特集

3. 分散オブジェクト技術

平野　聡
電子技術総合研究所／
HORB Open &
Ring Server Project

分散処理の動機

ネットワーク上にある複数の計算機にまたがって「1つのJavaプログラム」を動作させる技術がJava用分散オブジェクト技術（Java DOT）です．本稿では，現在広く普及しつつあるJava DOTの全体像を捉えるために，Java DOTによって分散システムの記述がいかに簡単になるか，Java DOTの基本原理，そして，Java DOTの現状とその動向について述べます．

解説を具体的にするために，ひとりの脳科学の実験家に登場してもらいましょう．彼は情報処理に永くたずさわっており現在は脳機能の解明を仕事としています．ここ数カ月，脳のサンプルを薄くスライスし，記憶の痕跡を示す特殊な印が現れているかを顕微鏡で観察することが日課です．彼は10年前からMacを愛用していますが，彼の顕微鏡はWindowsのパソコンでコントロールされています（**図-1**）．顕微鏡から取り込んだ画像データの解析には計算センターのスーパーコンを使います．現在は顕微鏡のワンショットごとにいちいちファイル転送をしているため，手間がかかります．もっとインタラクティブに，「Windowsの計算機で取り込んだ脳の画像データをスーパーコンピュータで処理し，その結果を即座に手元のMacで3次元画像をくるくると回して表示する」にはどうしたらよいでしょうか．実験家は簡単なプログラムを書くことはできますので，これを「1つの小さなプログラム」として書くことが希望です．

この例は小さな例ですが，生産ライン管理システム，在庫管理システム，受発注システムなど，ネットワークで結合されたたくさんの計算機上で，一連の処理を行うプログラムを協調・分散して動作させる分散処理は私たちに身近なものです．計算機にはWindows，Mac，Unix，メインフレームなど多様な機種があります．これらの異機種を混在して相互運用することが可能な，分散言語や分散OSを実現することが，情報処理 技術開発の大きな目標でした．Java以前の分散処理技術や参考文献についてはタネンバウムの著書[1]を参照してください．この本にはいろいろな分散処理技術が紹介されていますが，残念ながら，実験家の環境で共通して利用できるものは1つもありません．Javaの出現前の時点で利用可能なプログラミング手法を使うならば，おそらくは，Windows，スーパーコン，Macそれぞれのマシンのために1つずつプログラムを書いて，プログラム間はファイル転送を行うか，もう少しプログラム能力があれば，それぞれのプログラムをTCP/IPのソケットで接続してデータ転送を行うようなプログラムを書いていたでしょう．

Java用分散オブジェクト技術の概要

1995年，Javaの出現によって，計算機の機種に依存せずにプログラムを書くことが可能になりました．Write Once, Run Anywhere. どの機種の上でも，同じプログラムが動作します．では，実験家もJavaを使えば，彼の分散システムを「1つの小さなプログラム」として簡単に書くことができるでしょうか？　否，依然として計算機ごとにプログラムを書いて，ソケットでデータ転送を行う必要があります．なぜなら，Javaのプログラムは依然として1つの計算機の1つのプロセスの中でしか動かないからです．これを克服するために，Java用のDOTが一斉に開発され，またたく間に実際のシステム開発に利用されるようになりました．

