Vol. 36 No. 8　　情　報　処　理　　Aug. 1995

解　説

日本におけるオペレーティングシステム研究の動向

3.2 オブジェクト指向OS Apertos†

横　手　靖　彦††

オペレーティングシステム（OS）におけるオブジェクト指向技術に関して，Apertosシステムを例に解説する．まず，オブジェクト指向技術と“Separation of Concerns”と呼ばれるソフトウェア技術の重要性を議論する．次に，その技術をApertosではオブジェクトとメタオブジェクトの分離技術を開発して実現しているが，その詳細を議論する．本稿での議論はApertosシステムの場合が中心になっているが，一般的なオブジェクト指向オペレーティングシステムにもあてはまる．

1. はじめに

情報スーパハイウェイ構想は，ユーザの要求を多様化し，オペレーティングシステム（OS）の設計に新しい変革を要求している．たとえば，OSはマルチメディア，データベース，仮想現実感などの様々のアプリケーションを効率よく実行する必要があるが，その最適な実行はアプリケーションが知っている．既存の単一の固定化されたインタフェースでは，これらアプリケーションの最適実行は難しく，しばしばOS機能をアプリケーションごとに再実装し直す必要が生じている．一般に，これをマッピングジレンマと呼んでいる[1]が，OSとアプリケーションの協調によりこれを解消することができる．一方，OSは様々の種類のデバイスを効率よく管理する必要がある．ICカードによるプラグ&プレイ機能の実現から，FDDIやATMをはじめとする高速通信デバイスの管理，MPEGをはじめとする画像圧縮・伸長装置の管理，などとOSが管理すべきデバイスは年々増加している．

このような状況にあって，Apertosはこれら新しいアプリケーションのためのOSとして開発されている[2)~6)]．プラットフォームとしては開放型組込み機器の分野を想定している．ここでは，カムコーダ，VCR，TV，ケーブルネットワークのためのサーバ/セットトップボックス，ネットワークゲームマシン，コピー，FAX，電話，計測器，携帯可能計算機，などが単独で存在するのではなく，それらがネットワークでつながれた環境を考える必要がある．

アプリケーションの特徴は，開放型システムであり，異種システムであり，分散システムであるという点である．開放型システムであるということで，様々のアプリケーション，通信媒体，計算機がシステムに接続されることになり，多様なユーザの要求を満たすためにシステムの構成が変更されることになり，システムの将来に関する振舞いは予想が難しい．異種システムとしての側面から，システムには様々のアーキテクチャのプロセッサや，OSが接続されている．通信媒体も高速ネットワークから電話網まで様々である．また，アプリケーションも実時間性を要求するもの，耐故障性を要求するもの，他人に知られたくないもの，などの様々の性質を持ったものが存在する．分散システムとしての側面から，アプリケーションを構成するオブジェクトや，複数の様々のアプリケーションがシステム内に分散配置される．規模もマルチプロセッサやローカルネットワークの規模から，インターネット上に分散されたもの，ケーブルネットワークに分散されたもの，など様々である．さらに，上記の本質的な特徴のほかに，従来から言われているような実時間制約，制限された計算資源，などの特徴も有している．Apertosは，これらアプリケーションのプログラミングを簡単化し，移植性を増し，メンテナンス

† Object-Oriented Operating System, Apertos by Yasuhiko YOKOTE (Sony Computer Science Laboratory Inc.).

†† (株) ソニーコンピュータサイエンス研究所

を容易にするための基盤を提供することを目的とする．

2. オブジェクト指向技術と Separation of Concerns

Apertos ではユーザやプログラマに対して単一のシステムビューを与えるために，オブジェクト指向技術を導入した．オブジェクトは状態とその状態を変化させるメソッドをカプセル化したものであり，定義されたインタフェースのみを通してアクセスできる．オブジェクトのこの性質はオブジェクト間のインタオペラビリティや再利用可能性を増大させる．初期のオブジェクト指向 OS が，メモリセグメントの保護に主眼がおかれていたのに対し，最近のオブジェクト指向 OS はこの点に主眼がおかれている．しかし，このためには，オブジェクトプログラミングにおいて，その実行環境への依存をできるだけ避けた形でのプログラミングが必要になる．

