システムソフトウェアと
オペレーティング・システム　89－１６
シ　ス　テ　ム　評　価　2－１６
（２００２．２．１５）

エージェント技術を用いた分散オペレーティングシステムの構成手法

瀧本 栄二 †　芝 公仁 †　大久保 英嗣 ††

† 立命館大学大学院理工学研究科　†† 立命館大学理工学部

従来の分散システムは, システムを構成する計算機同士が資源を効率的に使用するために密接に結合されているため, 環境の変化に柔軟に対応することが困難である. 各計算機が自律的に動作し, 必要に応じて他計算機上の資源を管理することで, 計算機間の結合を弱くし, 環境の変化に対する柔軟性を向上させることが可能となる. 本稿では, 分散オペレーティングシステムをエージェントを用いて構成する手法と, 各計算機が自律的に動作する分散システムの構築手法について述べる.

A Construction Method of Distributed Operateing System Based on Agents

Eiji Takimoto †　Shiba Masahito †　Eiji Okubo ††

†Graduate School of Science and Engineering, Ritsumeikan University
††Faculty of Science and Engineering, Ritsumeikan University

In the conventional distributed systems, it is difficult to adapt to change of environment. This is because computers that consist of a system are closely related each other, in order to efficiently use system resources. It is possible to improve flexibility for change of environment by making each computer work autonomously and managing system resources of other computers only if those resources are requested. In this paper, a construction method of distributed system based on agent is described. By using this method, distributed system that each computer can run autonomously is realized.

1　はじめに

分散システムは、システム上の資源を有効活用するための機構を持っている。このため、分散システムを利用するユーザは、システム構成を意識することなく分散した資源を利用することが可能である。従来の分散システム構築手法では、位置透過性を実現するために、資源を管理するためのサーバやそれに準ずる機構を用いている。そのため、サーバにおける障害がシステム全体に影響を及ぼしたり、システムの変更が困難など、耐障害性や柔軟性において問題がある。

これらの問題を解決する 1 つの手段として、サーバなどのシステム全体を統括する管理機構を持たない自律分散システムがある。しかし、従来の分散システムの実現に用いられてきた分散オペレーティングシステム (以下 OS と記す) は、イベント駆動であり、自律的な動作を実現することが困難である。そこで、本研究では、自律的な概念を含んだ実行実体であるエージェントを分散 OS の構成要素とすることで、自律的に動作可能である分散 OS を実現することを目的としている。

エージェントには多様な概念が存在するが, 本 OS では, 自律分散を実現する上で必要となる自律性と移動性をエージェントに実現する。エージェントは、自律的にシステム全体や個々の計算機の状態を把握し、必要に応じて他計算機上へ移動したり他のエージェントと協調することで分散システムを実現する。また、本 OS では、ハードウェアをオブジェクトとして抽象化し、エージェントは、オブジェクトを操作することで資源の管理を行う。

本稿では、エージェントの実現法およびエージェントを用いた分散 OS の構成法について述べる。以下、2 章で本 OS の構成, 3 章でオブジェクトとエージェント, 4 章で分散システム構築のための機能, 5 章で関連研究について述べ, 6 章で本稿のまとめを述べる.

2　OS の構成

本 OS は、マイクロカーネル方式を基本としており, 計算機資源を抽象化するオブジェクト群とカーネルとしての機能を実現するエージェント群から構成される. 図 1 にシステム構成を示す.

図 1 の物理オブジェクト層は, CPU やメモリと

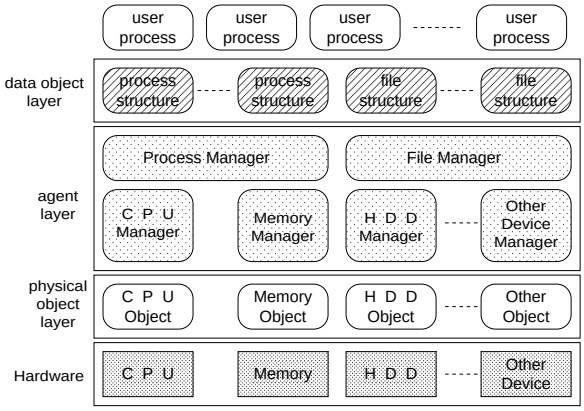

図 1　システム構成

いった計算機資源を抽象化するオブジェクトから構成される. エージェント層は, 物理オブジェクトやデータオブジェクトを操作するエージェントプロセスの集合である. データオブジェクト層は, 物理オブジェクト以外にエージェントが管理するデータ構造を抽象化したオブジェクトから構成される.

