Vol. 35 No. 9　　情　報　処　理　　Sep. 1994

解　説

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 1.1†

佐藤広行††　大野邦夫†††

分散オブジェクト環境を実現するための基盤となるCORBA 1.1について，CORBAの目指す分散環境の姿とCORBAの構成要素，およびCORBAのインタフェース仕様についての解説[4]を行う．さらに，CORBA 2.0の標準化状況と課題を紹介する．

1. CORBAの狙い

OMGの活動は，今から5年前，HP社のNewWaveから話が始まる．NewWaveとは，マイクロソフト社のWindowsにマッキントッシュのようなGUIを付加するソフトウェアである．アイコンをダブル・クリックすると，アプリケーション・プログラムにデータがロードされて立ち上がる便利ものだ．

この仕掛けを実現するために，アプリケーション・プログラムとデータとのペアをアイコンに結びつけるデータベース機構 OMF (Object Management Facility) をNewWaveはもっている．このOMFは，アプリケーション・プログラムとデータとのペアをオブジェクトという単位で取り扱う．このオブジェクトの管理機構がオブジェクト・マネジメント技術の発端となったのである．

このOMFにおけるオブジェクトの管理範囲は，一つの計算機内に限られていた．しかし，ユーザの視点からみれば，ネットワーク上に分散配置された計算機に含まれるすべての資源を管理できたほうが便利である．このためには，ネットワーク上に広がった計算機内に存在するオブジェクト群の統一的な管理機構が必要であり，また，そのオブジェクト群のインタフェース部分は，ユーザからの統一的な使用に耐えられるように，標準化される必要がある．

すなわち，CORBAの狙いとは，ネットワークで接続された遠隔の計算機上やユーザと同じ計算機上などに位置するさまざまな資源をオブジェクトとして捉え，クライアントから一様に取り扱える仕組みを提供することである．

このような目的を掲げて，OMGは，1989年4月に結成され，その一つ目の標準化項目がCORBAであった．そして1991年10月にCORBA 1.1が制定された．

2. オブジェクト間通信の考え方

次に，CORBAのおおよその動作を説明しよう．CORBAの動作には，リクエストの発行者であるクライアントとリクエストの仲介者であるORB (Object Request Broker)，それにリクエストの実行者であるオブジェクトが登場する．

使用したいオブジェクトの「インタフェース定義」と「オブジェクト・リファレンス」とをクライアントが知っているということが，CORBAを使用するための前提条件となる．インタフェース定義には，クライアントがオブジェクトに作用できる機能がインタフェースとして定義されている．オブジェクト・リファレンスは，オブジェクトを識別するための識別子である．

実際の動作は，**図-1**で示すように，①クライアントは，この識別子とインタフェースとをリクエストとしてORBに引き渡す．② ORBは，この識別子を使ってオブジェクトを探し出し，リクエストを引き渡す．③オブジェクトがその作用の結果をORBに戻すと，ORBは，この結果をクライアントに返すという順になる．

さて，クライアントが，ORBにリクエストを引き渡す方法には，実は，静的起動と動的起動と

† Comment on Distributed Object, CORBA, by Hiroyuki SATO (Corporate Software Planning & Development Center, Canon Inc.) and Kunio OHNO (NTT Human Interface Laboratories).

†† キヤノン(株)ソフトウェア企画開発センター
††† NTT ヒューマンインタフェース研究所

の2種類がある．どちらの方法でリクエストを発行したかをオブジェクト側で判別することはできず，リクエストの受け側からみれば，等価となる．

2.1 静的起動

ORB にリクエストを引き渡すとは，アプリケーション・プログラムの中から，リクエストを発行することである．プログラムとしてC言語を例にとると，静的起動では，リクエストの発行がC言語での関数の呼出しに置き替わる．すなわち，定義されたインタフェースが，C言語の関数に置き換えられ使用される．

このように，インタフェースの定義から各プログラム言語への置換えを言語バインディングと呼んでいる．現時点では，C言語への言語バインディングのみが規定されており，C++ 言語への言語バインディングは標準を作成中である．

さて，リクエストの発行が，関数の呼出しとなるわけだが，この関数の実体は，スタブと呼ばれる．このスタブの中で，オブジェクト・リファレンスやインタフェース情報が，データ列に変換され，ネットワークに流し込まれる．このように，静的起動は，実行時でなく，コンパイル時に，リクエスト発行メカニズムを生成する．このため最適化しやすく，後から述べる動的起動よりも，実行時効率が優れている．