たとえば，オブジェクトの粒度に関して次のような状況を考えることができる．図-1 には，プログラミング上の理想的なオブジェクトの粒度が定義されている．システムはこれを，実行 (activity)，記憶領域 (storage)，保護 (protection) の座標にそのオブジェクトに最適な形でマップする．ここで，実行はプログラム命令をプロセッサにどのように実行させるかに関する軸であり，記憶領域はデータをどのように保存するかに関しての軸であり，保護はオブジェクトをどのように保護するかに関する軸である．UNIX の場合には，マルチスレッドを持つプロセスにマップされる．この場合には，スレッドがオブジェクトの実行単位であり，プロセスに提供されている仮想空間がオブジェクトの記憶領域であり，プロセス間の保護機構がオブジェクト間の保護になる．プロセス間の通信はコストがかかるため，オブジェクトをプロセス内に複数配置することにより，これを軽減することができるが，この場合には，保護の軸位置が変化することになる．すなわち，オブジェクト間の保護は，コンパイラに任されるか，無視されることになる．この最適なマッピングはアプリケーションごとに異なるが，多くのシステムでは単一のマッピング方法しか提供してないため，プログラマがアプリケーションごとにこの制限を緩和するためのプログラムを記述することになる．このコードは一般に OS 依存であり，オブジェクトのインタオペラビリティや再利用可能性を弱めることになる．これを我々はマッピングジレンマと呼んでいる[1]が，既存の様々なシステムに見られ，そのための方策がこうじられている．たとえば，RT-Mach で用いられている方針と機構の分離による実時間プロセススケジューリング，ワシントン大で研究されたユーザレベルスレッド管理機構，Mach における仮想記憶システムのページ置換アルゴリズムのユーザレベルによる定義，x-Kernel に見られるネットワークプロトコルの再構成，などがある．

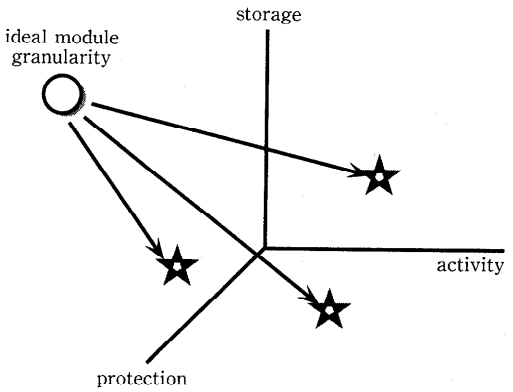

図-1　理想的なオブジェクトの粒度を実際の実装にマッピングする

また，アプリケーションプログラミングと OS との関係について，次のような状況を考えることができる．一般に，プログラマは与えられた問題を解くことに専念したい．OS の提供する API を用いてアプリケーションが記述されるが，マッピングジレンマ問題はこれを困難にする．たとえば，アプリケーションの振舞いに仮想記憶のページング方針が合わないためにそのオーバヘッドを減らすため，データのアクセスパターンを常に考慮してプログラミングを行うようであってはならない．また，オブジェクトを分散配置する場合は，そのためのコードがアプリケーションに埋め込まれていたり，ローカルオブジェクトとリモートオブジェクトを区別する必要があるのは望ましくない．さらに，リモートオブジェクト用のインタフェースを別の言語で記述する必要があったり，リモートオブジェクト呼び出し用のスタブを

明示的に記述するのも避けたい．これらは，OSの提供するAPIが，アプリケーションの要求を満たしていないからである．OSはちょうどブラックボックスと見なされ，その内部で何が行われているかは，外部には見えない．したがって，ブラックボックスがアプリケーションの要求を満たさないときには，その機能をスキップすることになる．たとえば，データベースシステムにおいて，OSの提供するファイルシステムが十分要求を満たさないときには，データベースシステムはファイルシステムを独自に持つことになる．これは，似た機能が重複して存在することになる．この状況は開放型組込み機器の場合には，資源制約の問題が生じる．そこで，ブラックボックスとしてのOSを取り除くことになるが，これでは，アプリケーションのインタオペラビリティや再利用性は低下する．

