システムソフトウェアと
オペレーティング・システム 73－10
（1996．8．26）

分散 Java 実行のためのポータブルな ORB の構成法

平野 聡　　塚本 享治
hirano@etl.go.jp

電子技術総合研究所

概要

HORB は Java を分散オブジェクト指向言語に拡張する Java 用の Native ORB(オブジェクト間通信機構)である。アプリケーションと ORB 自身がネットワーク可搬であるためには、ポータビリティとインターオペラビリティが必要である。本論文では、Java で記述するネットワーク可搬な ORB の構成法について特に、リモートメソッド実行における効率的なクラス継承の実現と、インターフェース継承による分散透明性の実現について述べる。

Building a Portable ORB for Java

HIRANO Satoshi　　TSUKAMOTO Michiharu
hirano@etl.go.jp

Electrotechnical Laboratory

HORB is a native ORB (Object Request Broker) for Java that extends Java as a distributed object oriented language. Applications of HORB and HORB itself are network portable and trasnferable, since HORB provides both portability and interoperability among different kinds of OSes. This paper describes the organization of HORB. Especially it focuses on class inheritance in remote method invocation and distribution transparency by interface inheritance.

1 はじめに

ソフトウェアをひとつのバイナリコードとして利用者に供給すると、利用者に提供できる機能は設計時に設定した機能だけである。そこで現在では、利用者が必要に応じて機能を拡張可能なように、ソフトウェアを複数の部品(コンポーネント)から動的に構成することが一般的になっている。たとえば、ワードプロセッサで文書を作成している際に、表や図を貼り込むと、内部ではワードプロセッサに表計算や作図のコンポーネントが組み込まれる。そのようなコンポーネントは OLE や Java 等のオブジェクト指向技術で実現することが多い。更に、Java や OLE 等の部品をネットワークを介して他のマシンからダウンロードし、クライアント上で実行することが一般的になってきた。ネットワーク内でダウンロードされて実行されるソフトウェアは、サーバと通信してデータベースをアクセスするなど、分散処理を行うことが多い。

複数のマシンに分散するオブジェクトを互いに連携して分散実行するには、オブジェクト間で通信をする必要である。そのために、言語にオブジェクト間通信の機能を提供する ORB(Object Request Broker)の開発が進んでいる。CORBA や OLE(DCOM)が特定の言語に依存せずに複数の言語から利用可能な共通 ORB(Common ORB)であるのに対し、本論文で述べる HORB は Java 用の ORB(Native ORB)である。HORB は、Java を分散オブジェクト指向言語に自然に拡張し高いプログラマビリティを実現しながら、異機種間での可搬性と相互運用性を実現している。また、HORB 自身も部品としてネットワーク内で自動的にダウンロードして実行されることが可能なように、可搬性と相互運用性を備えている。ダウンロードされる HORB の大きさは最小２４ＫＢと非常に小さい。

本論文[1]では以下、HORB の概要について述べた後、ネットワーク可搬な ORB の要件について述べる。4 章では要件を満たす ORB の構成法について特に、リモートメソッド実行におけるクラス継承の実現と、インターフェース継承による分散透明性の実現について述べる。

1.1 Java について

Java は C++からポインタやテンプレート等の複雑な機能を取り除いて簡略化し、マルチスレッド、各機種で共通のクラスライブラリ等の機能を追加したオブジェクト指向言語である。Java コンパイラはクラス定義を含むソースファイル(java ファイル)から中間コード(class ファイル)を生成する。Java の実行は Java VM(仮想マシン、中間コードインタプリタ)が中間コードを実行することにより行われる。Java の中間コードはマシンアーキテクチャに依存しないため、Java VM が動作する機種であれば、どんな機種でも Java プログラムを実行することが可能である。また、Java VM を内蔵する WWW ブラウザは、WWW サーバから class ファイルをダウンロードして「アプレット」として実行することができる。Java は既に市場のほとんどの機種で動作する。

2 HORB の概要と機能

HORB は Java を分散オブジェクト指向言語に拡張する。また、分散システム構築フレームワークとしての機能を有する。以下に主要な分散オブジェクト機能を列挙する[1]。

リモートオブジェクトの生成: クライアント上のオブジェクト(アプレットを含む。以下、単にクライアントと記す)からサーバ上にリモートオブジェクトを新しく生成する[2]。

