プログラミング 10－7
（1996．11．1）

分散仮想マシンを用いたオブジェクト指向プログラミング環境

地引昌弘† 芦原栄登士†† 山下雄三††
上田哲郎† 大木敦雄†† 久野靖†

従来より、分散オブジェクト環境におけるオブジェクトアクセスは、ユーザからのアクセスに対しては透過性を提供しているものの、オブジェクトの実装者に対しては十分な透過性を提供しているとは言えない。また、オブジェクトアクセスに関しても、分散アプリケーションにおけるサーバクライアントモデルと比べて、自由度の高いマイグレーション機能が提供されているとは言えない。本稿では、これらを解決するために、分散仮想マシンを導入することにより、複数レベルの透過性やマイグレーション機能を提供することを目指す。分散仮想マシンは、互いに通信し合うことで仮想空間を構築し、これをオブジェクト空間とすることで、これらの機能を実現する。

Distributed Virtual Machine-Based Object Oriented Programing Environment

Masahiro Jibiki , Eitoshi Ashihara , Yuzo Yamashita ,
Tetsuro Ueda , Atsuo Ohki and Yasushi Kuno

Object access in current distributed object environments usually provide user level transparency, but they don't sufficiently provide implementation level one. They either do not provide unrestricted object migration faculty also. In this paper, we propose "Ditributed Virtual Machines" communicating each other to construct single distributed virtual space, and we consider the space an object space. Moreover we define an original language and its translater to construct an object. We aim to achieve above-mentioned object access faculties in the object space by the ditributed virtual machine and an executable environment of the language.

1. はじめに

分散環境において透過的なオブジェクトアクセスを実現するために、従来より多くのシステムが提案されてきた。その代表的なものとして、OMGが提案しているCORBA[1]があるが、最近では、Java[2]アーキテクチャ上で稼働するRMI[3], HORB[4]なども注目されている。

本稿では、これら従来の分散オブジェクトシステムにおける

- 一面的な透過性
- 静的なクライアントサーバモデル

という問題に着目して、これらを解決することにより、オブジェクトアクセスに対する複数レベルの透過性を提供し、また動的なオブジェクトアクセスを実現するために、分散仮想マシンを導入したオブジェクト環境およびそのプログラミングモデルを提案する。

以下、第2節では、従来の分散環境における問題点について述べる。第3節では、分散仮想マシンについて説明し、第4節ではその実装について述べる。最後に、第5節で議論とまとめを行なう。

2. 従来の分散オブジェクト環境

本節では、従来の分散オブジェクト環境におけるリモートオブジェクトアクセスに付随する、2つの問題に関して述べる。

2.1 一面的な透過性

分散環境では、透過的なオブジェクトアクセスを提供するために、多くの努力が払われている。しかしながら、多くの場合、オブジェクトアクセスは、ユーザ(オブジェクトサービスの享受者)に対して透過的であるものの、オブジェクト実装者(オブジェクトサービスの提供者)に対して透過的であるわけではない。

† 筑波大学 企業科学専攻
†† 筑波大学 経営システム科学専攻

例えば、オブジェクト実装者が、他者の実装したオブジェクト**B**を利用して、オブジェクト**A**を実装した場合を考えてみる。ユーザが**A**を実行する場合、**B**の場所を知らなくても、**A**を介して**B**のサービスを享受することが可能である(図1)。

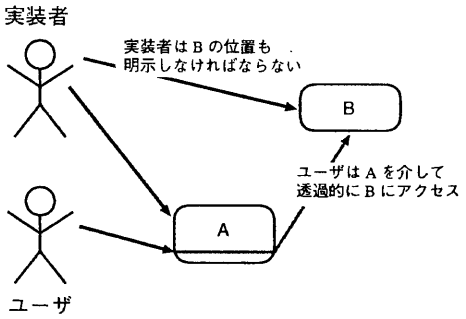

図1 オブジェクトアクセスの透過性

しかしながら実装者は、**A**を実装する際に、**B**の存在する場所を把握して、ユーザに対して隠蔽された**A**から**B**への通信手段を用意する必要がある。この場合、実装者は、自分自身の責任で必要なオブジェクトへのアクセスを提供しなければならず、オブジェクトアクセスは何ら透過的ではない。