ここで，我々に必要なことは“Separation of Concerns”と呼ばれるソフトウェア技術である．すなわち，ソフトウェアモジュール/プログラムコードを上位の概念から独立にすることである．言い換えれば，プログラミングのための理想的なインタフェースによって記述されたオブジェクトと，それを実際のハードウェア/環境にマッピングする手順をできる限り分離することである．これにより，プログラムは単純になり，インタオペラビリティや再利用性を増すことになる．先のブラックボックスの例では，それを利用するものと実現するものに分けることに該当する．ブラックボックスの中身を巧く外部に見せることによって，それを利用するアプリケーションの独立性を維持したまま，最適な環境/機能を提供できるようになる．特に，開放型組込み機器の場合には，この技術は重要である．ここでは，従来からのプログラミング時の制約に加えて，先に述べたアプリケーションの特徴から，さらに複雑度が増している．実時間制約やメモリ制限からアセンブリ言語でアプリケーションを記述することが多く，開放型組込み機器の開発には限界が生じている．そこで，より高度の抽象化機能を備えたプログラミングモデル/言語を利用する必要があるが，性能が問題になる．しかし，“Separation of Concerns”技術によりこの問題を解決できる．

3. Apertosにおけるオブジェクトとメタオブジェクトの分離技術

Apertosにおけるオブジェクトは，状態とその状態を変化させるためのメソッド，そしてメソッドを実行するための仮想プロセッサをカプセル化したものと定義する．すなわち，並行オブジェクトモデル[7]を採用している．先のブラックボックスの例では，並行オブジェクトを実行するための機能がブラックボックスで提供される．Apertosではこの並行オブジェクトを，実際のハードウェア/環境にマップする手段を提供している．

このための我々の戦略は，オブジェクトとメタオブジェクトの分離技術を開発することであった．我々は，OSの構築とアプリケーションプログラミングに「メタ」の概念を導入した．すなわち，OSやアプリケーションは「ベース」オブジェクト，あるいは「メタ」オブジェクトで構成される．具体的に，オブジェクトとメタオブジェクトの分離技術とは次の3つの原理で説明される．

●すべてのオブジェクトは，メタオブジェクト空間と呼ばれるメタオブジェクトのグループによって与えられる実行環境内で実行される．すなわち，メタオブジェクトは，オブジェクトに対してその最適なインタフェースや，実行の意味付けを与え，そのための機構を提供している．

●メタオブジェクトは，オブジェクトである．

●オブジェクトはそのメタオブジェクト空間を必要に応じて別のメタオブジェクト空間に変更する．

最初の原理によって，オブジェクトは通常，OSによって決定される内部表現から独立になる．たとえば，オブジェクトの同一性はメタオブジェクトによって与えることができる．たとえ，メタオブジェクトがオブジェクトを構造体の配列として実現しても，別のオブジェクトからは識別子を持った1個のオブジェクトとして参照される．これは，そのオブジェクトへのアクセスが，そのメタオブジェクトを介して行われるからである．この意味から，メタオブジェクトとそのグループとしてのメタオブジェクト空間は，オブジェクトの外部からの見え方を定義している．実際には，記述子（descriptor）が導入され，オブジェクトのメタオブジェクト空間内での表現や，オブ

ジェクトのどの側面や性質が，どのメタオブジェクトで実現されているか，が記録されている．たとえば，オブジェクトが仮想空間上に割り当てられる場合には，その仮想空間を管理するセグメントメタオブジェクト名を記述子は保有することになる．

また，メタオブジェクトはオブジェクトの実行状態やオブジェクトグループの状態を表現するようになる．たとえば，単一のオブジェクト内部に複数のアクティビティを許すかどうかは，メタオブジェクトに任される．現在の Apertos の実装では，並行オブジェクトモデルを採用しているため，単一のアクティビティのみがオブジェクト内に許されるようにメタオブジェクトを実現している．これを，複数アクティビティモデル（マルチスレッドモデル）に拡張するには，記述子が複数のアクティビティを管理するメタオブジェクト名を保有するようにすればよい．従来のシステムの場合に，これら両方のモデルを使う場合には，複数アクティビティモデルを基本にすればよかった．しかし，その実現レベルでは，複数アクティビティモデルではオブジェクトとアクティビティの関係を維持しなければならず，単一アクティビティモデルの場合と比べてその実装は異なり，単一アクティビティで十分なアプリケーションにとっては余分なオーバヘッドとなる．