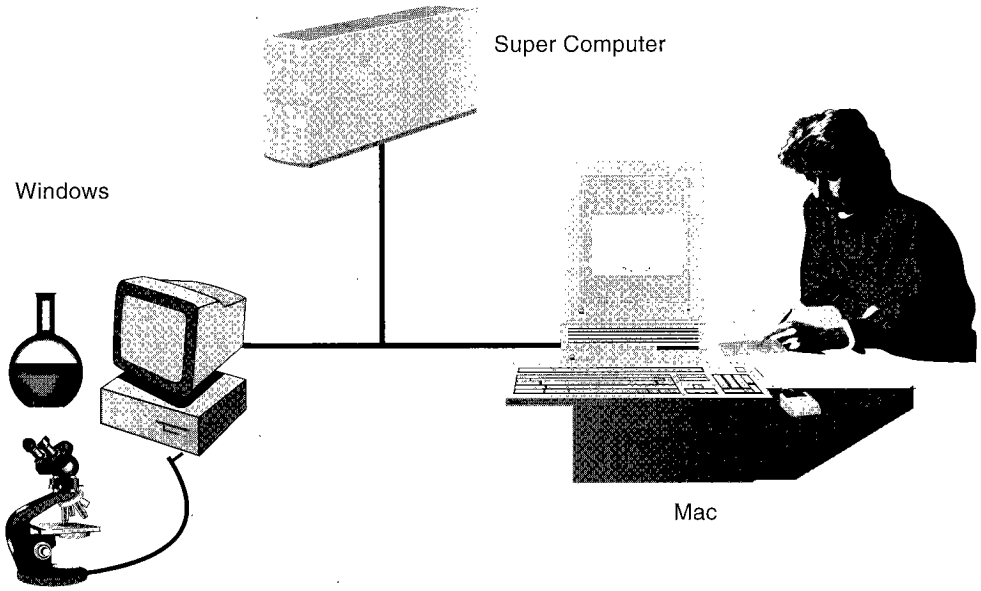

図-1　ある実験家の実験システム

Javaには大域変数やポインタがない，メモリ管理もガベジコレクションによって自動化されているといった，CやC++で取り扱いが難しかった問題がなくなっているために分散処理に向いています．そのため，長い間研究段階に留まっていた分散処理技術を実際に使える技術として実現することが比較的容易でした．Javaはもはやあらゆる機種の上で動作します．Java DOTを使えば，実験家は望み通りのプログラムを書くことができます．

一分散オブジェクトの記述

では，実際に実験家のシステムを記述してみることにしましょう．同じJavaを用いてもDOTの種類によって記述の方法はまったく異なります．ここでは，記述が簡潔なHORB[2)~3)]を用います．まず，計算機間でやりとりされる画像データをオブジェクトにしてみましょう．**リスト1**は画像データであるPictureクラスです．画像データを格納する変数と，画像処理，表示を行うメソッドを持っています．

顕微鏡はWindowsに接続されており，顕微鏡画像をイメージとして取り込むことができます．スーパーコンは画像データオブジェクトを高速に画像解析します．これらのオブジェクトを**リスト2**に示します．実験家のMacの上で動作するクライアント側のコードの主要な部分が**リスト3**です．mainから始まるこのプログラムは，win1.etl.go.jpというWindowsマシン上に顕微鏡オブジェクトを作って，脳のスライスの顕微鏡画像を取り込みます．次にスーパーコンであるsuper.etl.go.jpにスーパーコンオブジェクトを作り，脳画像を送り込んで画像処理を行います．処理が終了したらディスプレイに表示して終了します．このプログラムはそれぞれ数行からなる合計4つのクラスから構成されており，「小さな1つのプログラム」という条件に合格します．実験家はわずか1時間で完成したこのプログラムによって，毎日数時間の実験時間を節約できるようになりました．

一分散オブジェクトの基本原理

リスト3のプログラムは，みたところ通常のJavaプログラムとほとんど違いがありません．違うのは，MicroScopeオブジェクトをnewによって生成する代

```
class Picture {
  Image image;          // 画像データ
  void imageProcessing () {...} // 画像解析
  void display () {...}       // 結果の画像の表示
}
```

リスト1　画像データオブジェクト

```
class MicroScope {
  Picture capture (Picture data) {
    // 顕微鏡から画像を取り込む
    data.image = 取り込み ();
    return data;
  }
}
```

```
class SuperCom {
  Picture computation (Picture data) {
    data.imageProcessing ();
    return data;
  }
}
```