リモートオブジェクトへの接続: クライアントからサーバ上に既に存在するリモートオブジェクトに接続する。オブジェクトにはクライアントに対応するスレッドが生成され並行実行するため、サーバは複数のクライアントに同時にサービスを行うことが可能である。

リモートメソッド呼び出し: クライアントからリモートオブジェクトのメソッドを呼び出す(同期または非同期)。

オブジェクト転送: メソッドの呼び出しの際に、任意の基本型、オブジェクト、配列を渡したり、返り値として受け取る。オブジェクト転送では、リモートオブジェクトへの参照は参照渡し、それ以外は値渡しとなる。構造を有するオブジェクトはオブジェクトの参照関係が到達する範囲で、参照関係の構造を保ったまま転送される。

他に、分散ガベージコレクションや、クラス中の特定のメソッドにアクセス制限を加えたり、オブ

[1] 本論文は現在開発中のバージョンの機能や設計に基づいて記述されており、公開中のバージョンとは異なっている場合がある。また、本論文に記述してある通りの機能や設計が将来そのまま公開されるとは限らない。http://ring.etl.go.jp/openlab/horb

[2] クライアント、サーバという役割は記述の便宜上のもので、実際にはあるオブジェクトがクライアントとサーバの両方の役割を果たすことも可能である。

ジェクトが転送される際に特定のメソッドを自動的に呼び出す(hooks)など、多様なプログラミング機能を有する。
分散システム構築フレームワークとして次のような機能を有する。

セキュリティ: 分散アクセスコントロールリストにより、クラス単位でホストと利用者を制限、認証する。
ネットワークローディング: サーバからクライアントへ、クライアントからサーバへ、あるいは、インターネット中から実行場所へクラスのコードを供給する。
永続オブジェクト: オブジェクトを他ホスト上のファイルにセーブ/ロードする。
分散オブジェクト管理: オブジェクト、スレッドを分散管理する。
WWW との統合: HORB はシステム全体を WWW クライアントや WWW サーバ中で動作させることが可能であるため、WWW との統合が容易である。

多くのベンダーが提供する Java の拡張機能を併用することで、高度な分散システムを構築することが可能である。

3 ネットワーク可搬な ORB の要件

Windows、Macintosh、Unix 等、利用者の多い一般的な機種で同じプログラムが動作し、更に異機種間で相互運用が可能な分散言語処理系で広く利用可能なものは今まで存在しなかったため、それを実現することが HORB の第 1 の目標である。それらを含め、HORB がアプリケーションに提供すべき機能の要件を以下のように設定した。

バイナリ互換性: 異機種間でバイナリコードレベルで互換性があること。即ち、同じプログラムが Windows95, NT, Macintosh, いろいろな種類の Unix 等でも動作すること。
相互運用性: 異機種間で相互に接続して分散オブジェクト機能を利用可能なこと。
ネットワーク可搬性と移動性: 必要なクラスはネットワークを介して自動的に移動して利用可能なこと。オブジェクト(インスタンス)も転送可能なこと。
WWW との統合: WWW クライアントや WWW サーバ中で動作すること。

WWW ブラウザ内で動作するアプレットに ORB 機能を提供しようとすると、HORB のシステム自身がネットワークを介して WWW ブラウザ内に転送される必要がある。従って、HORB のシステム自身もネットワーク可搬であるために、上記の要件を満たしていなければならない。HORB がダウンロードされている間、WWW の利用者は待たされる。気にならない程度のダウンロード時間であるには、HORB システムは非常に小さく単純に実現されていなければならない。

一方、HORB はプログラマにとっては Java の分散拡張である。HORB のプログラムは、Java の文法に対し完全な互換性を有すると共に、既存の Java 開発環境との互換性と Java 実行環境との互換性が必要である。利用者にとって HORB は Java のごとく見えなければならない。Java 実行系にとっても HORB が生成したプログラムは Java が生成したプログラムのごとく見えなければならない。

4 HORB の分散オブジェクト実現方式

本章では前章で述べた要件を満たす分散オブジェクトの実現方式について述べる。

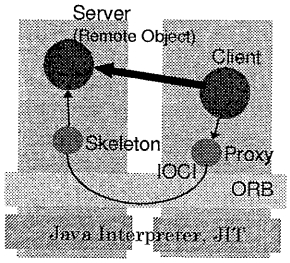

図 1 Proxy による分散オブジェクトの実現

HORB では分散オブジェクトの実現法として、Proxy オブジェクト[2]による方式をとる。利用者のオブジェクトは、リモートオブジェクトの代理としてローカルな Proxy オブジェクトをアクセスする。Proxy オブジェクトは通信処理を行う ORB を介してリモートオブジェクトを利用することにより、リモートオブジェクトと同等のメソッドをクライアントに提供する。
前述の要件のうち、バイナリ互換性、可搬性、WWW との統合の 3 点は Java が有する機能であるため、Proxy オブジェクトと ORB をすべて Java で記述すれば要件は満たされる。他の要件も機種依存性がないように ORB を記述すれば問題はない。WWW ブラウザの内部に HORB の