さらに，メタオブジェクトのグループによってメタオブジェクト空間が構成されているため，いくつかのメタオブジェクトをメタオブジェクト空間で共有することになる．たとえば，空間として異なった仮想記憶の管理方針を提供している場合で，オブジェクトのスケジューリング方針が同一の場合には，スケジューラメタオブジェクトを共有することによって，これを実現する．

2番目の原理によって，オブジェクトとメタオブジェクト空間の関係は，オブジェクトが与えられることによってそれを頂点とする階層構造を形成する．オブジェクトとメタオブジェクトはその階層構造内に存在する．従来のシステムを，この階層構造で考えるなら，アプリケーションと OS の2層構造を考えることができる．また，OS 内をシステムサービスを実現する上位層とデバイスドライバなどの下位層の2層構造で考えることもできるので，この場合には，アプリケーションを

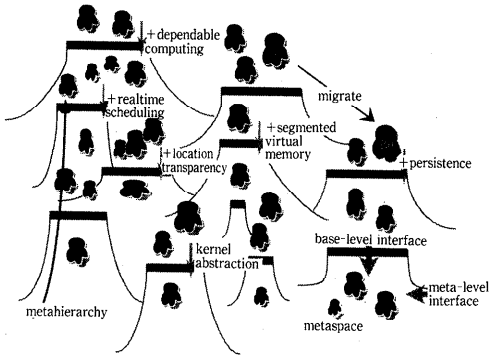

図-2　オブジェクトとメタオブジェクトの分離技術

頂点とする3層構造になる．

また，3番目の原理によって，“Separation of Concerns”のより高度の実現が可能になる．Apertos では，これをオブジェクトマイグレーションと呼び，そのプロトコルにオブジェクトの新しい環境（メタオブジェクト空間）への適応を含めている．たとえば，オブジェクトが永続性を備えるメタオブジェクト空間上に移動するときには，必要な内部表現の変更，一貫性検査機能の導入が，マイグレーションプロトコルの一部として行われる．たとえば，仮想空間上に割り当てられたオブジェクトの場合には，そのセグメントメタオブジェクトが表現形式を変え，永続性を扱うメタオブジェクト空間のメンバのメタオブジェクトに転送される．Apertos では，いくつかのプロトコルをあらかじめ用意しており，ユーザはそれを組み合わせて拡張できる．図-2 にこれらの様子を示した．

4. ま と め

現在，Apertos は R 3000 版の NEWS ワークステーションと PC 互換機の上で稼働中である．本稿で述べたオブジェクトとメタオブジェクトの分離技術は，コンテキストオブジェクトと MetaCore メタオブジェクトを用いた，メタレベルコンテキスト管理機構[3]によって実現されている．これらは，本分離技術を実現するための一種の仮想機械を定義しており，ネイティブコードにコンパイルされたオブジェクトのソースプログラムは，この仮想機械の命令を使って本分離技術を利用する．ここでは，カーネル内の実行を最小に抑えることに成功しており，MetaCore メタオブ

ジェクトにはその実行時間が厳密に予測可能な3つのサービスが定義されており，その大きさも，およそ2KB(テキスト）に抑えられている．

オブジェクトとメタオブジェクトの分離技術は，オブジェクト指向OSにおけるいくつかのモデル上の矛盾を自然に解決している．たとえば，すべてをオブジェクトとしたときに，そのオブジェクトをデバッグするデバッガはオブジェクトとなり得るか？ オブジェクト間のメッセージはオブジェクトか？ などが考えられるが，Apertosでは，これらをメタオブジェクトとすることによって，矛盾を解決している．Eden, Amoeba, Chorus, Springなどの既存のオブジェクト指向OSでは，一般に，すべてをオブジェクトととらえることはしていない．あくまでもOS(あるいはカーネル）はオブジェクトをサポートする機能を提供しており，OSの内部はオブジェクト指向の枠組みを利用していない．また，利用していたとしても，それは，アプリケーションの利用するオブジェクト指向の枠組みとは別のものである．

