分散オブジェクトに対するプロセス粒度の動的な変更について*

4N－1

篠原 正紀† 藤崎 智宏† 荒野 高志†

NTT ソフトウェア研究所‡

1 はじめに

オブジェクト指向アプリケーションでは、ソフトウェアを独立したオブジェクトの集まりとして扱い、これらオブジェクト間のメッセージ送受信によりシステム全体が動作する。オブジェクトの並列動作によるパフォーマンス向上や、オブジェクトのマイグレーションによる耐故障性の高いシステムの構築が可能である。

アプリケーションが実際にマシン上で動作するときには、実行主体であるプロセスとオブジェクトとを対応させる (図1)。本稿では、このときの１つのプロセスに対応するオブジェクトの数を、そのプロセスのプロセス粒度と呼ぶ。

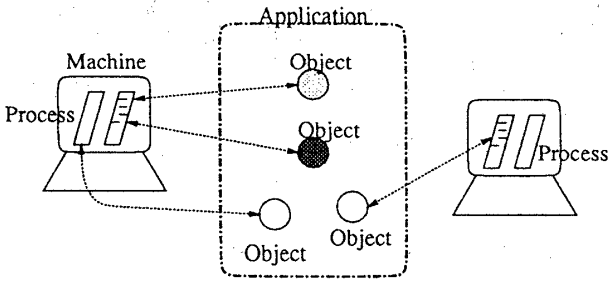

図 1: アプリケーションの実現

オブジェクト指向アプリケーションにおいても、プロセス粒度は設計時に決定される。これには次のような問題がある。

- 個々のオブジェクトが別々のプロセスに対応するように設計すると、オブジェクト数が増大した場合に、通信、主記憶などの使用資源の増加が起こる。
- 逆に、一つのプロセスで多くのオブジェクトを集中的に管理するように設計すると、プロセス間の通信が少なく実現が容易であるが、大規模なアプリケーションを管理する際に性能、安全性が低下する。

2 分散オブジェクトとプロセス粒度

分散オブジェクト環境では、オブジェクトはネットワーク中に分散することができる。またオブジェクト同士のメッセージ送受信は、オブジェクトの名前とそのオブジェクトに対し動作を指示するメソッドを指定することにより行われ、オブジェクトが実際にどのマシン上にあるかを意識しなくてよい [1]。これは RPC など既存のプロセス間の通信に比べ非常に簡便に使用でき、このオブジェクトの位置透過性により、効率的なオブジェクト配置による負荷分散が可能となっている。

ただしこの分散オブジェクトを用いた方法でも、プロセス粒度、つまりオブジェクトとプロセスの対応付けは設計時に行う。オブジェクトの配置はプロセスの配置に依存し、オブジェクトの数や場所などが制限されるため、1節で挙げたような問題が起こる。この問題を解決するには、プロセス粒度を設計時だけでなく実行時にも変更する手法が有効である。この手法を用いて以下のような操作を行う。

- 並列処理
  一つのプロセスに対応している複数のオブジェクトを、別々のプロセスに分配して並列動作させる。
- 負荷の適切な分散
  負荷が大きなマシンにはプロセス粒度の小さいプロセスを配置し、負荷が小さなマシンにはプロセス粒度の大きいプロセスを配置する。

これにより、変化する CPU 資源環境にオブジェクトを効率良く対応させる。特に大規模なシステムでは、オブジェクトの配置によって性能に大きな差が現れる [2]。ネットワークのトポロジや回線容量等を考慮してプロセス粒度の変更やオブジェクトの再配置を行い、システム全体のパフォーマンスの向上を図る。

3 プロセス粒度の変更

本節では、プロセス粒度を動的に変える仕組みを説明する。分散オブジェクトを実現するときの機能を、以下の２つの部分に分ける。

- メッセージの処理など、本質的なオブジェクトの動作を扱う部分
- オブジェクトを実際にマシン上で動作させるために、OS など環境とのインターフェイスを提供する部分

インターフェイス部分は、オブジェクトをプロセスと対応させるための処理を行う部分であり、各プロセス内にあるオブジェクトの名前や数の管理を行う。このインターフェイス部分をプロセス・コントローラと呼ぶ (図2)。

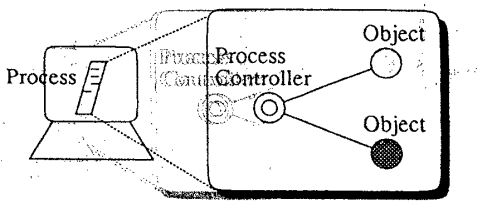

図 2: オブジェクトとプロセスの対応

3.1 プロセス・コントローラ

プロセス・コントローラはプロセスと１対１に存在し、自分が管理するオブジェクトの名前と数を把握している。プロセス・コントローラは実際に他のプロセスと通信するが、分散オブジェクト機構を用いることにより、どのプロセスに存

*Dynamic Process-Mapping Mechanism of Distributed Objects
†Masanori SHINOHARA, Tomohiro FUJISAKI, Takashi ARANO
‡NTT Software Laboratories