2.2 動的起動

動的起動では，インタフェース定義に従って，アプリケーション・プログラム内でリクエストを作成し，インタフェース情報をデータ列に詰め，このデータ列をリクエストとして発行する．すなわち，リクエストの個々の要素を，Named-Value という構造体に格納し，この構造体を Request 型の変数に格納して，オペレーション Request_invoke() で発行する．

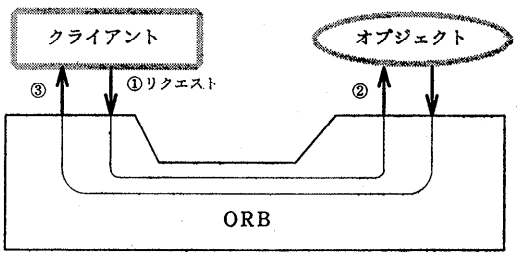


図-1 オブジェクトとの通信

また，動的起動では，インタフェース情報を実行時に読み出せるような，インタフェース・リポジトリも提供している．リクエストを実行する場合，まず，インタフェース・リポジトリに問い合わせて，この情報に沿ってリクエストを生成し，動的起動を行うこともできる．

このように，動的起動では，実行時にリクエストのデータ列を作成するため，静的起動より実行時効率は落ちるが，プログラムの柔軟性が増す．

このほかに，発行されたリクエストが定義したインタフェースに合致しているかどうかのチェックにもインタフェース・リポジトリは利用される．

3. オブジェクトの記述方法（インタフェース定義言語）

CORBA において，オブジェクトは，インタフェース定義言語 (IDL: Interface Definition Language) で定義される．この定義とは，クライアントからみた定義なのだが，このことの意味をまず説明し，次に IDL の具体例を解説する．

3.1 型とクラスの分離

CORBA のような分散オブジェクト・モデルでは，オブジェクトと言ったとき，二つの側面が存在する．すなわち，クライアントからみた側面と実装上の側面である．クライアントからの側面をインタフェース型，あるいは，型と呼び，実装上の側面をクラスと呼ぶ．CORBA では，オブジェクトの定義と言った場合，インタフェース型の定義のことであり，このインタフェースを備えたものがオブジェクトである．(クラスとは，オブジェクトの実装プログラム内での抽象化機構を指す．)

3.2 インタフェース定義言語 (IDL)

この IDL では，オペレーション，属性，例外，型，定数を宣言できる．すなわち，このインタフェースを実装するオブジェクトが，実行できるオペレーションと，保持できる属性と，発生できる例外と，使用できる新しい型／定数とを宣言する．クライアントは，IDL で宣言されたインタフェースのみを知れば，オブジェクトを自由に利用できる．

実際の IDL の定義の例とその使用方法を以下に取り上げよう．

Rectangle というオブジェクトのインタフェー

ス型の宣言の例を示してみる．このオブジェクトには，操作できるオペレーション fill がある．オペレーション fill は，引数として color と Pattern をもつ．

IDL は，C++ の字句解析規則に従い，C++ のプリプロセッサをサポートしている．構文規則としては，C++ からアルゴリズムの記述部や変数などを除き，リモート・プロシージャ・コールで必要となる予約語を追加した言語とでもいおうか．

図-2 で示すようにこの IDL で定義されたインタフェースの情報は，IDL コンパイラを通して，スタブとインタフェース・リポジトリとスケルトンに変換される．静的起動でスタブが使用され，動的起動でインタフェース・リポジトリが使用される．また，オブジェクトの実装部分を作成するために，インプリメンテーション・インストレーションという定義を行い，その情報がインプリメンテーション・リポジトリに蓄積される．(この部分は，実装に依存するため，定義外となっている．)

まず，静的起動の例をあげると，**図-3** のインタフェース宣言の例からC言語へのバインディングは，**図-4** のようになる．

インタフェース Rectangle のオペレーション fill は，C 言語では，関数 Rectangle_fill の呼出となる．第1引数 Rectangle obj は，オブジェクトの識別子であり，オブジェクト・リファレンスと呼ばれている．第2引数 Environment *ev には，例外や認証の情報が格納される．C言語バインディングによって，この二つの引数が追加されたわけだ．