リスト2　顕微鏡オブジェクトとスーパーコンオブジェクト

わりに，MicroScope_Proxyオブジェクトをnewしていることです．このProxyオブジェクトは，リモートのWindowsマシンの上の実際のMicroScopeオブジェクトの代理（proxy）として，あたかもリモートオブジェクトのごとく動作します（図-2）．Proxyの内部には，リモートメソッドの呼び出しをネットワーク処理に変換するスタブコードが含まれています[4]．すべてのJava DOTはこのアーキテクチャを用いています．

プログラマが記述するのは，リモートオブジェクトとそれをアクセスするクライアントオブジェクトです．コンパイラがネットワーク処理を行うProxyとSkeletonクラスを生成します．ProxyとSkeletonは，分散オブジェクトの通信路であるORBを介してリモートメソッドの実行を行います．このプログラムで中心的な役割を果たしているのがリスト1の画像データオブジェクトです．このオブジェクトはMac上で生成された後，Windowsに送られたり，スーパーコンに送られたりして仕事を進めていきます．このオブジェクト転送の機能は，分散システムの拡張性や再利用性を著しく高めます．たとえば，新しい画像解析アルゴリズムを使用したい場合は，画像データオブジェクトを継承するサブクラスに新アルゴリズムを実装すれば，他のクラスは一切変更する必要がありません．また，画像データオブジェクト自身に次の処理を決する処理を行わせればモバイルエージェントとなり，システムは自律的な動作を行うようになります．このようなオブジェクト転送機能は，複雑な構造のオブジェクトを異なる計算機にそのままの形で転送するオブジェクト・シリアライゼーションに基づいています．オブジェクト・シリアライゼーションはJDK1.1からJavaの標準機能になりましたが，すべてのJava DOTがオブジェクト転送の機能を備えているわけではありません．

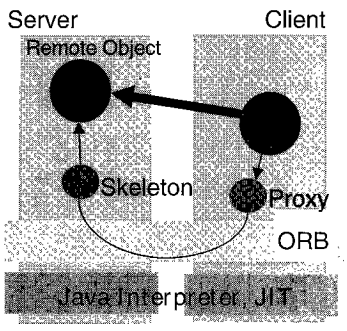

図-2　分散オブジェクト技術のアーキテクチャ

```
class Client {
    public static void main (String argv[]) {
        Picture picture = new Picture ();
        // 顕微鏡から画像を取り込んで
        MicroScope_Proxy microScope = new MicroScope_Proxy
("horb://win1.etl.go.jp/");
        picture = microScope.capture (picture);
        // それをスーパーコンに送って画像処理
        SuperCom_Proxy superCom = new SuperCom_Proxy
("horb://super.etl.go.jp");
        picture = SuperCom.computation (picture);
        // それを表示
        picture.display ();
    }
}
```

リスト3　手元の計算機で動くコード

―Java用分散オブジェクト技術の利点

以上のJava DOTの利点をまとめると以下のようになります．

- 分散オブジェクト技術がネットワークコンピューティングのフレームワークとして機能します．プログラマがデータ転送やセキュリティを記述する必要がありません．プログラマはアプリケーションの記述に専念できます．
- 通信処理でバグが出ないため，開発期間が大幅に短縮されます．
- アプリケーションの拡張性や再利用性が高まります．
- 異機種が混在するネットワーク上で相互運用可能であり，プログラムも機種にかかわらずポータブルです．
- コンパイラやORB自身がJavaで記述されているDOTでは，ORB自身も機種を選びません．

分散オブジェクトの現在と将来

Java DOTは急激な勢いで進化しています．本章ではJava DOTの現状と進化の方向について概観します．

―分散オブジェクト技術の分類

現在のJava DOTを周辺技術も含めた分散オブジェクトフレームワークとして分類すると以下の3種類に分けることができます．

(1) OMG CORBA

OMG（Object Management Group）は分散オブジェクト技術の確立を目的に，1989年に創立された世界最大の技術コンソーシアムです．CORBAのAはアーキテクチャを表しており，CORBAの仕様はOMG各社の分散オブジェクト製品の共通のアーキテクチャを意味しています．現在公開されているCORBA 2.1は，ORBコア[5]，共通サービス[6]，共通ファシリ

