフェースの提供

上記のように、ファサード・インタフェースは管理オブジェクトにアクセスするための異なる方法を提供する。粗粒度インタフェース(ファサード･インタフェースが存在する場合の１つ)においては、個々のCORBAインタフェースを使用して、管理オブジェクトがアクセスしてもよいし、アクセスしなくてもよい。したがって、管理するシステムは、ファサードに対する操作の起動することにより管理オブジェクトにアクセスするためにファサードを使用してもよい。管理システムが個々のオブジェクトとして管理オブジェクトをインプリメントしなくても、この勧告をサポートするによって、それは同様の作用となる。

8.1　ファサードのインスタンス化

この節は、管理オブジェクトにアクセスするためにファサード・インタフェースを提供する場合、管理システムが従わなければならない必要条件を定義する。

(R) FACADE-1 これらのオブジェクトがさらに直接の CORBA インタフェースをサポートしたとしても、管理システムは、それに関してインスタンス化されるかもしれない管理オブジェクトの各クラスに対して少なくとも 1 つのファサード・インタフェースを提供しなければならない。システム上でインスタンス化することができない管理オブジェクトクラスのためのファサード・インタフェースを提供する必要はない。これは

上位クラスを含んでいる。したがって、サブクラスの管理オブジェクトがインスタンスを作成されていなくても、そのタイプの管理オブジェクトがインスタンス化されていない場合、管理オブジェクトの上位クラス用ファサードのインスタンス化をする必要はない。しかし、これは上位クラスインタフェースを定義する必要がないことを意味しているのではない。ファサード・インタフェースの定義は管理オブジェクトと同じ継承階層に従う。下記の 9 節および 10 節を参照。ファサード・インタフェースに対するネームバインディングの必要条件は、ITU-T Q.816.1 で定義されている。

システム上でインスタンス化可能でない上位クラスに対するファサード・インタフェースを供給することから管理システムを取り除くことは、管理するシステムが多様性を利用することを排除しない。管理システムは、まだ、どんな CORBA クライアントも上位クラスとしてオブジェクトを扱うかもしれないのと同じ方法で上位クラスを通して利用可能なファサードとアクセスの能力の上位クラスとしてファサード・インタフェースを扱うかもしれない。しかしながら、操作は、そのファサードの IOR を使用して、サブクラスファサード上で現実に起動される。クライアント・システム上のプログラミング言語は多形を扱う。

(R) FACADE-2　管理システムは、管理オブジェクトの与えられたクラスに対し複数のファサード・インタフェースを供給してもよい。1 つのファサード・インタフェースを通してアクセス可能な管理オブジェクトは他のいかなるものによってもアクセス可能ではない。ITU-T Q.816.1 は、その名前に基づいた管理オブジェクトにアクセスするためにどのファサードを使用するべきであるか識別する方法についての詳細を提供する。

(R) FACADE-3　すべてのファサード・インタフェースは管理システムによって生成され削除される。ファサード・インタフェースは、それによってアクセス可能な管理オブジェクトのうちのどれでも存在する全ての時間に存在しなければならない。ファサードが作られるか削除される場合、通知されない。したがって、それらが新しいファサードを通ってアクセス可能であることを示す名前を備えた管理オブジェクトが現われ始める場合、管理するシステムは新しいファサードの存在を知るようになる。

(R) FACADE-4 特定の管理オブジェクトに向けられるそれぞれのファサード・操作は、操作の第 1 のパラメータにその管理オブジェクトの名前を含む。ファサードは、指定された管理オブジェクトに対するその操作を起動し、例外を含む結果を返さなければならない。指定された管理オブジェクトがそのファサードを通してアクセス可能でない場合、ファサードはその操作に対して applicationError 例外を上げなければならない。この例外におけるエラーコードは、付属資料 A の中の IDL に定義された objectNotFound コードにセットされなければならない。