例えば**HORB**では、オブジェクトの実装者は図2のように、サーバオブジェクトの位置を**URL**で指定しなければならない。

```
class Client {
 public static void main(String argv[]) {
        .
        .
  host = ''gssm.otsuka.tsukuba.ac.jp'';
  HorbURL url = new HorbURL(host, null);
  Server_Proxy server = new Server_Proxy(url);
        .
        .
  }
}
```

図2 HORBのリモートオブジェクトアクセスの例

つまり、透過的なオブジェクトアクセスが提供されていると言えるのは、最終的なユーザに対してのみであり、実装者は、状況に応じてユーザの立場となる場合が存在するにもかかわらず、透過性が提供されていないと言える。

2.2 静的なクライアントサーバモデル

従来の分散環境では、多くの場合、サービスを提供するサーバオブジェクトがその場所を移動することはない。従って、せっかくオブジェクトという存在でありながら、オブジェクトアクセスは、基本的に分散アプリケーションにおけるサーバクライアントモデルと大差ないものになってしまっている。

最近のJavaやHORBのようなシステムでは、オブジェクトを移動させる機能を備えてはいるものの、実行前のオブジェクトを移動できるのみで、実行中のオブジェクトを移動させ、移動した先で実行を再開させる機能(マイグレーション機能)は有していない。

これは、社内LAN環境のような、構成する要素(マシン, ネットワーク)に比較的差がなく静的な環境においては、欠点がクローズアップされることはないかも知れないが、モバイルマシンまで含めたインターネット環境を考えると、以下のような問題が発生する。

- インターネットは、通信網自体が一様ではないために、通信回線の違いによる通信コストに大きな差が生じる。
- 同様に、プラットフォームも多くの種類が存在しており、オブジェクトを実行するプラットフォームの種類によって、実行コストに大きな差が生じる。

従って、もしオブジェクトが自由に移動可能であれば、通信/実行コストの安いオブジェクトアクセスを提供することが可能となるが、現状のサーバクライアント型アクセスでは、これらを提供することができない。

3. 分散仮想マシン

この節では、前節で示した、分散オブジェクト環境でのオブジェクトアクセスに関する問題を解決する手段として、分散仮想マシンを用いることについて検討する。

3.1 分散仮想マシンの導入

モバイルマシンまで含めたインターネット環境のような、多くの種類のプラットフォームやネットワークが混在する環境で、自由度の高いオブジェクトのマイグレーション機能を実現するには、

1. プラットフォームに依存しないオブジェクト表現
2. プラットフォームの違いを吸収するアーキテクチャ

が必要となる。

今回は、1. を実現するために、新言語 “えーとし”、およびその処理系を開発した。新しい分散オブジェクト記述言語である “えーとし” を用意する理由として、以下を挙げることができる。

- 分散環境におけるオブジェクトアクセスに関して、言語レベルでサポートを行なう。
- オブジェクトの状態を抽象化して扱う機構(抽象状態)[6] をサポートし、分散実行と抽象状態同期との結合の可能性について調べる。

“えーとし” で作成したオブジェクトは、図3のように、Javaと同様1オブジェクト1ファイルの、プラットフォームに依存しない一般的な表現形式であるバイトコードに変換される。

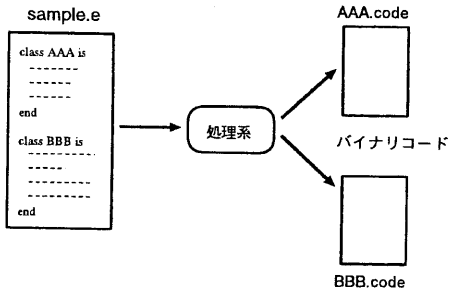

図3 “えーとし” の処理系

また、2. を実現するために、仮想マシンを用意する。仮想マシンは、オブジェクトのバイトコードを各プラットフォーム上で実行する。

オブジェクトアクセスに対する複数レベル(ユーザ／実装者)の透過性を提供するには、

3. オブジェクトの位置管理
4. オブジェクトの探索／アクセス仲介

を行なう機能が必要となる。

これは、分散環境内の各構成要素(プラットフォーム)が、内部でオブジェクトに関する情報を交換し合い、外部(ユーザ／実装者)に対して1個の仮想空間を構築して見せることで実現できる。今回は、前述の仮想マシンを分散仮想マシンに拡張し、これらの分散仮想マシンが近隣の分散仮想マシンと通信し合うことで、必要なオブジェクトの探索／仲介を行なう(図4)。

