マルチメディア通信と分散処理 66－4
（１９９４．７．７）

オブジェクト指向分散環境 OZ++ のオブジェクト管理系の設計

西岡 利博 * 大西 雅夫 * 吉田 泰光 * 籠 浩昭 *
三菱総合研究所 東洋情報システム 日本ユニシス 三菱総合研究所

鈴木 敬行 * 濱崎 陽一 塚本 享治
SBC 電子技術総合研究所 電子技術総合研究所
* 情報処理振興事業協会「開放型基盤ソフトウエア研究開発評価事業」研究員

分散システムとオブジェクト指向の概念とは親和性が高く、これまでにも多くのオブジェクト指向分散システムが研究されてきたが、それらの基本的なオブジェクト管理の方針は、個々のオブジェクト性質によらず画一的であるため、アプリケーションによっては柔軟性に乏しい構成を強いられることがあった。本論文では、オブジェクト管理部分をオブジェクトで実現するフレームワークを与えることにより、オブジェクトに応じてカスタマイズできる設計例を報告する。

Object Management in OZ++ : Object-Oriented Distributed Systems Environment

Toshihiro Nishioka* Masao Oonishi* Yasumitsu Yoshida*
Mitsubishi Research Institute Toyo Information Systems Nihon Unisys

Hiroaki Kago* Takayuki Suzuki* Yoichi Hamazaki
Mitsubishi Research Institute Sharp Business Computer Electrotechnical Lab.

Michiharu Tsukamoto
Electrotechnical Lab.
* Researcher of Research, Development and Evaluation of
Open Fundamental Software Technology Project
in Information-technology Promotion Agency, Japan

Although many of object-oriented distributed systems have been researched and developed because of the affinity between distributed systems and the concept of object-orientation, structures of their applications are often forced to be less flexible since their policies of basic object management are less sensitive to characteristics of an individual object. This paper reports an experience in designing object management system of an object-oriented distributed system. The object management system itself is provided as a framework comprised by objects to allow customization.

1 はじめに

分散システムでは、地理的に分散した実行主体が相互に連係するため、それらの間には明確なインタフェースが定義され、相互の依存関係がなるべく希薄になるように設計される。このような実行主体をオブジェクトと考え、オブジェクトの持つメソッドをインタフェースとし、情報隠蔽の性質を用いて相互の依存部分を減らしていくことは、分散システムの考え方によく適合している。

これまでにも多くのオブジェクト指向分散システムが開発されてきた [1, 2, 3]。

ところで、実用的な分散環境では、さまざまな環境で作られたオブジェクトが同時に動作しなければならない。それらのオブジェクトは、同じメソッド起動というインタフェースでアクセスできるとは言え、性能上の特徴はさまざである。保持しているデータの量、計算時間、メモリ消費量、コードの大きさも違えば、そのオブジェクトを管理するプロセスが常駐しているかどうかという違いもある。このようなさまざまなオブジェクトのすべてを、一元的なスキームで管理するのには効率上の問題がある。すなわち、不要な機能まですべて備える必要があり、実行時にさまざまな検査が入り、個々のオブジェクトやノードの特性に根差した調節が難しい。

本稿では、オブジェクトの基本的な振舞いを規定する部分を、やはりオブジェクトで実装することにより、継承を用いたカスタマイズを念頭に置いて設計した例を報告する。これにより、特徴の異なるオブジェクトの管理を、柔軟に、かつ効率的にできるようにする。

以下、第 2 節では、 OZ++ システムの概要を説明し、第 3 節では、オブジェクトの基本的な振舞いを定めるオブジェクトであるオブジェクトマネージャの設計方針について述べる。

2 OZ++ システム

OZ++ は、オブジェクトの交換と共有に基づく、オブジェクト指向分散処理環境である。オブジェクトモデルと実行意味論は、電総研で研究開発された OZ+ [6] を基本とし、これを洗練したものとして研究開発されており [7]、その成果は広く公開される予定である。現在、実装が進められており、ランタイムカーネルが動作し始めたところである。

従来のオブジェクト指向分散環境と異なり、OZ++ は、ネットワークを流れるメッセージの中にもオブジェクトを含めることができ、かつ、そのようなオブジェクトのコードもまた、ネットワークを通じて自動的に供給する機構を提供する点で特徴がある [8]。