各計算機は、ネットワークで接続され、同一アーキテクチャの CPU を持つものとする。メモリサイズ、さらにディスクの有無などの周辺デバイスに関しては、それぞれの計算機で異なっていてもよい。各計算機は, システム全体を通して一意な計算機 ID によって識別される. 計算機 ID は, 固定ではなくシステムへの接続時に与えられる整数値である. また, 周辺デバイスには, その種類毎にデバイス ID を与え, 計算機 ID とデバイス ID の組合せによって, 位置透過にデバイスを指定することができる.

各種の計算機資源とデータ構造は、オブジェクトとして抽象化される。オブジェクトは、資源操作やデータ変更するためのインタフェースを持つ. エージェントは、カーネルを機能単位で分割し, それぞれを 1 つのプロセスとして実現したものである。複数のエージェントの協調動作によって、OS としての機能が実現される。エージェントは、オブジェクトを操作することで資源を間接的に操作し、高度な機能は、エージェントによって提供される。

エージェントとオブジェクトは, それぞれ計算機毎に一意な値を持つエージェント ID とオブジェクト ID を持つ. ネットワーク上では, 周辺デバイス

と同様, 計算機 ID とエージェント ID またはオブジェクト ID の組合せで参照することが可能である.

本システムにおけるプロセスは, 一般的なプロセスモデルに基づいている. 1 つの OS 上で複数のプロセスが動作し, 1 つのプロセスは 1 つのスレッドを持つ. プロセスはユーザ権限で動作し, プロセスは互いに保護される. 本システムの提供する機能を用いることで, プロセスはエージェントとして実現することが可能となる.

3 オブジェクトとエージェント

オブジェクト指向では、開発するアプリケーションが扱うデータや機能を、オブジェクトとして実現する。オブジェクト指向を用いることで、機能とデータのカプセル化やクラスの継承など、手続き指向にはない効果を得ることが可能である。

オブジェクトは、あるイベントを契機として処理を開始し、オブジェクト間でメッセージを伝達することで協調して処理を行う。そのため、システム全体が何らかのイベントに依存することになる。対象システムが巨大かつ複雑になるにつれ, イベントの種類と数も複雑かつ増加し, 適切なイベントを選択することが困難となる.

OS をオブジェクト指向で設計する場合、各種計算機資源はオブジェクトとしての定義が容易である。カーネルに関しては、それ自体を巨大なオブジェクトとして捉えることができるが、現実的でない。カーネルは、様々な資源を操作するという目的の集合体であり、ものとして捉えることができない。目的は、常に定まった静的なものではなく、状況によって変化する。オブジェクトは、構成が変化することのない静的なものであり、目的を抽象化するための概念としては不適切である。

そこで、本研究では、カーネルを自律的な概念であるエージェントによって抽象化し、それ以外をオブジェクトとして実現する。これにより, OS がイベントに依存せず自律的に動作することが可能となる. 以下、本章では、オブジェクトとエージェントの構造体、それらの実現手法について述べる。

3.1 オブジェクト

本 OS は、図 2 で示すように C 言語で記述されるため、オブジェクトはすべて構造体として定義

```
struct PSEUDO_CLASS{
   /* object identifiers */
   int ObjectID;
   int ObjectType;
   /* date structure present object condition */
   struct CONDITION{
                  ¦
                  ¦
               }Condition;
   /* functions for changing object's condition */
   void (*ChangeCondition)();
                  ¦
                  ¦
};
```