ティ[7]から構成されています．CORBAはアーキテクチャですから，その実装は各ベンダに依存しており，CORBA製品は準拠のレベル，実現する機能の範囲，独自拡張の度合い，相互運用性などが非常に多岐に渡っています．文献10）によれば，CORBA準拠のORBは製品，フリーなど合わせて40種に達するそうです．ベンダが違えばCORBA準拠製品同士でも相互運用性がない状態が永く続いていましたが，CORBA2から仕様化されたIIOP（Internet Inter-ORB Protocol）によって，相互の通信ができるようになりました．1996, 7年はJava上にIIOPを実装したCORBA製品のリリースが相次ぎました．JDK1.2にはJavaIDLが標準で装備されています．CORBAはプログラミング言語に依存せずにC，C++，COBOL，Javaなどの間で相互にオブジェクト呼び出しができるよう，リモートオブジェクトの定義をOMG IDL（Interface Definition Langu-age）言語で行います．IDLコンパイラによってIDLからProxyが生成され，通常のプログラミング言語で書いた実装のコードと共に実行します．OMG IDL言語はJavaが生まれる前に定義された言語ですから，そのオブジェクトモデルはJavaのオブジェクトモデルと異なっており，プログラミングを難しくしている面があります．オブジェクト転送は，現在，参照渡しのみが利用可能ですが，現在策定中のCORBA3によって，オブジェクトの値渡しができるようになる予定です[18]．OMGはビジネスオブジェクト，モーバイルエージェント，電子商取引など広範囲に渡って仕様を策定中です．これらの仕様がカバーする範囲の多くはJava APIと重なっており，システム設計時にどちらを選ぶかが問題となっています．

(2) Windows環境を前提とするMicrosoftのDCOM

Microsoftのオブジェクト技術であるActiveXコンポーネントは，現在オブジェクト市場のほとんどを占めており，Windowsのユーザは毎日その恩恵を被っています．ActiveXのオブジェクトモデルはCOM（Common Object Model）と呼ばれ，やはり言語に依存しません．オブジェクトはMicrosoft ODL（Object Definition Langu-age）言語によって記述し，通常のプログラミング言語で書いた実装のコードと共に実行するのはCORBAと同様です．COMを分散化する技術であるDCOM（Distributed COM）[8]は，膨大なCOM資産を無変更でリモート実行することを可能としています．Microsoftは次期Windows NTに向けてWindowsの分散アーキテクチャDNA[9]を策定しています．DNAはオブジェクト・ディレクトリ，トランザクション，電子商取引などを含み，JavaからWindowsのAPIでシステム開発のすべてが記述できるようになるとされています．

(3) Java環境を前提とするJava専用分散オブジェクト技術

Javaの豊富なAPIの利用を前提とするJava専用DOTを用いれば，CORBAやWindowsに依存しない分散オブジェクトフレームワークの構築が可能です．IDLを書くことなくJavaだけでリモートオブジェクトの記述が可能なので，IDLやODLにある技術的な制約を受けません．自然なオブジェクトの記述が可能となるため，プログラマの負担は大幅に軽減されます．CORBAやDCOMにはないオブジェクト転送も可能です．新たにシステムを開発する場合には，有力な選択肢となっています．SunのRMI[11]はJDK1.1から標準装備となったJava DOTで，最も基本的な分散オブジェクト機能を提供します．Microsoftが同社のWWWブラウザにRMIなどのAPIを実装しなかったため，両社の間で訴訟となっています．HORB[2),3)]は，筆者によって1995年に公開された世界で最初のJava DOTです．相互運用性や記述性など，永かった分散処理の夢を初めて実現しました．記述が簡潔で性能がよいこと，また，広く普及しているJDK1.0ベースのWWWブラウザでオブジェクト転送が可能なのはHORBだけであることから，Javaの初期から現在まで広く使われています．ObjectSpaceのVoyager[12]はJavaのReflectionを有効に使ったモーバイルエージェント機能を有するJava DOTで，高い人気があります．Spaceというオブジェクトグループに対してグループコミュニケーションをする機能が特徴です．