OZ++ は OS ではなく、既存 OS 上で動作し、分散環境を提供する環境である。 OZ++ 上のプログラムのために、 OZ++ 言語と、分散環境上で利用できるクラスライブラリ、ブラウザ、デバッガなどのプログラミング環境が提供される。また、OZ++ オブジェクトを管理するオブジェクト管理系も、 OZ++ 言語で記述されたものが提供される。

2.1 OZ++ 言語

OZ++ システムの提供する機能を利用する、簡単かつ強力なモデルを提供する言語として、OZ++ 言語が提供される [5]。 OZ++ 言語は、C 言語をもとに、オブジェクト指向、並列プログラミングなどの言語機能を入れたものである。

OZ++ のオブジェクトには、大域的に識別可能なオブジェクト ID を持つ**グローバルオブジェクト**と、グローバルオブジェクトの実装に使われる**ローカルオブジェクト**とがある。**セル**は、一つのグローバルオブジェクトと、任意の数のローカルオブジェクトの集まりである。複数のセルにまたがって存在するオブジェクトはない (図 1)。

2.2 OZ++ の実行系

前節で述べた OZ++ 言語は、コンパイラによって C 言語のソースコードに変換される。これをコンパイルして得られるコードをロードし、実行させる実体 (ランタイムカーネル) を、エグゼキュータと言う。エグゼキュータは次のような機能を持つ。

1. **スレッド管理:** OZ++ のプログラムはマルチスレッドで実行されるが、複数のスレッドを管理し、スケジューリングするのはエグゼキュータである。

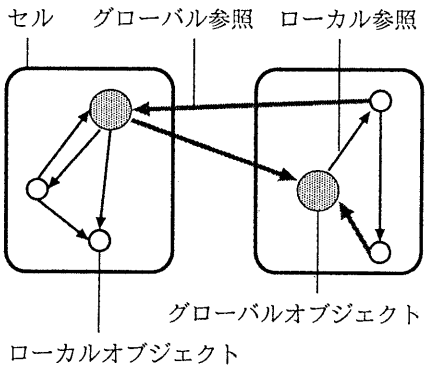

図 1: OZ++ 言語のオブジェクト

2. **コード管理:** エグゼキュータ上で動作するオブジェクトのコードを二次記憶からロードし、管理する。

3. **メモリ管理:** オブジェクトのためのメモリを確保し、ガーベジコレクションを行う。

4. **通信機能:** 分散環境上のオブジェクトが互いにメソッド起動を行う機能を提供する。

2.3 分散オブジェクト管理システム

OZ++ では、ネットワーク上の各種のサービスはもちろん、オブジェクトの基本的な振舞いを管理する部分についても、すべて OZ++ 言語で記述されたオブジェクトが行う。これらの、他のオブジェクトの動作に必須であるいくつかのオブジェクトから構成されるフレームワークを、分散オブジェクト管理システムと呼ぶ [4]。これには次のようなものが含まれている。

1. **クラスオブジェクト:** クラスに関する情報を管理する。コンパイラやブラウザに情報を提供し、実行時には、エグゼキュータにコードを供給する。

2. **ネームディレクトリ:** OZ++ プログラムからネットワーク上のサービスを利用する場合、最初は名前をキーとしてサーバのオブジェクト ID を得る。このため、名前とオブジェクト ID との対応を管理するネームサーバが提供されており、これをネームディレクトリと呼ぶ。

3. **オーセンティケータ:** 他のグローバルオブジェクトのメソッドを起動する際に、認証データを支給する。ユーザに関する情報も管理する。

4. **オブジェクトマネージャ (OM):** 各エグゼキュータに必ず一つだけ存在し、OZ++ のプログラム中では、エグゼキュータを抽象する役割を果たす。エグゼキュータ ID から一意に決定できるオブジェクト ID を持っており、プログラミング言語から、他の情報を用いずに参照できる。セルの状態遷移の管理と、オブジェクトの世界とエグゼキュータとの橋渡しを行う。