8.2　ファサード・インタフェース基底クラス

この節は、付属資料 A.の中の IDL に定義される管理オブジェクトファサード基底クラスについて記述する。管理システム上で提供される管理オブジェクトファサード・インタフェースの全てはこのインタフェースから直接あるいは間接的のいずれかで継承されなければならない。このインタフェースは、すべてのファサード・インタフェースがこのフレームワークで使用するためサポートするよう、1 セットの基礎的な操作を提供する。

付属資料 A の IDL の中で"*ManagedObject_F*"と呼ばれる、管理オブジェクトファサードインタフェースが、細粒度インタフェース上のきめの細かいインタフェース上の *ManagedObject* インタフェースと同じ役割を果たす。したがって、それは、ITU-T X.780 に定義されている ManagedObject インタフェースに非常に似ている。*ManagedObject_F* インタフェースに関するほとんどの操作は、*ManagedObject* インタフェースに関する操作とほとんど同一である。新しい 1 つの操作である *attributesBulkGet* は加えられ、そして 1 つ、*nameGe*t

が落ちる。

8.2.1 管理オブジェクトファサード基礎的な能力

管理オブジェクトファサード・インタフェースがすべてサポートしなければならない能力は次のとおり:

- 指定された管理オブジェクトのクラス名を返す方法。
- 指定された管理オブジェクトにサポートされた条件付きのパッケージを返す方法。
- 指定された管理オブジェクト(それは、管理操作に応じて、管理資源によって自主的に作成されたか、それとも、未知だったか)の生成リソースを返す方法。
- 指定された管理オブジェクトに対して削除ポリシを返す方法。これは列挙型数で、それがオブジェクトを削除可能でないことを示し、オブジェクトを含んでいない場合のみそれが削除されるかどうかを示す、そうでなければ削除されたとき、含まれるすべてのオブジェクトが削除されることを示す。
- 指定されたもののための判読可能な属性をすべて含んでいる CORBA 値タイプ・オブジェクトを返す方法。
- 1 セットの管理オブジェクトのための判読可能な属性をすべて含んでいる CORBA 値タイプ・オブジェクトを返す方法。
- 破棄操作。

8.2.2 管理オブジェクトファサード IDL

ManagedObject_F インタフェース(コメントなし)を説明する IDL は:

```
ObjectClassType objectClassGet(in NameType name)
        raises (ApplicationError);
StringSetType packagesGet (in NameType name)
        raises (ApplicationError);
SourceIndicatorType creationSourceGet(in NameType name)
        raises (ApplicationError);
DeletePolicyType deletePolicyGet (in NameType name)
        raises (ApplicationError);
ManagedObjectValueType attributesGet (
        in NameType name,
        inout StringSetType attributeNames)
        raises (ApplicationError);
boolean attributesBulkGet (
        in NameSetType names,
        in StringSetType attributeNames,
        in unsigned short howMany,
        out AttributesGetResultSet attributes,
        out AttributesGetResultIterator iterator)
        raises (ApplicationError);
void destroy(in NameType name)
        raises (ApplicationError,
            DeleteError);
}; // end of ManagedObject_F interface
```

8.2.3 objectClassGet()操作

objectClassGet()操作は、指定された管理オブジェクトの検索されたインタフェース名を返す。指定された管理オブジェクトは”name”と命名される第1のパラメータに含まれた名前を備えた管理オブジェクトである。ファサードのクラス名とはこの値が異なることに注意。それがさらに、粗粒度インタフェース上の管理オブジェクトには直接の CORBA インタフェースをサポートすることができるため、もしそれが管理オブジェクト上で等価な操作を直接起動すれば管理するシステムが得るのと同じ値を返すことをこの操作に要求するために決定された。細粒度管理オブジェクトインタフェースが、ファサードを通してアクセス可能な非管理オブジェクトのクラスに対して定義されていない場合、レスポンスはトレール”_F”のないファサードの検索されたインタフェース名でなければならない。この操作の応答で返された同じ値は、オブジェクトからの通知、およびオブジェクトのために返された値タイプに含まれるだろう。リターン・タイプである *ObjectClassType* はストリング用のタイプ定義である。