一分散オブジェクト技術発展の動向

ここではJava DOTの動向のうち，主要と思われる3つについて述べます．

(1) 分散オブジェクトフレームワークからアプリケーションフレームワークへ

ビジネスシステムのアーキテクチャは，サーバ上のデータベースをクライアントからアクセスする2層構造から，ビジネスロジックをアプリケーションサーバ内に分離した3層構造へと変わりつつあります．アプリケーションサーバとデータベース間はJDBCで，アプリケーションサーバとクライアント間はDOTで接続します[13]．さらに，業務ごとのビジネスロジックを実装したコンポーネントを，アプリケーションサーバ上にあらかじめ用意している，アプリケーションフレームワークが形を現してきました．IBMのSan Francisco Frameworks[14]はその一例です．アプリケーションフレームワークを用いると，業務内容にあったビジネス・コンポーネントを選択し，それとクライアントのGUIをカスタマイズすることがシステム開発となります．システムを1から作成し，人月単位で報酬を得ているシステム開発のスタイルが大きく変わるでしょう．

(2) 共通バスとしてのIIOPの採用

CORBA3になるとIIOPがオブジェクト転送機能を

備えるため，Java専用DOTも機能をあまり損なうことなくIIOPを使うことができるようになります．すると，IIOPを介して既存システムのCORBAと接続することが可能となります．そのため，VoyagerはすでにIIOPを実装していますし，RMIとHORBもIIOPの実装を表明しています．

(3) オブジェクトモデルのJava化

システム開発の記述言語にJavaが占める割合が高くなるため，CORBAやCOMもJavaへの対応を急いでいます．CORBAでは，IDLを書くことなしに，Javaのインタフェース定義からIDLを生成する仕様が検討されています[17]．VisiGenicのVisiBroker[15]はすでにjava2idlコマンドによってそのような機能を実現しています．Microsoft COMの次世代版のCOM+[16]では，ODLなしでJavaやC++などの言語で直接リモートオブジェクトを定義できるようになります．また，COMは実装の継承をサポートしていませんが，COM+では実装の継承は単一継承，インタフェースの継承は多重継承とJavaのオブジェクトモデルに適合されています．COM+用C++もimportやimplementsなどJava風のプリミティブが追加されます．