上に述べられた各種の管理系のオブジェクトの中でも、OM は、他のオブジェクトと異なり、エグゼキュータとのインタフェースを有している。それゆえ、次節に述べるような特殊な機能を担っており、このそれぞれについて、エグゼキュータとの機能の切りわけが問題となる。次節では、このような OM の設計方針を述べる。

3 オブジェクトマネージャの設計

3.1 OM の機能

OM は、各エグゼキュータに一つずつ配置されるオブジェクトであって、以下のような役割を与えられている。

1. セルの状態遷移を管理する。

2. エグゼキュータごとに固有の情報 (使用しているクラスのバージョン、エグゼキュータのオーナなど) を管理する。

3. エグゼキュータからオブジェクトの機能を使う際、その橋渡しを行い、逆に、オブジェクトがエグゼキュータの機能を使う際の窓口となる。

本稿では、OM の設計思想が最もよく表れている、1 の設計について次節で述べ、2, 3 については、割愛する。

3.2 オブジェクトの状態遷移の管理

状態遷移について、OM とエグゼキュータは、以下のように機能分担する。すなわち、OM は、

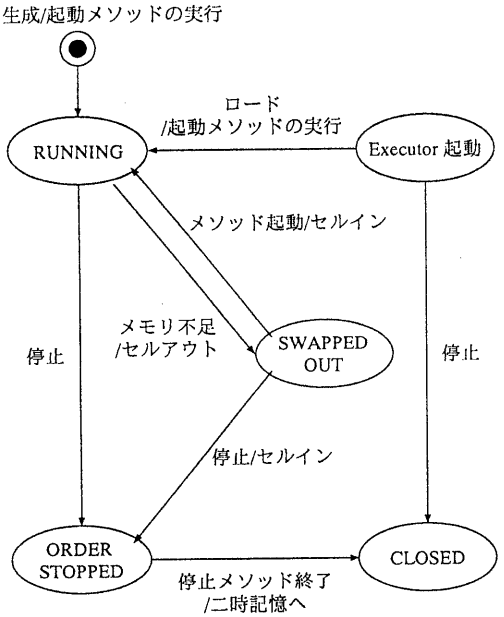

図 2: オブジェクトの状態の一例

状態遷移そのものを含む、セルの状態の管理方針を決定し、それを実装するために必要なメカニズムをエグゼキュータが提供する。

図 2は、その方針にしたがって現在実装中の OM が管理する状態遷移の概略である。

状態遷移は OM 側が管理するが、エグゼキュータも、エグゼキュータから見たセルの状態を記した表を持っている。すなわち、オブジェクト間のメソッド起動は、オブジェクト ID を指定して行われるので、エグゼキュータは、そのオブジェクトが存在するのかどうか、また、存在したとして、メソッドを起動できる状態かどうかを判定しなければならない。このため、オブジェクトテーブル (以下 OT と略す) という表に、そのエグゼキュータ上に存在するセルの状態を記す。'READY' 状態では、到着したメッセージは直ちに処理され、スレッドが生成される。'QUEUE' 状態では、到着したメッセージをキューイングし、 OM にメッセージが到着したことを伝える。'STOP' 状態では、到着したメッセージは例外となる。 OT の内容は、 OM で管理している状態と整合していなくてはならないので、 OM からの指示によってのみ更新される。

以下、設計の詳細を述べる。

セルの起動:

図 2の中の、'Executor 起動' から 'RUNNING' に移る遷移がセルの起動を表している。

セルは最初、主記憶上にはなく、 OM によって二次記憶からロードされる。最も単純には、エグゼキュータの起動と同時に、すべてのセルを主記憶にロードする設計が考えられる。しかし、使用頻度が低く、ロードに時間のかかるセルを常にロードすれば、効率上の問題が生じる。このため、メソッドが起動されてから動的にセルをロードする。メソッドが起動されたことは、前述の OT の 'QUEUE' 状態のメカニズムを用いて検出できる。

さらに、セルの種類によっては、一貫性維持処理の都合上、ロードされた後、他のメソッドが起動される前に、準備のためのメソッド (起動メソッド) が起動される必要があることがある。このため、 OT 上のセルの状態は、起動メソッドが終了してから 'READY' に変更する。