C 言語上では，この Rectangle_fill を呼び出すことによって，オペレーション fill を発行したことと等価になる．IDL コンパイラは，この関数のライブラリもはきだすので，これでリンクする．

このライブラリはスタブと呼ばれている．スタブ内の機構は，CORBA の規定外である．スタブ内では，このリクエストをデータ列に詰め替え (marshalling)，ネットワークを通してオブジェクト位置探索機構に引き渡す．

IDL コンパイラは，通常，スケルトンと呼ばれるオブジェクト側の受信側のスタブも生成する．これを利用してオブジェクト側のソースを作る．

次に，図-3 のインタフェースを動的起動で発行した（実行文の）例を**図-5** に示す．(以下の IDL，C 言語の例では，引数などの詳細は，略してある．)

まず，ORB_create_list で引数のリスト arg_list を生成する．次にインタフェース名と型とを arg_list に詰める．そして，Object_create_request

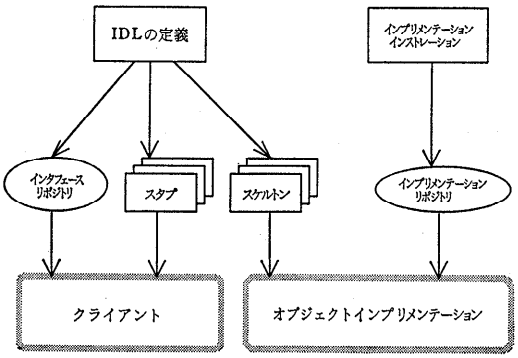


図-2 IDL からの定義情報の流れ

```
                                        /* IDL */
  /* インタフェース型 Rectangle の宣言     */
interface Rectangle {
          /* オペレーション fill の宣言     */
    void fill ( in RGB color,
                in long Pattern );
};
```

図-3 インタフェース宣言の例

```
                                        /* C */
typedef Object Rectangle;
extern long Rectangle_fill (
                Rectangle obj,
                Environment *ev,
                RGB color
                long Pattern
                );
```

図-4 IDL コンパイラを通した例

```
                                              /* C */
                          /* 引数リストの生成 */
ORB_create_list (       ORB orb_obj, ,
                         NVList * arg_list);
                        /* 以下で引数を詰める */
arg_list -> name = "fill";
arg_list -> any._type = OperationDef;
arg_list++;
/* 同様に "color" と "Pattern" も詰める.   */
      /* 引数リストを含むリクエストの生成 */
Object_create_request ( Object target_obj, ,
                         NVList * arg_list, ,
                         Request * req_obj);
                        /* リクエストの発行 */
Request_invoke (Request req_obj);
```

図-5 動的起動の例

で，この arg_list をもつリクエスト req_obj を作る．最後に Request_invoke でリクエストを発行する．

図-3 のインタフェースは，インタフェース・リポジトリ内に格納される．インタフェースの構造がそのままインタフェース・リポジトリに反映される．

インタフェース・リポジトリ内のインタフェース Rectangle を探索するには，**図-6** のような関数の呼出しを用いる．

4. CORBA の構成

ORB は，基本的には，**図-7** で示すように三つの部分から構成されている．一つ目は，クライアントが ORB にリクエストを引き渡す手続きであり，**3.** で詳しく述べたように「スタブ」と「動的呼出し」がある．二つ目は，オブジェクト・インプリメンテーションが使用できる手続きである「オブジェクト・アダプタ」と「スケルトン」である．三つ目は，クライアントおよびオブジェクト・インプリメンテーションが共通に使用できる手続きである「ORB インタフェース」である．

ここで，CORBA では，リクエストの受け側をオブジェクトと言わず，オブジェクト・インプリメンテーションと呼んでいる．これは，ORB と実際に交渉をもつのは，多くのオブジェクトを実装しているサーバであるという点を強調しているためだ．(もちろん，オブジェクトを一つしか含まないサーバということもあるが)．

4.1 オブジェクト・アダプタ

オブジェクト・アダプタは，オブジェクト・インプリメンテーションが，ORB にアクセスするためのインタフェースであり，種々のアダプタが作成可能である．すべての ORB が標準で装備するアダプタとして，基本オブジェクト・アダプタ (BOA : Basic Object Adapter) があり，そのインタフェースが定義されている．

図-8 を用いながら，BOA の動作の一例を説明しよう．

1) BOA がオブジェクト・インプリメンテーションを活性化する．(ここで活性化とは，サーバを起動すること．)