図 2　オブジェクト定義の擬似コード

される。オブジェクトは、オブジェクトを識別する変数、内部状態を表す状態変数と、状態を変化させるためのメソッドを持つ。

本 OS におけるオブジェクトは、大きく分けて以下の 2 種類に分類することができる。

- データオブジェクト
- 物理オブジェクト

データオブジェクトは、プロセス構造体やファイル構造体などデータ構造を抽象化したオブジェクトである。データオブジェクト自身は、管理対象となるデータのみを持ち、計算機資源やエージェントに直接影響を与える動作を行わない。例えば, プロセス構造体をデータオブジェクト化した場合、オブジェクト内部には、プロセス ID やコンテキストなどのプロセス構造体が持つデータと、それらを操作するメソッドが含まれる。

物理オブジェクトは、CPU、メモリ、周辺デバイスを抽象化し、抽象化の対象を操作するためのメソッドを提供するオブジェクトである。CPU を抽象化したオブジェクトは、オブジェクトの状態を変更するメソッドに加え、CPU を操作するためのメソッドも持つ。周辺デバイスの場合は、デバイスドライバに相当するメソッドを持つ。原則として、図 3 に示すように、1 つのデバイスに対して 1 つのオブジェクトが存在する。

物理オブジェクトの特殊な例として、メモリを抽象化した物理オブジェクトが挙げられる。メモリは、実際の参照は物理アドレスで行われるが、OS からは論理アドレスで参照される。このとき、MMU(Memory Management Unit) の提供

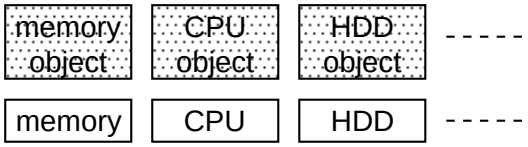

図 3　計算機資源の抽象化

するページング機能を用い、またセグメンテーションを利用して論理アドレス空間を構成する。各プロセスには、コード、データ、スタックの 3 つのセグメントからなる仮想アドレス空間が与えられる。このように、メモリに関しては、メモリ自体がすでに MMU によって抽象化されているため、1 つのオブジェクトとして抽象化することが難しい。したがって、本 OS では、メモリを次の 3 つのオブジェクトで管理する。

- 仮想メモリオブジェクト
- セグメントオブジェクト
- ページオブジェクト

仮想メモリオブジェクトは、1 つのプロセスに対して 1 つ生成されるオブジェクトであり、4GB の仮想メモリ空間全体を抽象化する。セグメントオブジェクトは、1 つのセグメントにつき 1 つ生成され、セグメントを抽象化する。したがって、1 つのプロセスにつき、コード、データ、スタックセグメント用に 3 つのセグメントオブジェクトが生成されることになる。ページオブジェクトは、ページをオブジェクト化したものであり、1 つのページにつき 1 つのオブジェクトが生成される。

メモリ管理の際には、プロセス毎に 3 つのセグメントで区切られた仮想メモリ空間がある。また、ページングにより、メモリ空間全体はページ単位で扱われる。ページには、物理メモリ上にマッピングされているものといないものがあり、これらを区別する必要がある。このように、メモリに関する 3 つの概念は、それぞれが互いに相関している (図 4 参照)。

本 OS では、メモリの概念における相関関係を、オブジェクトに適用することで、3 種類のオブジェクトによるメモリ管理を実現する。オブジェクト間の相関関係は、オブジェクトへの参照を持たせることで実現することができる。

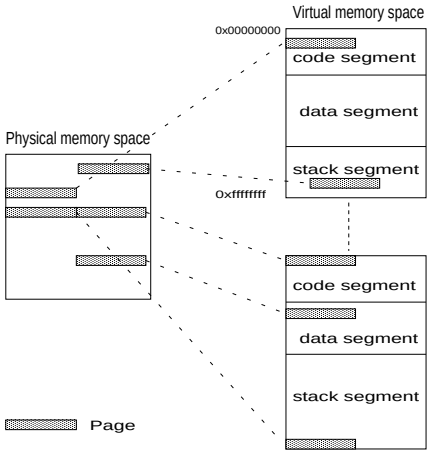

図 4　物理メモリと仮想メモリの対応

3.2　エージェント

本 OS では、カーネルとしての機能をエージェントとして実現する。カーネルは、機能毎に分割され、基本的に 1 つの機能を 1 つのエージェントで実現する。エージェントには、様々な種類がありそれぞれ特性が異なる。本研究におけるエージェントは、オブジェクトに以下の 2 つの特性を追加したプロセスである.