ファサードへの言及を与えられると、管理するシステムは標準の CORBA 呼び出し(例えばインタフェース CORBA::Object の上の get_interface 呼び出し)を通じてそれがどんなタイプのファサードかを決定することができる。したがって、このための個別の操作は基礎ファサード・インタフェース上で定義されない。

8.2.4 packagesGet()操作

packagesGet()操作は、指定された管理オブジェクトにサポートされた条件付きのパッケージのリストを返す。それは、同じファサードを通してアクセスされた管理オブジェクトが異なる条件付きのパッケージをサポートすることが可能である。条件付きのパッケージについての概念、ストリングの名前で各々、ITU-T X.780 に定義される。*StringSetType* はストリングのシーケンス用のタイプ定義である。

8.2.5 creationSourceGet()操作

creationSourceGet()操作は、指定された管理オブジェクトが作成されたシステムを示す値を返す。*SourceIndicatorType* は3つの値を備えた列挙型である:*resourceOperation*、*managementOperation*、そして未知。それは、管理操作に応じて、資源によってオブジェクトが自主的に作成されたか、なぜオブジェクトが作成されたかが知られていないかどうか示す。

8.2.6 deletePolicyGet()操作

deletePolicyGet()操作は指定された管理オブジェクトのために削除ポリシを返す。これはそれがオブジェクトを含んでいない場合のみ あるいはそれが削除される時含まれているオブジェクトがすべて削除されれば、それが削除することができるということである場合、オブジェクトが削除することができるということでないかどうか示す、列挙型の値である。(オブジェクトの削除（しかしその含まれているオブジェクトではない）は許可されない。)オブジェクトが、生成操作で識別された情報を結び付けている名前に基づいたそのファクトリによって作成される場合、このポリシーがセットされる。

8.2.7 attributesGet()操作

attributesGet()方法は、1 つの操作におけるオブジェクトの属性値のすべて、あるいはすべてのサブセット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またはファサード・インタフェースに対して、そのインタフェース上の読込可能な属性の各々に対するデータ・メンバーを含んでいる CORBA 値型が定義される。(判読可能な属性は <属性名>Get ()操作)この方法は任意の管理オブジェクトに対するこの値型を取り出すために使用することができる。値型は、管理オブジェクトインタフェースの継承階層に続いて、定義される(値型が多重継承をサポートすることができないことを除いて)。また、各々は結局 ITU-T X.780 に定義された *ManagedObjectValueType* から導かれる。管理オブジェクトは、この方法に応じてそのインタフェー

スに対して定義されたサブクラス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*()操作を起動する時、そのクライアントは、操作が起動されたインタフェースのために定義された値型に制限する (キャスト)*ManagedObjectValueType* への言及を受け取る。

これを多少複雑にすることはクライアントがあるインスタンスから属性値のすべてを検索したくないそして、条件付きのパッケージ中である属性のすべてをサポートしてもよいとは限らないという懸念である。(値型は条件付きのパッケージに属性を含んでいる)これは in/out の *attributeNames* パラメータの使用を通じて提供される。実施においては、クライアントが、サポートされた属性のすべてが返されるべきである特別な意味を持つ空のリストと共にそれが関連する属性の名前のリストを提出してもよい。リストに載っている有効な属性名でないいかなる名前も管理オブジェクトによって無視されるべきである。そのレスポンスで、オブジェクトは、値が供給される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かもしれないことに注意すること。提出されたリストが空だったか、無効の名前があったとしても、オブジェクトが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、オブジェクトは無効のリストと空の値型を返すべきである。

値型の構造があらかじめ定められるので、オブジェクトは要求されない、サポートされていない属性に対する値を埋めなければならない。基本的に、オブジェクトは、これらの属性に対するどんな値も返してもよいが、しかし、値は効率のためにできるだけ短くあるべきである。したがって、どんな種類のストリング、参照およびリストに対しても無効の値は返されるべきです。整数および列挙型に対して、どんな値も返してもよい。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名前がない属性が null であるいかなる値に対しても考慮しなければならない。