在するオブジェクトに対しても、その違いを意識しないでプロセスとの対応を変更することができる。
プロセス・コントローラは以下のような機能を持つ。

1. 自分の管理するオブジェクトを消去する。
2. 自分の管理するオブジェクトを、他のプロセス・コントローラの管理下へ移動する。
3. プロセス内にオブジェクトを新しく生成する。
4. オブジェクトリファレンスの要求に対して、指定された名前をもつオブジェクトリファレンスを返す (図3)。
5. 新規のプロセス・コントローラを生成する (プロセスが同時に生成される)。
6. 自分自身を終了する (プロセスも同時に kill される)。

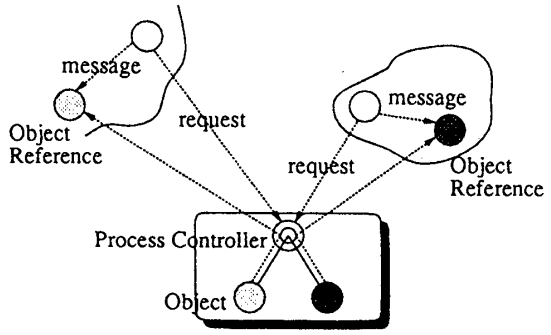

図 3: オブジェクトリファレンスの要求

3.2 プロセスの結合と分割

プロセス・コントローラの 3.1節の機能を用いることにより、次のような操作を可能にする。
複数のプロセスに分散しているオブジェクトを一つのプロセスにまとめることを、プロセスの結合と呼ぶ (図4)。これは 3.1節の (2) と (6) により実現される。プロセスの結合により、例えば大規模システムにおける分散処理でプロセス数が膨大になってしまうときなど、複数のオブジェクトを一つのプロセスにまとめて管理することができる。

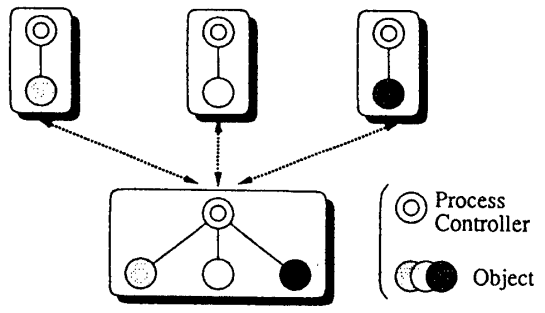

図 4: プロセスの結合と分離

一つのプロセス中のオブジェクトを複数のプロセスに分けることをプロセスの分割と呼ぶ (図4)。これは 3.1節の (2) と (5) により実現される。プロセスの分割によりそれぞれのオブジェクトを並列動作させ、パフォーマンスの向上を図ることができる。

4 考察

細かいオブジェクトをプロセスにすると、並列性によりシステムの動作効率を上げることができる。しかしそれによってプロセス数が増大し、通信トラフィックや、主記憶などの使用資源の増加を招いてしまう。そのため、適切なプロセス粒度について考察する。
オブジェクトの配置を決定するための基準として、静的なものは以下のような情報がある [2]。

- マシンの CPU パワー
- ネットワークのトポロジや回線容量

さらに動的な情報としては以下のようなものがある。これらは、オブジェクトの配置を行うために収集する必要がある。

- マシンの負荷
- マシン上に存在するオブジェクト数

またオブジェクトに対応するプロセスが消滅してしまった場合でも、マイグレーションによって動作を継続し耐故障性を向上させることができる。これは専用のオブジェクト管理デーモンを用意することにより実現する [3]。
プロセスの結合や分割を行うと、管理デーモンが監視すべきプロセスが変化する。そのためプロセス・コントローラはプロセスの変化を管理デーモンに通知する役割を持つ。

5 まとめ

従来分散オブジェクト環境では、オブジェクトをどのように実際のプロセスと対応させるかは設計時に決定していた。個々のオブジェクトが別々のプロセスに対応するように設計すると、並列動作によるパフォーマンス向上が期待できるが、オブジェクト数が増大した場合には使用資源の増加が起こる。逆に、一つのプロセスで多くのオブジェクトを集中的に管理するように設計すると、プロセス間の通信が少なく実現が容易であるが、大規模なアプリケーションを管理する際に性能、安全性が低下する。
そのため、オブジェクトとプロセスのインターフェイスとしてプロセス・コントローラを導入し、プロセス粒度を実行時にも変更する仕組みを提案した。プロセス・コントローラはプロセス毎に存在し、各プロセス内にあるオブジェクトの名前や数の管理を行う。これを用いて、オブジェクト側ではプロセス操作を意識することなく、プロセス粒度を変更することができる。
この仕組みにより、オブジェクト数やネットワークのトポロジ、回線容量等を考慮したオブジェクトの再配置を行うような、システム再構築が実行時にも可能となる。

参考文献

[1] Takashi Arano et al. A Computer Network Management System Platform based on Distributed Objects. *IFIP/IEEE DSOM '95*, 1995.

[2] 青野 博, 藤崎 智宏, and 荒野 高志. 性能を最適にするための分散オブジェクトの配置法−分散オブジェクト指向プラットフォーム上の NMS の実例より−. 信学技報, pages 23–30, May 1995.

[3] 藤崎 智宏 and 荒野 高志. 分散オブジェクト指向ネットワーク管理システムにおけるオブジェクト管理手法. 情報処理学会分散システム運用技術研究グループ研究会, 1995. DSM-951153.