ところで、このためには、 'QUEUE' 状態のセルであっても、起動メソッドだけは起動できなくてはならない。このため、エグゼキュータは、特定のメソッドだけは 'QUEUE' 状態でも起動できるというメカニズムを提供する。

セルアウト:

仮想記憶を大量に占有できないプラットフォームでは、使用されていないセルを二次記憶に退避できれば便利であろう。これをセルアウトと言い、逆をセルインと言う。図 2中の 'SWAPPED OUT' 状態が、セルアウトされた状態である。

方針にしたがえば、セルアウト / セルインのタイミングは OM が指定すべきである。エグゼキュータは、実際にセルアウト / セルインを行うサブルーチンの他、タイミングの検出に役立ちそうな、さまざまな統計量を知るためのサブルーチンを提供する。セルアウト中のグローバルオブジェクトに対するメソッド起動は、 OT の 'QUEUE' 状態のメカニズム用いて処理できる。

セルの停止

図2の中の 'ORDER STOPPED' 状態が、セルが停止させられた状態である。

エグゼキュータを停止する場合、最も単純には、すべてのスレッドの実行を中断する設計が考えられる。しかし、単にスレッドを中断すると一貫性維持の問題が生じる。このため、実行中のスレッドの終了を待ち合わせ、一方で、新しいスレッドの生成を禁止する。新しいスレッドの生成の禁止は、前述の OT の 'STOP' 状態のメカニズムで実現できる。この他に、エグゼキュータは、スレッドの終了を待ち合わせるメカニズムを提供する。

さらに、一貫性維持を行うセルでは、セルを停止させる際に、停止準備のためのメソッド (停止メソッド) を起動する必要があることがある。ところで、このためには、'STOP' 状態のセルであっても、停止メソッドだけは起動できなくてはならない。このため、エグゼキュータは、特定のメソッドだけは 'STOP' 状態でも起動できるというメカニズムを提供する。

また、セルは停止後、 OM によって永続記憶領域に書き戻されるが、中にはその必要のないセルもある。このため、 OM に対して、永続化するセルを指定できる。

3.3 議論

このような設計をするとき、いくつかの方向からの議論が必要である。

ランタイムカーネルで実現する場合に比べて、性能を著しく落さないか

実装上の工夫により、 OM とエグゼキュータの間のインタフェースは、遜色ない速度で動作する。性能上の差が最も大きいのは、オブジェクトが OM のメソッドを起動する部分である。また、 OM 内部のローカルオブジェクトに対するメソッド起動も、やや性能上の不利を被っている。

ところで、 OM が集中的に動作するのは、オブジェクトの状態遷移が頻繁に起こる、エグゼキュータの起動と停止のときである。しかし、起動時に支配的なのは、各種のオブジェクトやコードのロードであって、 OM の実行そのものではない。停止時は、 OM の性能が全体に影響するかもしれないが、これから停止するエグゼキュータの性能であるから、全体に与える影響としてはわずかである。

これに対して、例えば、主記憶の少ないノード上で、プロセスサイズの大きさを制限できる利益は大きい。

実際にカスタマイズする作業はどの程度柔軟にできるのか

有効にカスタマイズできるためには、 OM 全体が、いわゆるフレームワークとして、抽象クラスを主体とした設計になっている必要がある。中でも、セルの状態遷移を管理するのは、 OM 内で保持しているグローバルオブジェクトの表の各エントリに入っているローカルオブジェクトである。状態遷移をカスタマイズする場合には、この部分に対する変更が主体となる。したがって、特にこの部分については、抽象クラスの仕様、利用するプロトコルなどについての詳細なドキュメントなどの情報がきちんと与えられる必要がある。

他に、 OM 自身に、何かパラメータを変更するようなメソッドを設けることで、ここで述べているような柔軟性を実現する方法もある。しかし、ソフトウェアは常に、設計時には予想もしなかった方法で使われるものである。例えば、セルアウトという概念がない段階で、それを導入するとなったら、状態遷移図の再設計から始めなくてはならない。パラメタ化は、有効な方法ではあるが、パワーユーザの要求を満たすためには、メソッドを再定義させる方が望ましい。