8.2.8 attributesBulkGet()操作

attributesBulkGet()方法は、ファサードを通してアクセス可能である同じタイプの多数の管理オブジェクトから多数の属性を返すために使用される。管理するシステムは、属性名のリストと それらの属性を取り出す管理オブジェクト名のリストを供給します。属性名のリストは、上記の *attributesGet* 操作と同じ方法で扱われる。管理オブジェクト名のリストの処理は以下に記述される。

8.2.8.1 属性を検索するオブジェクトの決定

名前のリストが空の場合、ファサードを通してアクセス可能なオブジェクトはすべて暗黙に要求される。そのリストが”Object”あるいは<空>のどちらかの ID 値を備えた最終コンポーネントを含んでいないアイテムを含んでいる場合、その後、そのリスト・アイテムは (それは名前がその部分的な名前で始まる管理オブジェクト(ファサードを通してアクセス可能)のセットを暗黙に要求する) 部分的な名前として役立つ。そのような名前は、管理オブジェクトの名前の終りから 1 つ以上の名前コンポーネントを取り除くことにより生成される。例えば、“ me1 ”の ID で指定され、acme telecom のために定義されたローカルのルートの下である種の“ ME ”を使用した、管理要素で始める:

acme¥.com/me1.ME/.facadeID1

注意：最終の名前コンポーネント(最後の’/’の後の)では、ID ストリング(それは’.’に先行する)が空であること。したがって、これは部分的な名前ではない。次に、種類として”EQ”を使用して、me1 の下で指定されている設備オブジェクト”eq1”があると仮定する:

acme¥.com/me1.ME/eq1.EQ/.facadeID2

最後に、この設備オブジェクト(種類”EH”を使用して)の下で指定された設備ホルダ管理オブジェクトがあると仮定する、図 4 の中で下に示されるように、すべて、ID”facadeID3”を備えた設備ホルダーを通してアクセス可能。

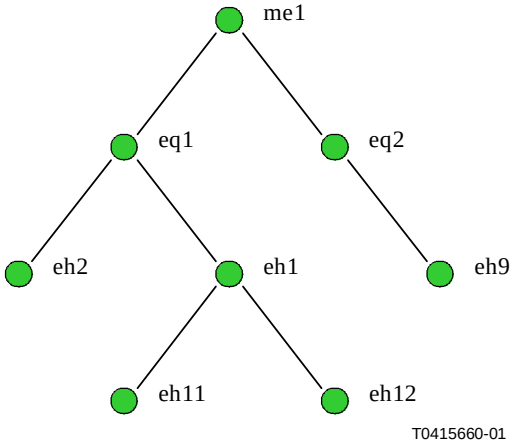

図 4 /JT-X780.1　ネーミング・ツリーの例

(ITU-T X.780.1)

部分的な名前”acme￥.com/me1.ME/eq1.EQ”が設備ホルダー・ファサード上で、attributesBulkGet 操作の管理オブジェクト名前パラメータリストの中で使用される場合、次の設備ホルダー・オブジェクトからの属性値が返される:

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

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

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

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

ファサードは、最終名前コンポーネント中の ID 値が”オブジェクト”あるいは<空>でないので、名前が部分的な名前であることを認識することができる。この例がそれを実証しなかった一方、部分的な名前によって識別された木の根のオブジェクトは、操作の範囲に含まれている。

8.2.8.2　結果を返す

データは強くタイプされた管理オブジェクト値型の中で返される、各管理オブジェクトからの一つ、指定された。ファサードが、クライアントによって提供される管理オブジェクト名にアクセスしない場合、そのオブジェクトのための値型は返されない。管理システムが同じタイプの多数のファサード・インタフェースを提供するかもしれないので、クライアントは、システム上の与えられたタイプの管理オブジェクトのすべてから値を検索する多数のインタフェースに対するこの操作を起動しなければならないかもしれない。

”name”属性に対する値が返されることをクライアントが要求しなくても、ファサードは各管理オブジェクト値型中の名前を返すものとする。値が、どれがオブジェクトを管理したかに、どれを適用するかクライアントが知るように、それはこれをするに違いない。