2) オブジェクト・インプリメンテーションは，リクエストの受入れ準備ができたことを BOA に知らせる．

3) オブジェクトにリクエストが到着すると，そのオブジェクトの活性化をするようにオブジェクト・インプリメンテーションに知らせる．

4) リクエストの内容に沿って適宜にメソッドを選ぶ．このメソッドの選択時には，「IDL スケルトン」が使用される．これは，実装の範囲であり，そのインタフェースは定義されていない．

4.2 ORB インタフェース

ORB インタフェースは，クライアントとオブジェクト・インプリメンテーションとの両方から利用されるインタフェースである．たとえば，オブジェクト・リファレンスのストリングへの変換やオブジェクト・リファレンスの複写を行うために利用する．

このストリングへの変換のオペレーションは，プロセスの停止後もオブジェクト・リファレンスを保持するために設けられた．複写のオペレーションは，他のプロセスにオブジェクト・リファレンスを引き渡すのに必要となる．

このように，オブジェクト・リファレンスは，一般の変数とは，違った扱いを受けている．実は，このオブジェクト・リファレンスの実体には，オブジェクトの位置を発見するためのキーと

```
/* C */
Repository_lookup_name (, , "Rectangle", ,);
```

図-6 インタフェース・リポジトリの例

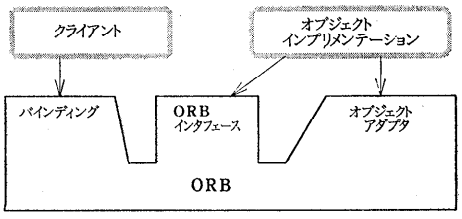


図-7 ORB の構成図

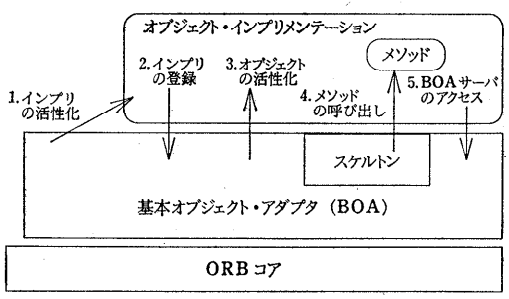


図-8 基本オブジェクト・アダプタ (BOA) の動作

なる情報やインタフェース型を特定する情報や発信者の認証情報などが含まれている．このため，ユーザが直接には触れられないようにしており，参照するには，ORB インタフェースを使う必要がある．

BOA や ORB インタフェースのインタフェースも，IDL で定義されている．しかし，BOA や ORB インタフェースは，一般のオブジェクトと同様に取り扱うことはできない．(たとえば，クライアントが勝手に生成できない．) このような BOA や ORB インタフェースは，擬似オブジェクトと呼び，通常のオブジェクトと区別している．

5. CORBA の将来

さて，現状の CORBA には，二つの動きがある．一つが，CORBA 1.1 の修正版である，CORBA 1.2[1] の TC (Technical Committee) での承認である．もう一つが，CORBA 1.1 の拡張版である，CORBA 2.0 [2),3)] の RFP (Request for Proposal) の TF (Task Force) での評価である．

5.1 CORBA 1.2

CORBA 1.1 から CORBA 1.2 への主たる変更点は以下のとおりである．

1) CORBA仕様内で，グローバルのスコープで定義されたすべての名称は，モジュール CORBA の下で定義されることとなった．たとえば，any, TypeCode は，CORBA_any, CORBA_TypeCode となる．

2) これにともない，不要なモジュールStExcep が削除された．また，ORB を含む名称は，ORB を削除することとなった．たとえば，ORBfree, ORBStatus は，CORBA_free, CORBA_Status となった．

3) IDL の型は，C言語の typedef にマッピングされることになった．たとえば，CORBA_char, CORBA_long など．

4) IDL の式と属性の宣言内で，sequence 型と array 型は，必ず，名称をもたなければならない．たとえば，

```
interface foo {
    sequence <long> opl();
};
```

から

```
interface foo {
    typedef sequence <long> ban;
    ban opl();
};
```

と変更になった．このほか，タイプミスの修正や説明の追加などが施された．

5.2 CORBA 2.0

4. で述べたように，オブジェクトを識別するオブジェクト・リファレンスは，各メーカの実装ごとに違っているのが実情である．CORBA は，クライアントからみた分散オブジェクト・モデルの標準化を図ったものであり，CORBA の実際の実装については，一切，規定していないからだ．

