# 並行オブジェクト指向言語による分散システムの記述

3M－1

鴻巣 淳　児玉 靖司　武田正之
東京理科大学 理工学研究科 情報科学専攻

## 1 はじめに

オペレーティングシステムの分野では、ネットワーク透過な分散環境の実現に関する試みが多く行われるようになっている。

我々は、「YAWE」OS を通じて、並行オブジェクト指向言語をシステム記述に取り入れる試みを行ってきた[1]。本稿は、「YAWE」の延長上で、複数ノード上にまたがる分散環境を、並行オブジェクト指向言語に基づいて記述する方法について議論する。また、我々がWS上に実装した並行オブジェクト指向言語上でおこなった、OS環境のシミュレーションについて報告する。

## 2 システム構成

ここでは、ABCM/1 [3] に基づく並行オブジェクトを、分散システムを実現する上で基本的な構成要素として用いる方法を提案する。

以下に説明する環境上では、オブジェクト管理、メッセージ送受関連のプリミティブを除いたほとんど全てのシステムサービスが、並行オブジェクトによって提供される。本環境では、各ノードには以下のようなオブジェクト空間の階層構造を構成する。

### 2.1 ノード上の階層構造

各ノード上には、安全で柔軟なネットワーク透過サービスを提供するために、次に示す 3段階のシステムの階層構造を構成する。(図1)

1. システム層:　ハードウエアおよび、マイクロカーネルなどが存在するレイヤ。システムの最も低層を構成するレベルである。

2. ローカル層:　単一ノード内の、ローカルなサービスを行うオブジェクトの存在するレイヤ。このレイヤのオブジェクトは、**ローカルオブジェクト**と呼ばれ、生まれたマシンに束縛される。

3. 共有層:　最も抽象度の高いレイヤ。このレイヤのオブジェクトは、**グローバルオブジェクト**と呼ばれ、すべての層のサービスを用いることができ、ネットワーク透過なサービスを利用することができる。また、このレイヤのオブジェクトは、他ノードへのマイグレーションが可能である(禁止も可能)。

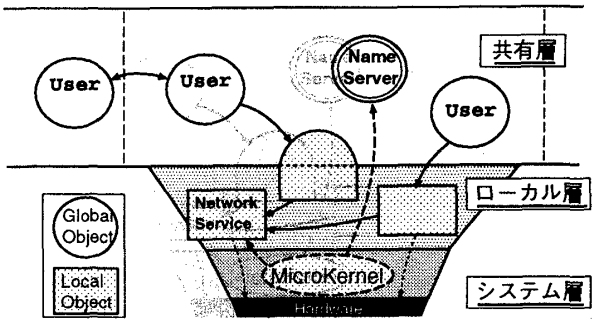


図 1: ノード上のシステム階層

上の3層構造は、**システム層**が、並行オブジェクト環境を作り出すための基盤、**ローカル層**が、ネットワーク透過な分散環境を創り出すための基盤であり、この下2層のサポートのもとで、**共有層**という高度にネットワーク透過な空間が構成されているものととらえることができる。

ローカル層と共有層のもっとも明確な差は、前者のオブジェクトがノードに束縛されるため、ノード内でローカルなオブジェクトID(**ローカルID**)のみを持つのに対し、後者のオブジェクトは、ノード間のマイグレーションが仮定されているため、ローカルIDに加えて環境全体で一意に割り当てられるグローバルなID(**グローバルID**)を持つ点である。

グローバルIDは、必要に応じて、**ネームサーバ**を通じてローカルIDに変換され、ローカルIDが実際のアドレスとして使用される。

### 2.2 ローカルオブジェクト

ローカルオブジェクトとグローバルオブジェクトの最も本質的な差は、ローカルオブジェクトの利用できるサービスは、自ノードのローカルオブジェクトのメソッドの

---

An Inplementation of Distributed System Using Concurrent OOPL.
Jun Konosu, Yasushi Kodama, Masayuki Takeda
Science University of Tokyo
2641 Yamazaki,Noda-shi,Chiba,278,japan