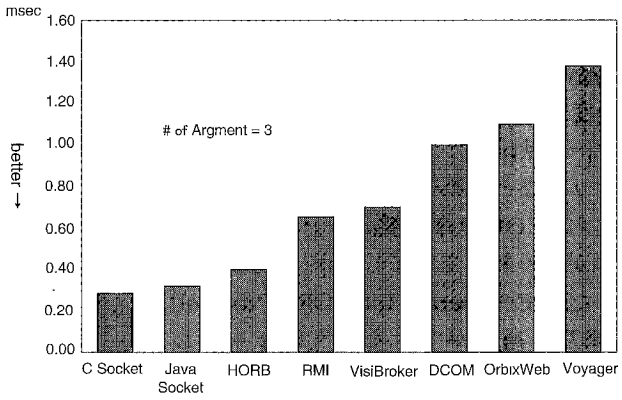

図-3 リモートメソッド呼び出しのベンチマーク

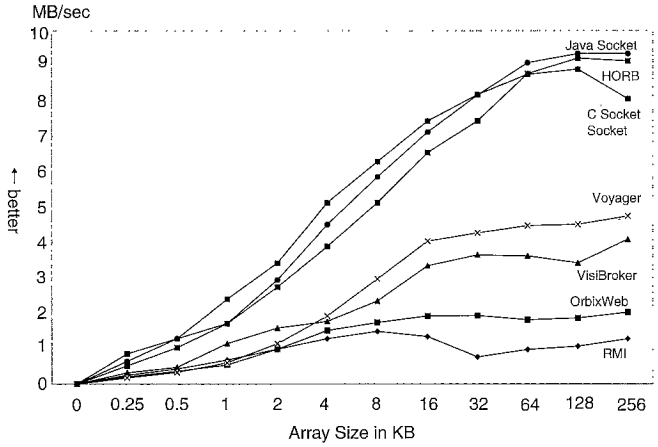

図-4 データ転送速度のベンチマーク

ー現在の処理系間での性能比較

現在のJava DOTは性能が低いものが多く，システム開発上の問題点の1つとなっています．現状を調べるために代表的なJava DOTのベンチマークを行いましたので紹介します[3]．**図-3**にリモートメソッド呼び出しのベンチマーク結果を示します．リモートメソッドはint foo (int, int, int) です．**図-4**はvoid foo (byte[]) をリモート実行した際のデータ転送速度の測定結果です．測定環境は100Base-TXで接続された2台のPentiumII 266MHz Windows NTを用いました．Java実行系にはいずれもSunとMicrosoftのJITを使用しています．ソケットと同程度の速度を実現しているのはHORBだけでした．JavaはCと比べてまだ実行速度が遅く，Cとソケットのプログラムから Java DOTに移行するには十分な注意が必要です．DOTのベンダは性能向上の努力が求められています．

参考文献

1) Tanenbaum, A.S.: Distributed Operating Systems, Prentice Hall (1995). 水野他訳, 分散オペレーティングシステム, プレンティスホール (1996).
2) Hirano, S.: HORB: Distributed Execution of Java Programs, Worldwide Computing and Its Applications, Springer Lecture Notes in Computer Science 1274, pp.29-42 (1997).
3) Hirano, S., Yasu, Y. and Igarashi, H.: Performance Evaluation of Popular Distributed Object Technologies for Java, Proc. of ACM 1998 Workshop on High-Performance Network Computing (1998), http://www.cs.ucsb.edu/conferences/java98
4) Shapiro, M.: Structure and Encapsu-lation in distributed Systems: The Proxy Principle, ICDCS, pp.198-205 (1986).
5) Object Management Group, The Common Object Request Broker: Architecture and Specification 2.1 (1997), http://www.omg.org
6) OMG, Common Services Architecture (1995).
7) OMG, Common Facilities Architecture (1995).
8) Brown, N. and Kindel, C.: Distributed Component Object Model Protocol－DCOM/1.0, Internet Draft (1996).
9) http://www.microsoft.com/sitebuilder/dna/
10) 鈴木: CORBA技術の全貌と今後の動向, Dr.Dobb's Journal Japan, pp.28-43 (Aug. 1997).
11) JavaSoft, Java Remote Method Invocation Specification, 1997, http://java.sun.com/products/jdk/1.1/docs/guide/rmi/spec/rmiTOC.doc.html
12) Glass, G.: ObjectSpace Voyager－the Agent ORB for Java, Worldwide Computing and Its Applications (1998), http://ci.etl.go.jp/wwca98/
13) Duan, N.: Distributed Database Access in a Corporate Environment Using Java, 5th Int. WWW Conference (1996).
14) http://www.developer.ibm.com:8080/welcome/java/sfindex.html
15) http://www.visigenic.com
16) Kirtland, M.: Object-Oriented Software Development Made Simple with COM+ Runtime Services, Microsoft Systems Journal, Vol.12, No.11 (1997).
17) IBM, et al.: Java to IDL Mapping, OMG TC Document orbos/98-01-07.
18) BEA, et al.: Objects by Value, OMG TC Document orbos/98-01-01.

（平成10年2月23日受付）