- 自律性
- 移動性

自律性により、エージェントは、外部入力の有無に因らず、能動的に処理を開始もしくは再開することが可能となる。例として、自律性をメモリ管理に適用した場合について考える。従来のメモリ管理手法では、物理メモリの空き領域が不足するとページフォルトが発生し、適切なページをページアウトする。メモリロックにより、特定のページをページングの対象外とする手法があるが、これはユーザによって明示的にページが指定されている必要があり柔軟とはいえない. エージェントが能動的に動作できる場合、ページフォルトが発生する前にユーザプロセスの重要度に応じて、重要でないページをページアウトして空き領域を確保したり、頻繁に参照されるページを動的にロック・アンロックすることが容易に実現可能である。

真の自律性を実現することは非常に困難である。そこで、本研究では、エージェントの動作が、基

本的に以下の順序で行われることに着目した。

(1) 管理対象となるオブジェクトの状態を監視する。

(2) 監視の結果得られた情報から、次に行うべき処理を決定する。

(3) 決定した処理もしくは外部から受け取ったイベントに対応する処理を行う。

最初の 2 つの手順は、行うべき処理を決定するために行われ、最後の手順は、決められた処理を行うのみである。すなわち、前者は能動的な処理であり、後者は受動的な処理である。このことから、エージェントを能動的な処理を行う部分とそうでない部分に分割することができる. 能動的な処理部分を 1 つのスレッドとし、受動的な処理部分は複数あるため、それぞれをスレッドとして実現する。このようにエージェントを内部で複数スレッド化することで、能動的なスレッドが常に状態を監視し, 状態に応じて受動的なスレッドを実行することで自律的な処理を実現する.

エージェントに移動性を持たせることで、エージェントは任意の計算機上に移動し、処理を続けることが可能となる。しかし, エージェントが単独で計算機間を移動することは困難である. 本 OS では、エージェントの移動性を実現するために、各エージェントに移動用の機能を提供する移動補助エージェントを用意する. エージェントの移動は, 次の手順で行われる.

(1) 移動するエージェントは, 移動補助エージェントに自身のエージェント ID と移動先の計算機 ID を渡す.

(2) 移動補助エージェントは, エージェントの実行継続に以下の必要なデータを移動先の計算機に移送し, 移動先のメモリ管理エージェントにメモリオブジェクトの作成を要求する.

- プロセス管理用オブジェクト
- 仮想メモリオブジェクト
- エージェント ID

(3) プロセス管理用オブジェクトを登録する.

(4) 移動先計算機上でエージェントの実行を再開する.

以上の手順により, エージェントは, 計算機間を移動しても処理を継続することができる.

4 分散システムとしての機能

4.1 分散システムの構成手法

従来の分散システムの構成は, 図 5(a) のようにサーバとなる計算機を中心としたスター型か, 図 5(b) のようにすべての計算機が接続されるメッシュ型のいずれかである. 図中の線は, 両端の計算機間でコネクションが張られていることを表す. すべての計算機は, 物理的に接続されているが, 通信を行う対象は図中の実線で結ばれた計算機間でのみ行われる.

スター型の構成は, サーバを中心に配置することで資源の集中管理が行い易く, また計算機の追加と切り離しが容易に行える. メッシュ型の構成は, 各計算機上の OS が協調してシステム全体の資源を管理するためサーバを用いずに実現することができる. しかし, スター型はサーバの障害や移動に弱く, メッシュ型はシステム構成が複雑となり, 柔軟性に欠ける.

本研究では, 柔軟なシステム構成を実現するために, 分散システムを図 5(c) に示すように構築する. 本構成では, サーバを用いずに計算機間を接続する. このとき, メッシュ型構成のようにすべての計算機間ではなく, 一部の計算機との間でのみ接続するように構成する.

図 5(c) では, 各計算機は一部の計算機とのみ相互に接続している. したがって, 資源などに関する情報の同期は接続先の計算機とのみ行われる. 図中の計算機 A の場合, B, E のみが同期する相手となる. このとき, 計算機 A, B, E は, サブシステムを構成していると考えることができる. このようにサブシステムを捉えた場合, 1 つの計算機は, 2 つ以上のサブシステムに含まれることになる. 例えば, 計算機 A は, B, E からなるサブシステムの他に, B, D と C, E からなる 2 つのサブシステムに含まれる. このように, 本研究で構成される分散システムは, 複数のサブシステムの集合体として実現される.