ORB を自動的にダウンロードすることも可能になる。

だが、果たして(C++を簡単にしたような)Java によって Java 自身の ORB を記述することが可能であろうか？ C++と比較して Java には若干のメタ機能が追加されていたために、制限はあるものの(オブジェクト転送において private 変数が転送できないなど)、記述は可能であった。

4.1 HORB の全体構成

HORB は HORBC コンパイラと、ORB を含むランタイムクラス群から構成される。開発の際には Java 開発キット(JDK)が、実行の際には各プラットホームで動作する Java VM と標準クラスライブラリが必要である。

HORBC コンパイラ
: リモートオブジェクトとしてアクセスされるクラス、及びオブジェクト転送の対象となるクラスは HORBC コンパイラによってコンパイルされなければならない。HORBC はクラスのソースファイルから、そのクラスの class ファイル、そのクラスに対応する Proxy クラス及び Skeleton クラスを生成する。Proxy クラスはクライアント側のスタブコードを、Skeleton クラスはサーバ側のスタブコードを含む。たとえば、HORBC コンパイラはリスト 1の Server.java から Server.class、Server_Proxy.class、Server_Skeleton.class を生成する。HORBC コンパイラは Server のクラス定義から Proxy と Skeleton のソースファイルを生成するフロントエンド部と、ソースファイルから class ファイルへのコンパイルを行うバックエンド部から構成される。フロントエンド部は class ファイルを解析してメソッドシグニチャを得て、スタブコードを生成する。バックエンド部には javac コンパイラ(標準コンパイラ)を利用している。

ランタイムクラス群
: ランタイムクラス群は ORB、分散システム構築用フレームワークから成る。これらはクライアントからのみ使用されるクライアント ORB の部分、サーバとして動作する ORB の部分、そして両者に共通の部分から構成される。

4.2 リモートオブジェクトの生成と接続

HORB のリモートオブジェクトは世界で一意に決まる名前を持つ。オブジェクト名は URL(Universal Resource Locator)に基づく表記により、どのホストのどのポートで動作している HORB サーバ上のオブジェクトであるかを指定する。例 horb://host:998/myObject。リモートオブジェクトを生成する際に、URL にオブジェクト名が含まれない場合は、新しいリモートオブジェクトが生成され、含まれる場合は既存のリモートオブジェクトへの接続となる。

リスト 1の Server クラスをリモートオブジェクトとして生成し play()メソッドを呼び出す処理はリスト 2のごとく簡単な記述となる。以下に、リスト 2を実行した際に起きる処理の流れの概略について述べる。

```
// Server.java
class Server {
  Video play(String title) {...}
}
```

リスト1　リモートオブジェクト

```
Server_Proxy s = new Server_Proxy(url);
video = s.play("Apollo 13");
```

リスト 2 リモートオブジェクトの生成と呼び出し

Proxy オブジェクトと Skeleton オブジェクトの生成と通信路の確立
: Server_Proxy オブジェクトを生成すると、ORB の内部に通信処理を行う IOCI オブジェクトが生成される。IOCI オブジェクトは URL で指定するサーバの ORB に通信コネクションを開設する。サーバ側では、URL にオブジェクト名が指定されていない場合、Server_Skeleton オブジェクトを生成し、オブジェクトテーブルに登録される。同時にクライアントに対応するスレッドを生成し、スレッドテーブルに登録する。以後、そのクライアントからのアクセスはこのスレッドによって行われる。オブジェクト名が指定されている場合、オブジェクトテーブルを検索し、指定の名前にマッチするオブジェクトにスレッドを生成する。

Server オブジェクトの生成
: リスト 2では Server のコンストラクタは呼び出されていないが、Server オブジェクトを生成する際に引数付きのコンストラクタを明示的に呼び出したい場合もある。両方のプログラミング方法をサポートするために、Server オブジェクトそのものは遅延生成される。プログラマが Server のコンストラクタを明示的に呼び出した場合、Server オブジェク