返された各管理オブジェクト値型に加えて、有効な値を持っているその値型中の属性の名前のリストがある。実例が要求された属性のすべてをサポートするとは限らないかもしれないので、このリストは要求された属性のリストと一致しないか。実例が要求された属性のどれもサポートしない場合、ファサードは、有効な値を含んでいる名前属性だけを備えたその実例のために管理オブジェクト値型を返すものとする。

潜在的に大量のデータが返されるかもしれないので、イテレータ設計パターンが使用される。クライアントは、返される値型の最大の数を指定します。残りはイテレータの中で返されるに違いない。イテレータが必要な場合、返り値は true であるものとする。そうでなければ、それは false でなければならない、また、イテレータ参照はゼロでなければならない。

8.2.9 *destroy*()操作

基礎ファサードに対する最終操作、*destroy*()操作は、指定された管理オブジェクトに関連したどんな資源もリリースし、かつそれを削除するために使用される。オブジェクトが *NotDeletable* の削除ポリシを持っている場合、*DeleteError* 例外が発生する。さらに、DeleteError 例外はモデル依存のオブジェクトを破棄する問題を報告する拡張可能な手段である。例えば、Trail が削除される前に Trail Termination Point オブジェクトを削除しようとすることは、DeleteError という結果になるかもしれない。しかしながら、削除ポリシを必要とするロジックをインプリメントしネーミング･ツリーの完全性を維持するために ITU-T Q.816 は、”Terminator Service”と呼ばれるサービスを定義する。破棄操作は、このサービスによって使用されるように現実に意図され、管理するシステムによって直接起動されてはならない。ターミネーターService についての詳細に関しては、ITU-T Q.816 を参照のこと。

(R) FACADE-5 －粗粒度 CORBA インタフェースのために定義されたファサード・インタフェースは、*ManagedObject_F* から上に記述され、付属資料 A の中の CORBA IDL に定義されたインタフェースを継承しなければならない(直接あるいは間接的に)。上に記述された能力がサポートされるものとする。

8.3 *AttributesBulkGet* イテレータインタフェース

8.2.8 に言及されるように、多くの結果が *attributesBulkGet* 操作に応じて返されることになっている場合、イテレータが要求される。イテレータ設計パターンは有名な CORBA デザインパターンである。大量のデータが操作に応じて再度調整されることになっている場合、イテレータインタフェースへの言及は代わりに返される。その後、クライアントは、扱いやすいチャンクの中の結果を検索するためにイテレータを問い合わせることができる。

attributes get result"イテレータの IDL 記述は下に示される。

```
interface AttributesGetResultIterator {
boolean getNext(in unsigned short howMany,
out AttributesGetResultSet results)
raises (ApplicationError);
void destroy();
}; // end of interface AttributesGetResultIterator
```

(R) FACADE-6 －*attributesBulkGet* 操作の応答で返される結果の数がクライアントによって要求された数を超える場合、管理システムは、付属資料 A の IDL の中で *AttributesGetResultIterator* 定義の記述と一致するインタフェースを備えたイテレータのインスタンスを作成しなければならない。

(R) FACADE-7 －クライアントがイテレータに対する *getNext* 操作を起動するごとに、それは結果の次のセットを返さなければならない。イテレータは、いくつの結果がクライアントによって既に検索されたか把握しなければならないし、結果をすべて一度に返さなければならない。*attributesBulkGet* 操作への応答で最初に返された結果は、イテレータによって再び返してはならない。イテレータは、*getNext* 操作に応答として、多くても *howMany* パラメータの値によって示された名前の数を返さなければならない。イテレータは要求されたバッチ・サイズ未満を返してもよいし、より多くの結果が利用できるまで返す効率のバランスを考えることはブロックにするため必要な大きなバッチという結果になる。返すべき (返されているものに加えての)さらに多くの結果がある場合、*getNext* 操作の戻り値は true でなければならないし、そうでなければ false でなければならない。もし *howMany* が 0 にセットされなかったか、そうすることがイテレータを獲得する