より柔軟な設計はできないか

現在の設計では、オブジェクトに対するメタ操作は、すべて OM を経由するようになっているが、唯一、メソッド起動だけはエグゼキュータが自動的に行う。これも OM で管理するという選択肢もあった。すなわち、グローバルオブジェクトのメソッドは、直接起動せず、いったん OM に通知することとする。これを採用すれば、高度に柔軟なシステム構成が可能であり、かつ、エグゼキュータで OT を管理する必要もない。しかし、特にそのような処理を必要としない、通常のメソッド起動の方が、割合としては圧倒的に多いことと、その場合に著しい実行性能の低下を招くので、断念した。

より柔軟な設計という意味では、さらに検討を要する課題もある。例えば、複製やトランザクションの管理は、クラスライブラリによって別のフレームワークとして提供される。しかし、これらについても OM が管理できるような、より包括的なフレームワークの設計ができれば、さらに柔軟なシステム構成が可能となる。今後の検討課題である。

従来型の設計との比較

Eden [1] は、カーネルが画一的なトランザクションのメカニズムを提供しており、オブジェクトの状態遷移は固定されている。

Emerald [2] は、多重実装の概念を導入しており、コンパイラが、ソースコードを見て最適な実装を選択する。これは強力な機構であるが、静的にしか調整できない欠点を持つ。

ただし、これらは、OZ++ にはない、モービルオブジェクトの機能を実現しており、そのための設計上の制約も多いはずで、どちらが優れているとは一概には言えない。異機種環境を扱う OZ++ では、グローバルオブジェクトのマイグレーションを実現するよりも、データのモビリティはローカルオブジェクトのコピーアウトによって、可用性の向上は複製によって実現する方針を採用している。

重要な差異は、OM を継承によってカスタマイズしたとしても、それはやはり OM であり、分散環境上で他の OM と共存して利用できるという点である。

4 終りに

オブジェクト指向分散処理環境のオブジェクト管理系の設計について、オブジェクトの性質に応じた管理を行えるように、オブジェクトで実装し、継承によるカスタマイズができるように設計した例を述べた。

現状の実装では、まだフレームワークとその実現との切りわけが不十分である。今後、フレームワークとしての設計を充実させていく考えである。

謝辞

本研究を通じて、熱心な討論を頂いている、藤野晃延氏、吉屋 英二氏 (以上、富士ゼロックス情報システム)、千葉 滋氏 (東京大学理学部)、および、情報処理振興事業協会「開放型基盤ソフトウェア研究開発評価事業」のオブジェクト指向分散処理環境の研究チームのメンバー諸氏に感謝の意を表する。

本研究は、情報処理振興事業協会「開放型基盤ソフトウェア研究開発評価事業」の一貫として行われたものである。

参考文献

[1] Almes, G.T, Black, A.P., Lazowska, E.D. and Noe, J.D.: "The Eden System: A Technical Review." In *IEEE Trans. on Software Engineering*, Vol. SE-11, p. 43–58, 1985.

[2] Black, A., Hutchinson, N., Jul, E. and Levy, H.: "Object Structure in the Emerald System." In *OOPSLA '86 Proceedings*, p. 78–86, 1986.

[3] Dasgupta, P., LeBlanc, R. and Appelbe, W.: "The Clouds Distributed Operating System." In *IEEE 8th International Conference on Distributed Computing Systems*, San Jose, 1989.

[4] 籠他: 「オブジェクト指向分散環境 OZ++ のオブジェクト管理機構の概要」, 情報処理学会第 47 回全国大会, Oct. 1993.

[5] 西岡他: 「オブジェクト指向分散環境 OZ++ の言語の基本設計」, 情報処理学会第 46 回全国大会, Mar. 1993.

[6] 塚本他: 「オブジェクト指向開放型分散システム OZ+ の研究開発」, In 電総研彙報, 56(9), Sep. 1992.

[7] 塚本他: 「オブジェクト指向分散環境 OZ++ の基本設計」, SWoPP 93, Aug. 1993.

[8] 吉屋他: 「オブジェクト指向分散環境 OZ++ のクラス管理方式」, 情報処理学会第 47 回全国大会, Oct. 1993.