トは Server_Skeleton 中のコンストラクタのスタブ内で生成される。コンストラクタ用のスタブが呼び出されずに、通常のメソッドが呼び出された場合は、Server オブジェクトはデフォルトコンストラクタによって生成される。

Server オブジェクトの消滅

オブジェクトテーブルはオブジェクトへのリモート参照の数を管理している。自動生成されたリモートオブジェクトは、クライアントからの接続がなくなるとオブジェクトテーブルから削除される。オブジェクトの実体も消滅する。

4.3 リモートメソッドの実行方式

リモートメソッドの実行における引数送信の処理方式としては、メソッドシグニチャにある引数リストを実行時に順次解釈しながら送る方式ではなく、予め引数を送るコードをコンパイルしておく方式をとった。これは高速化のためと、Java では実行時に次元数を指定して配列を生成することができないことが理由である。

```
public synchronized Video play(java.lang.String arg0)  {
    IOCI io = _getIOCI();
    try {
      io.selectMethod(classNo, 1000, 1);
      io.sendString(arg0);
      io.request();
      short stat = io.recvStatus();
      Video retValue;
      retValue=
          (Video)io.recvObject(Video_Proxy.targetClass,
          Video_Proxy.proxyClass, new Goldberg());
      return retValue;
    } catch (IOException ioe) {
      throw new NetException(ioe);
    }
  }
```

リスト3 Server_Proxy のスタブコード

クライアントが play を呼び出すと、実際には play のスタブコードである Server_Proxy.play が実行される。リスト 3に Server_Proxy.play を示す。 Server_ Proxy.play はストリームである IOCI オブジェクトに対して、メソッド番号(本例では 1000)と引数を順に押し込み、request()によって送出する。

サーバ側ではディスパッチャがメソッド番号に対応する Server_ Skeleton 中のスタブコードを呼び出す。このサーバ側のスタブは引数を受信した後、Server のリモートオブジェクトの play を呼び出す。メソッドの返り値や例外は、呼び出しと逆の方向でクライアントに返される。リスト 3 では Video 型の返り値を recvObject()によって受け取り、クライアントに返している。

4.4 リモートメソッドのクラス継承

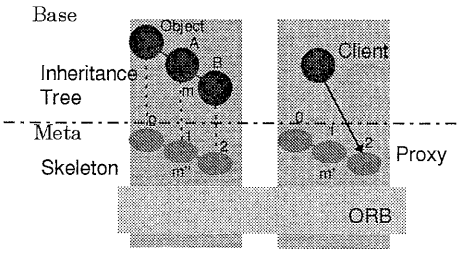

図2 リモートメソッド実行におけるクラス継承の実現

Java のクラス継承は単一継承である。リモートメソッドの継承の実現にあたっては、リモートメソッドの継承を、高速に、プログラマにトランスペアレントに、単一継承という限られた記述力によって実現する必要がある。また、継承木中でのメソッドの探索が必要になるため、これを高速に行うことも重要である。

Java ではクラスの継承関係は静的であるため、上位のクラスが有するメソッドはコンパイル時に明らかになっている。従って、コンパイル時に継承するすべてのメソッドのスタブを生成しておくことができる。しかし、クラスはネットワーク中の様々の場所で開発されるために、あるクラスを開発している者が知らないうちに、上位クラスにおいて継承関係やメソッド名に変更があるかもしれない。従って、継承関係のバインドは実行時に行う方針とする。クラス C が他のクラスを継承している場合でも、C_Proxy はクラス C のメソッドのためのスタブのみを含み、継承している上位クラスのスタブは含まない。

ユーザプログラム(ベースレベル)のリモートメソッド実行におけるクラス継承は、ベースレベルのクラス継承木に対応する Proxy の継承木と Skeleton の継承木をメタレベルに作成することによって実現する。ベースレベルにおいて、Object、A、B の順で継承するクラスがあるとする[3]。クライアントがリモートオブジェクトとして B のインスタンス b を生成すると、図 2のように Object_Proxy, A_Proxy を継承する B_Proxy のインスタンス b_Proxy がクライアント側に、Object_Skeleton, A_Skeleton を継承する B_Skeleton のインスタンス b_Skeleton がサーバ側に生成される。