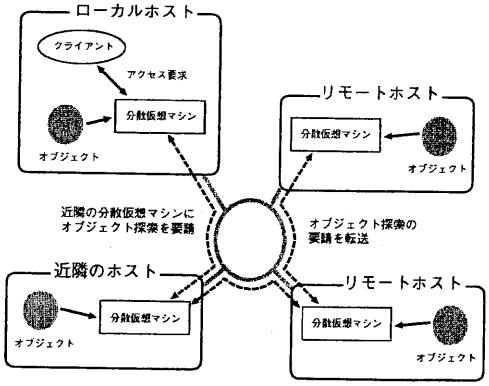

図4 分散仮想マシンによるオブジェクト管理

3.2 オブジェクトの探索

従来は、オブジェクト実装者が、オブジェクトの存在(位置)に関して責任を持つことで、ユーザに対してオブジェクトアクセスの透過性を提供してきたが、今回は、実装者に対しても透過性を提供することを目指し、分散仮想マシンが、オブジェクトの位置に関して責任を持つ。従って、分散仮想マシンは、オブジェクトの探索を行なう必要がある。

オブジェクトの探索を行なうには、各オブジェクトの名前管理を行なう必要がある。オブジェクトの名前管理は、従来より、階層的なOID (オブジェクトID)を用いる方法などが数多く提案されて来た[5]。オブジェクト作成用言語として今回開発した “えーとし” では、Javaと同様、1オブジェクトが1バイトコードファイルに対応するので、そのバイトコードファイルのURLを名前管理の手掛かり☆にすることが可能であり、これと文字列表現などに基づく名前(オブジェクト名)を付随させることでOIDとする。

分散仮想マシンは、オブジェクトアクセスが発生した場合、以下のようなアルゴリズムに基づいて、動的にオブジェクトの探索を行なう。

1. オブジェクト名の全部または一部にマッチングするパターンを提示して☆☆、自分自身が管理している(ローカルな)空間を探索
2. 1. で発見できなかった場合、他の分散仮想マシンに、オブジェクトの探索を依頼

OIDによる一意的なオブジェクトの指定にとどま

☆ “えーとし” では、1つのオブジェクトクラスから、複数のインスタンスを生成することが可能なので、URLをそのままOIDとして流用することはできない。

☆☆ 当然のことではあるが、OID (URL)が直接指定されているような場合は、探索を行なう必要はない。

らず、実行時にオブジェクトを探索する利点としては、以下の理由が挙げられる。

- オブジェクトの指定をOIDで行なうことは、分散環境内における周囲の状況とは関係なく、オブジェクトの位置を一方的に限定してしまうことであり、オブジェクトアクセスにおける効率の低下を招く可能性がある。
- 当然のことではあるが、あらかじめOIDを取得していない場合は、必要なオブジェクトにアクセスすることができない。

オブジェクトの探索は、安全性や効率性を考え、以下のようなポリシーで行なう。

オブジェクト空間の複数レベルのドメインへの分割

オブジェクトの探索のために、全分散仮想マシンが互いに通信し合うこと、つまり、全仮想空間をオブジェクト空間として探索の範囲とすることは、効率の点から考えても事実上不可能である。

従って、全オブジェクト空間の部分集合をドメインとして定義し、各オブジェクトの定義される空間を、このドメインとすることで、オブジェクトの探索範囲を制限することが可能となる。

メソッドによるマッチング

オブジェクトを探索する際、オブジェクト名のみでマッチングを行なうのは危険である☆。よって、オブジェクトが備えているメソッドに関して、

- メソッド名／数
- 引数の数／型

もマッチングの対象とする。

リモートオブジェクトまでの距離

実際に探索を行なう場合、探索時間(探索速度)も重要な要因となる。ドメイン化は、論理的にオブジェクト空間を制限する手段であるが、物理的にオブジェクト空間を制限し、探索時間を短くする方法も必要である。このため、オブジェクトの距離という概念を導入する。