本稿では“Separation of Concerns”技術の，特に開放型組込み機器の分野での重要性を議論した．この考え方自体は新しいものではないが，OSがそれをサポートするための技術（オブジェクトとメタオブジェクトの分離技術）を備えることによって，従来のブラックボックス型システムにあった問題点を矛盾なく解決することに成功している．さらに，本分離技術は，将来の適応型システムや自己進化型システムの実現へ向けての重要な要素技術である．なお，Apertosの成果の一部はCD-ROMとSUP(Software Update Protocol）サービスを用いて公開されており，興味のある方は利用できるようになっている☆．

謝辞 多くの人がこのプロジェクトに貢献しているが，特に次の人に感謝したい．慶應大学/ソニーCSLの所真理雄教授，Xerox PARCのGregor Kiczales, John Lamping, Mark Weiser, Andreas Paepcke, Brian Smith, Dylan McNamee, Mike Dixon, Andy Berlin, カーネギメロン大学のChris Maeda，慶應大学の徳田英幸教授，伊藤純一郎氏，ソニーCSLのApertosプロジェクトの天満隆夫氏，藤波順久氏をはじめ研究諸員．

☆ http://www.csl.sony.co.jp/project/Apertosからもアクセス可能．

参考文献

1) Kiczales, G. and Lamping, J.: Operating Systems: Why Object-Oriented? In Proceedings of 3rd International Workshop on Object-Orientation in Operating Systems, The Computer Society of the IEEE (Dec. 1993).
2) Yokote, Y.: The Apertos Reflective Operating System: The Concept and Its Implementation, In Proceedings of Object-Oriented Programming Systems, Languages and Applications in 1992, ACM Press (Oct. 1992). Also appeared in SCSL-TR-92-014 of Sony Computer Science Laboratory Inc.
3) Yokote, Y.: Kernel Structuring for Object-Oriented Operating Systems: The Apertos Approach, In the Proceedings of the International Symposium on Object Technologies for Advanced Software (ISOTAS). Springer LNCS (1993). Also appeared in SCSL-TR-93-014 of Sony Computer Science Laboratory Inc.
4) Yokote, Y., Teraoka, F. and Tokoro, M.: A Reflective Architecture for an Object-Oricnted Distributed Operating System, In Proceedings of ECOOP'89 European Conference on Object-Oriented Programming (July 1989). Also appeared in SCSL-TR-89-001 of Sony Computer Science Laboratory Inc.
5) Yokotc, Y., Mitsuzawa, A., Fujinami, N. and Tokoro, M.: Reflective Object Management in the Muse Operating System, In Proceedings of the 1991 International Workshop on Object Orientation in Operating Systems, pp. 16-23, IEEE Computer Society Press (Oct. 1991). Also appeared in SCSL-TR-91-009 of Sony Computer Science Laboratory Inc.
6) Yokote, Y., Teraoka, F., Mitsuzawa, A., Fujinami, N. and Tokoro, M.: The Muse Object Architecture: A New Operating System Structuring Concept, ACM Operating Systems Review, Vol. 25, No. 2, pp. 22-46 (Apr. 1991). Also appeared in SCSL-TR-91-002 of Sony Computer Science Laboratory Inc.
7) Yonezawa, A. and Tokoro, M. ed.: Object-Oriented Concurrent Programming, The MIT Press (1987).

(平成6年7月28日受付)

横手 靖彦

1988年慶應義塾大学大学院理工学研究課博士課程修了．同年(株)ソニーコンピュータサイエンス研究所勤務．現在に至る．大規模分散システムやマルチメディアプラットフォームの研究・開発に従事．オブジェクト指向計算，プログラミング言語，プログラミング環境，オペレーティングシステムに関心を持つ．ACM，IEEE Computer Society，日本ソフトウェア科学会各会員．