本 OS は, 単体で基本的な処理を行うことができ

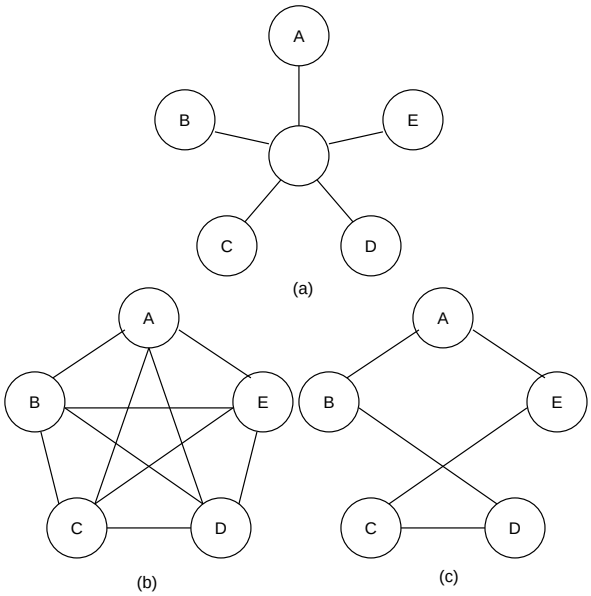

図 5　分散システムの構成手法

るだけの機能を持つ. そのため, 計算機間の同期を頻繁に行う必要がない. 同期を行うタイミングは, 最後の同期から一定時間経過したとき, 他計算機の情報が必要となったとき, もしくは他計算機から情報を要求されたときのみである. また, この同期は, サブシステム単位で行われる. システム全体の同期は, サブシステムでの同期が伝搬していくことで行われる. A, B, E からなるサブシステムが同期をとる際に, それぞれが属する別のサブシステムに関する情報も渡す. これを繰り返すことで, 各計算機はシステム上のすべての計算機の情報を得ることができる. ただし, このとき得られるサブシステム以外の計算機の情報は, 最新のものではない. しかし, 通常はサブシステム単位で協調するため, 特に問題はないと考えられる. この各計算機に関する情報は, オブジェクトとして管理される.

本システムは, システム上の任意の計算機から他の計算機を経由してすべての計算機を参照できるように構築する必要がある. また, 一部の接続が切断された場合でも障害が発生しないように考慮する必要がある. 本システムでは, 以下に示す手順で計算機の追加を行う.

(1) システムに追加される計算機は, システム上の計算機すべてに接続要求を発行する.

(2) 要求を受けた計算機は, 現在の接続状況から接続が可能か否かを判断し, 可能であればその旨を通知する.

(3) 接続可能な計算機が多数存在する場合は, その中からランダムに必要な数だけ選択し接続する. 少なければ, 一時的に必要なだけ適当な計算機に強制的に接続する.

計算機を切り放す場合は, 次のような手順をとなる.

(1) 切り離される計算機は, 接続先の計算機に切断要求を発行する.

(2) 要求を受けた計算機は, 切り離し処理を行い, 準備が完了したことを通知する.

(3) 通知を受けた計算機は, 接続を断ち切り離される.

(4) 残った計算機は, 接続数が十分であればなにも行わない. 不十分であれば, 追加手順を繰り返し, 十分な接続数を確保する.

4.2　他計算機上の資源操作

従来の分散システムでは, 他計算機の資源を利用することはできるが, 管理することができない. 分散 OS が行う資源管理の対象は, OS が動作する計算機上の資源のみである. したがって, 他計算機上の資源を使用する場合には, ユーザの要求を十分に考慮することができない. そこで, 本 OS では, 各計算機上に他計算機上の資源管理を補助するためのエージェントを配置する. このエージェントを用いた他計算機上の操作の手順を以下に述べる.

(1) 資源操作を行うエージェント A は, 他計算機の情報を持つオブジェクトから, 操作対象となる資源を抽象化するオブジェクトの位置を検索する.