オブジェクトの距離は、オブジェクト探索のリクエストを送る分散仮想マシンのホップ数で表わす。分散仮想マシンは、以下のように、リモートオブジェクトまでの距離に応じて、探索範囲の制限を行なう。

1. 最初にオブジェクト探索のリクエストを送る分散仮想マシンは、探索範囲を指定するためのホップ数を設定する。
2. 各分散仮想マシンは、次の(近隣の)分散仮想マシンにリクエストを送る際、ホップ数を1減らす。
3. ホップ数が0になったリクエストは、他の分散仮想マシンに転送しない。

以上をまとめると、各オブジェクトは、以下の項目をキーとして探索が行なわれる。

- オブジェクト名
- オブジェクトID
- ドメイン名
- メソッド
- 距離

3.3 オブジェクトの移動

分散オブジェクト環境におけるオブジェクトアクセスのモデルを考えてみる。

M: ホスト名
o: 未起動のオブジェクト
O: 稼働中のオブジェクト
→: サービス
⇒: オブジェクトの移動

とすると、例えば、ホストM上の稼働中のオブジェクトは、M1:Oと表わされ、M1がM2上のoからサービスを受ける場合は、M1 ← M2:oと表わされる。また、M2上のoをM1上に移動する場合は、M2:o ⇒ M1と表わされる。

これより、従来の分散アプリケーションに似た単純なサーバークライアントモデルは、

$$M1 \leftarrow M1{:}o,\ O$$
$$M1 \leftarrow M2{:}o,\ O$$

と表わされるオブジェクトアクセスを提供していることになる。JavaやHORBなどの最近の分散オブジェクト環境では、この他に

$$M1 \leftarrow (M2{:}o \Rightarrow M1)$$

と表わされるアクセスも提供している。

今回は、動的で自由度の高いオブジェクトアクセスを行なうために、以下のマイグレーション機能を提供する。

$$M1 \leftarrow (M2{:}O \Rightarrow M3)$$

ただし、$M2 = M1, M3 = M1$でもよい

これらの機能は、分散仮想マシンが提供する。各オブジェクトは、それぞれ1つのバイトコードファイルになっているので、ファイルを転送することに

☆ オブジェクト名は等しいが、性質の異なるオブジェクトが存在するかもしれない。

より、オブジェクトコードの転送を行なうことができる。また、分散仮想マシンは、現在実行しているオブジェクトの内部データや制御情報を管理しているので、これらを他の分散仮想マシンに転送することで、転送先でオブジェクトの実行を再開することができる。

4. 分散仮想マシンの実装

オブジェクト作成用言語“えーとし”は、現在、UNIX上の処理系を開発中である。“えーとし”の処理系の開発のしやすさから、バイトコードを実行する分散仮想マシンは、スタックマシンとしてC言語を用いて実装する。これにより、“えーとし”で作成したオブジェクトは、その処理系により図5のようなスタックマシン用バイトコードに変換され、1オブジェクトが1ファイルに格納される。

```
CodeFile {
  unsigned    magic;
  u_short     version;
  u_short     const_pool_cnt;
  const_info  const_pool[const_pool_cnt - 1];
  u_short     this_class;
  u_short     super_class;
  u_short     type_cnt;
  u_short     types[type_cnt];
  u_short     slot_cnt;
  slot_info   slots[slot_cnt];
  u_short     shared_cnt;
  shared_info shared[shared_cnt];
  u_short     methods_cnt;
  method_info methods[methods_cnt];
  u_short     name_cnt;
  u_char      src_name[name_cnt];
}
```

図5 “えーとし”のバイトコードイメージ

各フィールドは、以下のような情報を持つ。

magic
　マジックナンバー
version
　処理系バージョン番号
const_pool_cnt
　cons_pool のサイズ
const_pool
　このオブジェクトで利用する各定数情報
this_class
　クラス名の const_pool へのインデックス
super_class
　スーパークラスのインデックス
type_cnt
　types のサイズ
types
　型名のインデックスの配列
slot_cnt
　slots のサイズ
slots
　スロット (メンバ変数) の情報
shared_cnt
　shared のサイズ
shared
　スタティックメンバ変数の情報
methods_cnt
　methods のサイズ
methods
　メソッド情報
name_cnt
　src_name のサイズ
src_name
　ソースファイル名