[3] Java ではすべてのオブジェクトは Object のサブクラスである。

クライアントがb_Proxyを介してAに属するメソッドmを呼び出す場合、A_Proxyに属するスタブm'がA_Skeletonのサーバ側スタブm"を呼び出し、m"がAのmを実行しなければならない。クライアントがb_Proxy.m'を実行すると、B_Proxy.m'が発見されて実行される。これはJava実行系による処理であるため高速である。m'はm"のメソッド番号をb_Skeletonのディスパッチャに送る。メソッド番号はクラス内でローカルな情報であるため、これだけではm"がb_Skeletonの継承木中のどのクラスのメソッドであるかはわからない。継承木中のクラスの位置関係は、ProxyからSkeletonに対して、メソッド番号に加えてクラス番号を通知することによって指示する。クラス番号は、Objectを0とし、Objectを継承するクラスを1、そのクラスを継承するクラスを2...とする順序数として定義する。先述したように継承関係は動的に決定するという前提を置くため、クラス番号は実行時に割り振る。この処理はProxyオブジェクトとSkeletonオブジェクトの生成時にそれぞれ行われる。リスト4にSkeletonのクラス番号の設定を行うコードの例を示す。

```
Class B_Skeleton extends A_Skeleton {
  protected short classNo;
  B_Skeleton() { classNo = super.classNo+1; }
  boolean dispatch(IOCI ioci, short classNo, short methodNo) {
    if (classNo < this.classNo)
      return super.dispatch(ioci, classNo, methodNo);
    switch (methodNo) {
    case 1000: stub1000(ioci); break;
    case 1001: stub1001(ioci); break;
    }
  private void stub1000(IOCI ioci) {
    .....
```

リスト 4　Skeleton側での継承の実現

クラス番号とメソッド番号を受け取ったサーバ側ORBはb_Skeletonのディスパッチャを呼び出す。リスト4に示すように、ディスパッチャは継承木をクラス番号が一致するまで溯り、該当クラス内でメソッド番号に対応するサーバ側スタブを実行する。

以上のように、リモートメソッドのクラス継承の実現において、クライアント側ではProxyの継承によりメソッド探索はJava自身によって高速に行われ、サーバ側では上記の単純な仕組みによってメソッドの探索が高速に行われる。

4.5 分散透明性の実現

リモートオブジェクトとローカルオブジェクトを区別なく呼び出すことができることは分散オブジェクト指向言語の要件のひとつである。これまで述べてきたProxyによる実現では、ローカルオブジェクトの型Cと、リモートオブジェクトの型C_Proxyが異なるため、分散透明性が損なわれている。たとえば、C型の引数をとるメソッドにC_Proxyを渡すことはできない。

分散透明性を実現するひとつの方式は、C_ProxyがCを継承する方式である。C_Proxyの内部でローカルかリモートかを判断してリモートならばORBをアクセスし、ローカルならばスーパークラスCのメソッドを呼び出せばよい。しかし、単一継承のJavaではC_ProxyがCと同時にB_Proxyのサブクラスになることはできない。この方式をとると、前章で述べた継承の実現法をとることができないため、継承をもっと複雑で遅い方法で実現しなければならなくなる。

HORBでは、分散透明性を実現するために、interfaceを用いる。interfaceはクラスが実装すべきメソッドを定義するJavaの言語要素である。クラスCがinterface Iを継承するとCをIの型としても扱うことができる。

HORBCコンパイラはinterfaceの定義IからIを継承するI_ProxyとI_Skeletonを生成する。プログラマはIを実現するクラスをI_Implとして実装する。クライアントがI_Proxyを生成すると、クラスI_Implのインスタンスがリモートオブジェクトとしてサーバ側に生成される。一方、ローカルオブジェクトとしてI_Implを生成することも可能である。両者とも、I型としてアクセスすることができるため、ローカルオブジェクトとリモートオブジェクトはプログラム上区別なくアクセスすることができる。

5 おわりに

ネットワーク可搬なJava用のORBの実現について、リモートメソッド実行の面から述べた。リモートメソッド実行を行うためにWWWブラウザにダウンロードされるHORBの大きさは最小２４ＫＢと非常に小さい。

JavaでJavaのORBを記述するにあたって、オブジェクト転送も重要な一面であり、こちらの方がJavaのメタ機能に大きく依存している。別の機会に発表したい。

参考文献

[1] 平野、HORB: ワールドプログラミングのための分散並列オブジェクト指向言語, WOOC'96, 1996

[2] Shapiro, M., Structure and Encapsulation in distributed Systems: The Proxy Principle, ICDCS, pp.198-205, 1986