このような CORBA 1.1 準拠の製品が出てみると，実装の異なっている CORBA 間では，リクエストのやり取りができない，プログラムの移植性がないという問題点が浮かび上がってきた．これでは，異機種分散環境において，オブジェクトを統一的に取り扱えるという OMG の使命に反してしまう．この問題の解決を目指すのが，CORBA 2.0 の標準化である．CORBA 2.0 は，二つの RFP で構成されている．一つが，「相互運用性」と「初期化」であり，もう一つが，「インタフェース・リポジトリ」である．

「相互運用性」とは，「独自に開発された ORB A と ORB B において，ORB A 上のクライアントが，ORB B 上のオブジェクトに，IDL で定義されたオペレーションを起動できること」と定義されており，以下の事項が要求されている．

1) 二つのベンダの ORB が相手の実装方法に関する事前の知識なしで相互運用可能であること．

2) すべての ORB の機能を保持していること．

3) ORB に由来する情報 (セキュリティなど) の内容と意味とが，ORB の境界を越えても保持されること．

CORBA 1.1 では，ORB や BOA といった擬似オブジェクトを定義している．しかし，これらの擬似オブジェクトのオブジェクト・リファレンスの入手方法が定義されていないため，移植可能なプログラムを書くことができない．これらの解決を目指すのが「初期化」と呼ばれる部分であり，以下の事項が要求されている．

1) ORB, BOA, インタフェース・リポジト

リ，名前サービスへのオブジェクト・リファレンスの入手方法を決定すること．

2) BOA によってのみ提供される機能 (object_create など) へのアクセスを CORBA オブジェクト以外からでも可能とすること．(たとえば，インストール時にオブジェクトを生成するプログラムのため)

CORBA 1.1 の実装経験から，「インタフェース・リポジトリ」は，新しいインタフェース定義を獲得する標準の機構が必要であり，また，既存のインンタフェース定義を修正するための機構も必要であることが分かった．そして，この標準化案には，以下の事項が要求されている．

1) インタフェース定義の更新がもたらす，他のインタフェース定義への影響 (特に，階層と継承に対して) について考慮すること．

2) インタフェース定義が削除できるのか．できる場合，この定義を用いているオブジェクトはどうなるのかを説明すること．

3) インタフェース定義のバージョン管理をどのように行うのか．

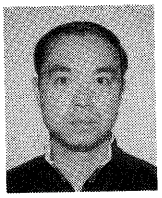
4) コンカレントなアクセスに対する制御．

OMG では，本年内に，ORBA 2.0, Obect Service RFP 2, 共通ファミリティ，C++ 言語マッピング，Smalltalk 言語マッピング，セキュリティなど多くの標準の制定を目指しており，94 年は，OMG にとって飛躍の年となりそうだ．

参考文献

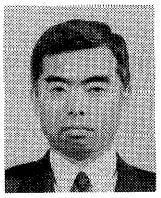
1) "The Common Object Request Broker: Architecture and Specification 1.2" OMG TC Doc 93-12-43.
2) "Object Request Broker 2.0 Interoperability and Initialization Request for Proposals" OMG TC Doc 93-9-15.
3) "Object Request Broker 2.0 Interface Repository Request for Proposals" OMG TC Doc 93-9-16.
4) 佐藤：OMG におけるオブジェクト指向技術，1993 情報学シンポジウム．

(平成 6 年 3 月 23 日受付)



佐藤　広行 (正会員)

1956 年生．1979 年早稲田大学電子通信学科卒業．同年日立製作所入社．同社計測機事業部にて，工業計測機器，医療機器のソフトウェア開発に従事．1991 年キヤノン入社．同社ソフトウェア企画開発センターにて NewWave の日本語化に従事．また，OMG の会員であり，CORBA 1.1 仕様等の翻訳を行う．



大野　邦夫

1945 年生．1968 年東京工業大学工学部機械工学科卒業．1970 年同大学院機械工学専攻修士課程修了．同年，NTT の前身である日本電信電話公社に入社．電気通信研究所に配属され，以後，端末機器，ワークステーション等の研究開発に従事．日本機械学会，電子情報通信学会各会員．e-mail:ohno@aether.ntt.jp