本稿における分散仮想マシンは、通常の仮想マシンと異なり、以下の機能を有している。

分散仮想マシンは、オブジェクトを実行するにあたり、そのバイトコードファイルをロードして、個々のオブジェクトごとに、図6のようなオブジェクトイメージを構築する。

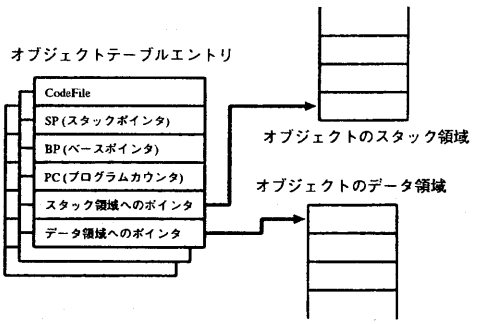

図6 オブジェクトの実行イメージ

オブジェクトテーブルエントリは、現在実行中のオブジェクトを管理するための制御情報を格納するテーブルで、OSにおけるプロセステーブルに該当する。従って、移動させたいオブジェクトのオブジェクトテーブルエントリや、そのスタック／データ領域を移動させることで、たとえ実行中であって

も、分散仮想マシン間でオブジェクトを移動させることが可能となる。

また、オブジェクトアクセスが発生した場合の処理を、外部に対して隠蔽し、仮想マシンの内部で全て行なうことで、オブジェクトアクセスにおける複数レベルの透過性の実現を目指している。例えば、他のオブジェクトへのアクセスを含む図7の“えーとし”のコードは、図8で示すバイトコードへ変換され、以下のように動作する。

```
clas JJJ is
    .
    .
  t = new TTT;
    .
    .
end
```

図7 オブジェクトアクセスを伴う“えーとし”コード

```
    .
    .
new         TTT      ; TTTオブジェクトを生成
invoke      create   ; コンストラクタ呼び出し
ref_local t          ; 変数tのアドレスをpush
assign               ; 代入
    .
    .
    .
```

図8 変換されたバイトコード

分散仮想マシンは、JJJ.eのバイトコードJJJ.code(図8)をロードしてJJJのイメージを構築し、仮想マシン命令を順次実行する。new TTT命令では、以下の処理を行なう。

1. 探索ポリシーに基づいて、TTTの探索を開始する。
2. 得られたTTT.codeからTTTのイメージを構築する。
3. JJJのデータ領域に、TTTインスタンスの領域を用意する。
   同時に、TTTのオブジェクトテーブルへのポインタも格納する。
4. tの領域をスタック上に用意して、tから3.の領域を指す。

5. まとめ

本稿では、従来の分散オブジェクト環境でのオブジェクトアクセスにおける問題点に着目し、オブジェクトアクセスに対する複数レベルの透過性を提供して、また動的なオブジェクトアクセスを実現するために、分散仮想マシンを導入したオブジェクト環境とそのプログラミングに関して述べた。

これらの機能を実現した分散オブジェクト環境では、この環境下で動作するユーザインターフェイスとなるシェルを用意して、またファイルをオブジェクトとすることで、環境自体に対する透過性も提供することが可能となる。

今後は、マルチスレッド機能を搭載した分散仮想マシンを実装し、その性能評価を行なうと共に、ファイルオブジェクトやシェルを用意して環境自体に対する透過性の提供について調べて行きたい。

参考文献

1) Zhonghua Yang and Keith Duddy: “CORBA: A Platform for Distributed Object Computing” OPERATING SYSTEMS REVIEW, ACM April, 1996
2) Sun Microsystems Inc.: “http://java.sun.com” JAVA Home Page
3) Sun Microsystems Inc.: “http://splash.javasoft.com-/pages/intro.html” Remote Objects for Java (tm)
4) Satoshi Hirano: “http://ring.etl.go.jp/openlab/horb” HORB Home Page
5) N.Fujinami and Y.Yokote: “Naming and Addressing of Objects without Unique Identifiers” Sony Technical Report SCSL-TR-92-004
6) 久野 靖, 大木 敦雄:「抽象状態に基づく並列オブジェクト指向p6」投稿中