ことをクライアントに強いたように、返す結果はこれ以上なければ、イテレータが空の結果セットを返してはいけない。

(R) FACADE-8　－管理システムは、イテレータのライフ・サイクルをコントロールしなければならない。しかし、管理者が最後の結果を取り出す前に結果を検索することをやめたい場合、破棄操作が提供される。*破棄*操作の実行に際して、イテレータは、それが使用しているすべての資源を解放し、それ自体を削除しなければならない。最後の結果を返す際、イテレータは自分自身を削除しなければならない。イテレータが不当に長い期間の間未使用の場合、イテレータも管理システムにより破棄してもよい。

8.4　ファクトリ・インスタンス化

ファクトリは他のオブジェクトのインスタンス化するために使用される永続的なオブジェクト・インタフェースである。ファクトリの使用はよく知られた CORBA デザイン・パターンに従う。ファクトリは、管理するシステムに管理オブジェクトの新しいインスタンスを作成する方法を供給するきめの細かいインタフェース上で使用される。たとえファサードがファクトリの役割を果たすことができるので、粗粒度インタフェース上で厳密に要求されなかったとしても、個別のファクトリ・インタフェースは粗粒度インタフェース上で使用されなければならない。これを実行するもう 1 つの利点は粗粒度できめの細かいアプローチをより高い互換性をもつようにするためである。別の利点はサブクラスによる上位クラスオブジェクトのための生成操作の継承を防ぐことである。ファクトリ・インタフェースが管理オブジェクトの継承階層に従わないため、この問題は個別ファクトリでは発生しない。粗粒度インタフェース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・インタフェースは、それらがきめの細かいインタフェースのために定義したものと同じである。第 9 節を参照。

(R) FACADE-6 ----- ある管理システムは、インスタンスが作成された管理オブジェクトの各クラスに少なくとも 1 つのファクトリ・インタフェースを提供しなければならない。システム上でインスタンスを作成することができない管理オブジェクトクラスのためのファクトリ・インタフェースを提供する必要はない。ファクトリ・インタフェースは ITU-T Q.816 に定義されて、ファクトリ・ファインダ・サービスで登録される。粗粒度インタフェース上のファクトリは、新しく作成されたオブジェクトへの言及ではなく生成操作に応じて皆無の言及を返してもよい。生成操作に応じてファクトリによって返された管理オブジェクト名は、ITU-T Q.816.1 に定義された規則を指定する管理オブジェクトによって新しいオブジェクトにアクセスするために使用されるかもしれないファサードを示さなければならない。

９．粗粒度の CORBA モデリングガイドライン

本章では、粗粒度の IDL インタフェースを定義するためのルールを定義する。粗粒度インタフェースは、ITU-T X.780 に定義されたガイドラインによってまず最初に細粒度インタフェースを定義することにより生成される。ITU-T X.780 は、GDMO インタフェースを IDL に翻訳することと共に、最初から IDL インタフェースを作成することもカバーする。一旦細粒度のインタフェースが定義されると、各管理オブジェクトインタフェースのためのファサードインタフェースが、10 節に定義されるルールによって開発される。細粒度の管理オブジェクトインタフェースの仕様書が保持されるものとする。データ型、値の型、例外、通知、工場を含む細粒度のインタフェースのために定義された他の構造物は、粗粒度のインタフェース上で修正無く再利用される。

１０．細粒度のモデルから粗粒度のモデルへの翻訳のためのガイドライン

粗粒度の IDL モデルは、下記のステップで細粒度の IDL モデルから生成される。

1) ファサードインタフェースは、各管理オブジェクトインタフェースのために生成される。管理オブジェクトインタフェースは、ManagedObjectインタフェースから直接または間接的に由来するインタフェースである。

2) ファサードインタフェースは、細粒度オブジェクトインタフェースと同じIDLモジュール内に生成される。これは、細粒度モデルのために定義された全てのタイプのためのタイプ定義を含まなければならないことから、モデルの生成者を楽にする。