みだということである。それ以外のメソッドの呼び出しを行うと、システムレベルで拒否される(メッセージの受渡し自体は可能だが、メソッドが起動される時点で失敗する)。

つまり、ローカル層のオブジェクトは、暗黙的にネットワーク透過なサービスを呼び出すことがなく、この空間内ではローカルに閉じた作業がおこなわれることが保証されるこの制限により、低レベルなシステムオブジェクトが資源要求のループを起こすなどの危険[2]が回避され、安全なシステムの動作が可能となる。

### 2.3 メッセージ送信

オブジェクト間のメッセージ送受信は、マイクロカーネルのサポートするシステムサービスであり、言語のプリミティブによって記述される。

全てのオブジェクトは、オブジェクトIDを介して、互いにネットワーク透過にメッセージを受け渡すことが可能である。

オブジェクト内でメッセージ送信プリミティブが呼び出されると、そのノードのマイクロカーネルは必要に応じてローカル層のネットワークサービスオブジェクトを呼び出し、メッセージの送信を行う。

### 2.4 ネットワーク仮想記憶

また、本環境上で、ネットワーク仮想記憶を実現した。ネットワーク仮想記憶は、他ノードに存在する記憶資源を、自ノードの記憶に対するアクセスと同様にアクセスする機能である。これは、我々の環境では、大きなデータのリモート参照時などに、共有層のオブジェクトが暗黙的に使用する。

## 3 WS上のシミュレーション

我々は、以上に紹介したシステムを、WS上で実現し、シミュレートを行った。現在、並行オブジェクト言語環境は、C++ 言語 (g++-2.7.0) + LWP (Solaris2.4) で作成されている。UNIX プロセスを仮想マシンに見立て、並行オブジェクトを、プロセス中のLWPに対応させた。

### 3.1 言語環境

言語レベルでの変更は特に加えず、C++の直接的なクラスとして、仮想マイクロカーネルや並行オブジェクトを実現している。

オブジェクトは、システムクラスとして用意されているクラスの子クラスとして記述する。

オブジェクトは、オブジェクトIDによって識別される。メッセージ「M」をグローバルオブジェクトgobj、またローカルオブジェクト lobj に送信する記述は、それぞれ

```
M > gobj;        M.SendL(lobj);
M >> gobj;       M.CallL(lobj);
```

と書く(上:過去型, 下:現在型)。また、グローバルIDからローカルIDを得るプリミティブは、

```
&<グローバルID>;
```

である。つまり、下の 2つの文、

```
M > gobj;        M.SendL(&gobj);
```

は等価となる。

## 4 まとめ

ネットワーク透過サービスの実装をマイクロカーネルから分離し、また、並行オブジェクト空間を、“ローカル”と“共有”の2層に分けることで、柔軟なシステムサービスを安全に提供することが可能となった。

ネットワーク透過性の実現部分をオブジェクト空間に切り出すことで、マイクロカーネル自身は小型で独立性の高いコンポーネントとなり、またシステムサービスのほとんどが、オブジェクトとのメッセージの受渡しという統一的な方法によって実現される環境が作り出された。

また、平坦なオブジェクト空間中に、ネットワーク透過サービスの実現部分を置くことによるサービスの安全性の問題を、ノード内でローカルに作業が行われることが保証される空間、“ローカル層”の導入により回避することができた。

そして、以上の環境を、実際にWS上でシミュレートし、並行オブジェクトによる記述が分散環境に有効であることを確かめることができた。

## 参考文献

[1] Yasushi Kodama, Norihisa Doi. YAWE: Yet Another Window Environment. Department of Computer Science, Keio University. 1992.

[2] SCONE: Using Concurrent Objects for Low-level Operating System Programming. Jun-ichiro Itoh, Yasuhiko Yokote, Mario Tokoro. In OOPSLA, 1995.

[3] 米澤明憲, 柴山悦哉 et al. オブジェクト指向に基づく並列情報処理モデル ABCM/1 とその記述言語 ABCI コンピュータソフトウエア Vol.3, No.3. 1986.