(2) エージェント A は, 検索結果を基に, 資源の遠隔操作を補助するエージェント B に計算機 ID, オブジェクト ID, 呼び出すメソッド名と引数を渡す.

(3) エージェント B は, 指定された計算機上の操作補助エージェント C にオブジェクト ID, メソッド名, 引数を渡す.

(4) エージェント C は, 指定された通りオブジェクトを操作する. 返り値があれば, エージェント B を経由してエージェント A に渡される.

この機構を実現することで, 複数のディスプレイを位置透過にアプリケーションが操作することや分散共有メモリを実現することが可能となる.

5 関連研究

OS の構成手法としてマイクロカーネル [7] を採用することで, OS に柔軟性と拡張性を持たせることができる. マイクロカーネルを採用したシステム [1][5] では, システムサーバにより高度な機能を提供し, マイクロカーネルで基本的な機能を提供している. また, ポリシとメカニズムの分離の概念により, ユーザによるカーネルのカスタマイズが可能である. 本稿で述べたエージェントも, ポリシとメカニズムを分割している. 本研究では, ポリシを自律的に動作させることで, ユーザだけでなくエージェント自身がその機能を変更することが可能である.

単一の OS で複数の計算機を管理するシステムとして, Solelc[2] がある. Solelc では, 各計算機上で動作する抽象化層によってすべての計算機資源を抽象化し, カーネルが資源の位置に依存せずに動作することを可能としている. 本 OS では, 各計算機上で OS が動作するが, 複数計算機の資源を 1 つの OS で管理することが可能である.

アプリケーションに最適な実行環境を提供する手法として, カーネルをライブラリ化し, アプリケーションによる資源管理を行う手法がある. exokernel/ExOS [3] では, マイクロカーネルである exokernel 上でライブラリ OS である ExOS が動作する. ユーザは, 実行するアプリケーションに最適なライブラリを選択, もしくはバインドすることで, アプリケーション毎に特化した環境を実現する事が可能である. 本 OS は, 同じ種類でインタフェースの異なるエージェントを同時に動作させる事が可能である. また, 複数の計算機にまたがる資源操作を単一のエージェントで行うことが可能であるため, より分散環境に適しているといえる.

6 おわりに

本稿では, エージェント技術を用いて分散 OS を構築するための手法について述べた. エージェント技術を用いることによって, 自律的な資源管理を行うことが可能となる. また, 分散システム構築時には, システムを複数のサブシステムに分割することでサーバを用いずにシステムを構成することができるため, 対障害性を向上させることができる.

現在, 我々は, 本手法に基づいた OS の実装を行っている. 今後は, OS の実装を進め, 本手法の評価を行う予定である.

参考文献

[1] 毛利 公一, 大久保 英嗣: “マイクロカーネル Lavender の設計と開発,” 電子情報通信学会論文誌 D-I, Vol. J82-D-I, No. 6, pp. 730-739 (1999).

[2] 芝 公仁, 大久保 英嗣: “分散オペレーティングシステム Solelc におけるメモリ管理手法,” 情報処理学会論文誌, Vol. 42, No. 6, pp. 1460-1471, (2001).

[3] Dauson R.Engler, M. Frans Kaashoek, James O’Toole Jr.: “Exokernel: An Operating System Architecture for Application-Level Resource Management,” Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP ’95), pp.251-266 (1995).

[4] Yasuhiko Yokote, “The Apertos Reflective Operating System: The Concept and Its Implementation,” OOPSLA’92 Proceedings, ACM, pp.414–434 (1992).

[4] A.S.Tanenbaum, “分散オペレーティングシステム, ” 株式会社トッパン (1996).

[5] 乾和志, 菅原圭資, “分散 OS Mach がわかる本,” 日刊工業新聞社 (1992).

[6] 木下哲男, 菅原研治, “エージェント指向コンピューティング,” 株式会社ソフト・リサーチ・センター (1995).

[7] 前川 守 監訳: “オペレーティングシステムの先進的概念,” 丸善株式会社, (1990).

[8] Bertrand Meyer 著 二木 暑吉 監訳: “オブジェクト指向入門,” 株式会社アスキー (1990)