ファサードインタフェースは、個別のファイルの中に生成されるかもしれないし、細粒度のファイルの新規バージョンに含まれるかもしれない。個別のファイル内の場合、細粒度モデルを含むファイルは、編集のために含まれなければならない。

ファイルを横断したモジュール分割は、OMG標準によって許可される。基本的にIDLモジュールは名前空間を定義する。モジュール内では、全ての名前はユニークでなければならない。従って、例えば、あるモジュールは、管理オブジェクト_Fという名前の2つのインタフェースを含むことはできない。しかし、2つの異なるモジュールは同一の名前を含むことができ、他のモジュール内にモジュールを含むことができる。モジュールがファイルを横断して分割されていると、ユニークな名称のルールは依然適用する。同一モジュール中であるが分割されたファイル内における名前の重複は許可されない。

付属資料 A内のIDLはitut_X780という名前の単一のモジュール内に定義され、これはITU-T X.780で使用されるモジュールと同一である。従ってこのモジュールは、ファイルを横断して分割されている。ITU-T X.780内のIDL中で使用される名前は本勧告内のIDLで再利用できない、ということが1つのインパクトである。しかしながら、ITU-T X.780に定義されるどのIDL構造も、プリコンパイラへの指示内に単純に含むことによって本勧告で再利用できることは利点である。もしIDLが個別のモジュールにあれば要求されるような、タイプ定義あるいはスコープ名は要求されない。遵守と適合が、IDLモジュールではなくドキュメントに依然基づくことに注意すること。従って、システムは本勧告へ適合するのではなくITU-T X.780へ適合するかもしれない。

3) ファサードインタフェースの名前は、(アンダースコアに大文字Fの付いた) “_F”が付与されて生成される細粒度のインタフェース名となる。従って、”Equipment”管理オブジェクトのために生成されるファサードインタフェースは、”Equipment_F”という名前になる。

4) もし細粒度のオブジェクトインタフェースが直接管理オブジェクトインタフェースを継承する場合、そのために生成されるファサードインタフェースは、管理オブジェクト_Fインタフェースを継承する。もし細粒度オブジェクトインタフェースが代わりに管理オブジェクトインタフェースのサブクラスを継承する場合、そのために生成されるファサードインタフェースは、サブクラスにより生成されたファサードインタフェースを継承する。従って、ファサードインタフェースの継承階層は、管理オブジェクトインタフェースのものと一致する。例えば、Equipmentインタフェースを継承するEquipmentHolder細粒度オブジェクトインタフェースについて述べる。EquipmentHolder_FインタフェースはEquipment_Fインタフェースを継承する。全ての細粒度オブジェクトのスーパークラスのためのfacadeインタフェースは、細粒度オブジェクトのfacadeが生成される前に生成されていなければならない。

5) 細粒度オブジェクトインタフェースの全内容は、下記の変更をした上でファサードインタフェースにコピーされる。

- 各操作の第一パラメータとして、NameType型のnameと命名されたパラメータが追加される。このパラメータは、操作が起動されるべき対象の管理オブジェクトの名前の中で渡すために使用される。操作が既に”name”という名のパラメータを有している場合、それは改名される。
- 管理オブジェクトのIORを使用するどのパラメータまたはリターンの型もNameTypeに翻訳されなければならない。管理オブジェクトのIOR値は、管理オブジェクトインタフェースかサブクラスインタフェースへの参照として識別されるであろう。このようなタイプの値の使用は、細粒度オブジェクトインタフェース上で排除され、もしあってもほとんど遭遇することはない。
- 細粒度インタフェース規定上の等価な構造の振る舞いの記述を複写するというより参照することは、粗粒度インタフェース規定上の振る舞いの記述のために受容される。

6) 細粒度のモデリングガイドラインは、管理オブジェクトの属性をモデル化するためにOMG IDLの属性の利用を現在禁止する。これは、ユーザ定義の例外が属性へのアクセス操作上に乗ることをOMG IDLが許容しないからである。従って、代わりに、管理オブジェクトの属性は、属性の値を獲得または設

定するために用いられる操作と、属性を追加または削除する操作とが個別にモデル化されている。しかしながら今後､OMGはおそらく属性にアクセスする操作に関するユーザ定義の例外を許容するであろう。もしこれが起こり、細粒度オブジェクトインタフェースがIDL属性を使用すれば、ファサードを生成するときに属性は個別のIDL操作に翻訳され、nameパラメータはこれら操作の第一パラメータとして追加される。

7) 残りのdata型やvalue型のような細粒度インタフェースIDLは、粗粒度インタフェース上で変更なく用いられる。

１１．粗粒度 IDL の遵守と適合

本節では、これらガイドラインへの遵守を主張する他の標準文書が満たさなければならない基準と、本標準への適合を主張するシステムが実装しなければならない機能を定義する。

11.1 標準文書の遵守

これらガイドラインへの遵守を主張するいかなる仕様も下記を満たすこと。

1) ITU-T X.780の標準文書遵守要件を全てサポートすること。

2) 10節で定義された細粒度IDLから粗粒度IDLへのマッピングルールに従うこと。

11.2 システムの適合

本標準への適合を主張する実装は下記を満たすこと。

1) 8節で明示された全てのファサード、ファクトリ、イテレータのインスタンス化要件を満たすこと。

2) 本勧告内のガイドラインを遵守するIDLインタフェースを実装すること。11.1節を参照のこと。

11.3 適合宣言のガイドライン

適合宣言を記述する場合、これらのガイドラインのユーザは、注意を払わねばならない。IDL モジュールが名前空間として使用されており、OMG IDL ルールによって許可されているようにファイルを横断して分割しているかもしれないからである。従って、モジュールが拡張される場合、その名前は変わらない。代わりに、新規の IDL ファイルが単に加えられる。よって、単純に適合宣言においてモジュール名を記載することは、IDL インタフェースセットを識別するのには不十分である。適合宣言は、IDL の正確なバージョンが必ず識別されるようにするために、文書名とその出版年を明らかにしなければならない。

付属資料 A

Coarse-grained modelling IDL

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

#ifndef ITUT_X780_1_IDL
#define ITUT_X780_1_IDL

#include <itut_x780.idl>

#pragma prefix "itu.int"

module itut_x780 {

// IMPORTED TYPES
// DATA TYPES

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

     struct AttributesGetResultType {
          ManagedObjectValueType  attributes;
          StringSetType           attributeNames; };

     typedef sequence <AttributesGetResultType>
AttributesGetResultSetType;

     interface AttributesGetResultIterator;
¥
// ATTRIBUTES GET RESULT ITERATOR INTERFACE

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

     interface AttributesGetResultIterator {

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

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

          /** This method is used to destroy the iterator and release its
```

```
        resources. The iterator, though, is automatically destroyed after
        the last results are returned, and may be destroyed if unused for
        an unreasonably long period. */

        void destroy();

    }; // end of interface AttributesGetResultIterator

// MANAGED OBJECT FACADE

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

    interface ManagedObject_F {

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

        ObjectClassType objectClassGet(in NameType name)
            raises (ApplicationError);

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

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

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

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

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

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

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

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

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

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

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

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

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

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

        Since a potentially large amount of data may be returned, the iterator
        design pattern is used. The client specifies the maximum number
        of value types to be returned. The rest must be returned in an iterator.
        If an iterator is used, the return value shall be true. Otherwise,
        it shall be false and the iterator reference shall be null.
        @param names            The names of the managed objects from which
                                to retrieve the attribute values.
        @param attributeNames   The names of the attributes to retrieve.
```

```
        @param howMany          The maximum number of value types to return
                                in the attributes parameter.
        @param attributes       The first batch of results.
        @param iterator         A reference to an iterator, if needed.
                                Otherwise, null.
        @return                 True if an iterator is being returned,
                                otherwise false.
        */

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

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

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

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

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

    }; // end of ManagedObject_F interface
```

// ApplicationErrorConst Module

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

module ApplicationErrorConst {

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

     const short objectNotFound = 4;

}; // end of module ApplicationErrorConst

}; // end of module itut_x780

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