TM-0061

逐次型推論マシン PSI のオペレーティング・システム

服部　隆，黒川利明，坂井　公，
辻　順一郎，近山　隆，
高木茂行，横井俊夫

May, 1984

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

逐次型推論マシンＰＳＩのオペレーティング・システム

服部隆、黒川利明、坂井公、辻順一郎、近山隆、
高木茂行、横井俊夫（ＩＣＯＴ）

１．概　要

ＳＩＭＰＯＳ (Sequential Inference Machine Programming and Operating System)[4]は、ＩＣＯＴで開発中の逐次型推論マシンＰＳＩ[1] のためのプログラミング／オペレーティング・システムである。ＰＳＩは論理型プログラミングの研究開発用ワークステーションであり、かつ知識情報処理システムのパイロット・モデルでもある。ＳＩＭＰＯＳはそのような目的に適したソフトウェア・システムとして設計・開発されている。

ＰＳＩのプロセサやメイン・メモリは、ＰＲＯＬＯＧをベースにした核言語第０版（ＫＬ０）[2] を高速かつ効率よく実行できるように設計されている。入出力機器として、ウィンチェスタ型ハード・ディスクやフレキシブル・ディスクの他に、マン・マシン・インタフェースの向上のためにビット・マップ・ディスプレイやオプティカル・マウスを装備している。また、ＰＳＩ間で資源や情報の共有を容易にするため、ローカル・エリア・ネットワーク（ＬＡＮ）もサポートしている。

ＳＩＭＰＯＳの主目標は、これらのハードウェア機器を有効利用し、優れたプログラミング環境を提供することである。ＳＩＭＰＯＳのプログラミング・システムは、

- プログラム／テキストの編集機能
- プログラムのデバッグ機能
- プログラムのライブラリ機能

などを総合的にサポートする。一方、オペレーティング・システム[5] はその下にあって、

- マルチ・プロセシング
- マルチ・ウィンドウ
- ファイルとディレクトリ
- ネットワーク通信

といった諸機能を提供する。

本論文では、ＳＩＭＰＯＳオペレーティング・システムの構築概念と構成、および実現例を示す。

２．構築概念

ＳＩＭＰＯＳの設計方針は、多種多様な機能の提供よりも、システムの簡潔性と拡張性に重きを置いている。その理由は、ひとつには限られたマンパワーとスケジュールではすべての機能を一度に設計・開発できないためであり、またシステムに明確な枠組と構造を与えることの方がより重要であると考えるためである。オペレーティング・システムを機能面からだけとらえて設計し、実現上の礎となる構築概念を疎かにすると、完成したシステムは当面の用には十分な機能をもつであろうが、将来の諸要求に応じた変更や拡張が困難になる恐れがある。ＳＩＭＰＯＳは今後とも引き続き拡充されるべきであり、このため我々はまずＳＩＭＰ

ＯＳの構築概念を設定し、その後に必要な機能を選定していくことにした。我々の取った構築概念はオブジェクト指向的アプローチによっており、それに基いたプログラミング言語としてＥＳＰ[3] を設計した。

２．１　オブジェクト指向的アプローチ

ＫＬ０のベースとなったＰＲＯＬＯＧはオペレーティング・システムの記述には、副作用とモジュール化の機能を欠いている。

まず、ＰＲＯＬＯＧは副作用を記述できない。（ＰＲＯＬＯＧの組込み述語であるassertやretract で副作用を表現できるが、それ自体をＰＲＯＬＯＧで実現しているわけでない。また、ＫＬ０の機械語命令としては複雑すぎると考え、ＫＬ０に含まれていない。）オペレーティング・システムは種々のシステム資源を管理・制御するために、状態の概念を必要とする。副作用でそれを容易に実現できる。ＰＲＯＬＯＧのタームで状態を表現し、状態変化の度に新しいタームを生成することでも実現できるが、効率を考えると適当でない。

もうひとつの欠点は、ＰＲＯＬＯＧにモジュール化の機能が欠けていることである。オペレーティング・システムのように、多数の人々で開発され、また利用されるソフトウェアではモジュール化機構が必須となる。

我々はこの２つの問題を一挙に解決する手段としてオブジェクト指向的アプローチを取ることにした。このアプローチが有効であることは、他のシステム[10][11]で示されている。

（１）オブジェクト

ＳＩＭＰＯＳでのオブジェクトは、外部的にはそれに許される（つまり、それが受けつける）操作の集まりで定義され、内部的にはクローズとスロットで定義される。クローズは操作手順を記述し、スロットは状態の保持のため値や他のオブジェクトを格納する。操作が実行されると、オブジェクトはスロットの内容を参照・変更しながらクローズを実行する。外部仕様が同じである限り、内部仕様を変更しても、オブジェクトの呼び出し側に影響が及ばないことに留意したい。

オブジェクトはＫＬ０の（ヒープ）ベクタで表現され、そのロケーションをオブジェクト・ポインタとして識別される。ベクタの先頭要素には操作の呼び出し時に実行されるクローズのＫＬ０コードが格納され、残りの要素はスロットとして使われる。

（２）クラス

同じ振舞いをするオブジェクトの集まりをクラスと定義する。クラス定義は、そのクラスに属するオブジェクトの原型（テンプレート）を与え、各オブジェクトはこのテンプレートから生成される。

（３）継　承

継承（インヘリタンス）により、いくつかのクラスから新しいクラスが定義できる。クラスａがクラスｂとクラスｃを継承すると、クラスａは自分で定義した操作に加え、クラスｂとｃの操作も含むようになる。内部的には、同じ操作を記述するクローズがＯＲ結合され、またスロットがすべて集められる。

継承がクラス間の“is-a”関係とすれば、“has-a ”（“is-part-of”の逆）関係も考えられる。クラスａがクラスｂ（正確には、クラスｂのオブジェクト）をもつとき、クラスａはクラスｂを持つ（“has-a ”）と言われる。我々はこの“has-a ”関係を導入せずに、“is-a”で代用する。例えば、クラスa-having-bはクラスａと、クラスｂのオブジェクトをもつようなクラスwith-bとを継承することで定義される。

（４）デーモン

オブジェクトがいくつかの要素をもつとき、そのオブジェクトの操作でこれらの要素も操作される場合がある。このような処理はそのクラスで定義すればよいが、もし多数のクラスが同じ要素をもつとすると、各々で定義するのはわずらわしい。むしろ、その要素をもつようなクラスをまず定義し、各クラスがこれを継承すればよいようにしたい。デーモンはそのために必要な手段を与える。

要素クラスがデーモン操作を、それを継承したクラスがプライマリ（通常の）操作を定義すると、実際にはデーモン操作がプライマリ操作にＡＮＤ結合され、この操作が呼ばれると、デーモンも暗黙の内に呼ばれることになる。デーモンには２つのタイプが用意される。“before”デーモンはプライマリの前に呼ばれ、“after ”デーモンは後に呼ばれる。

なお、“before”と“after ”だけでは必ずしも十分でない場合もある。

２．２　ＥＳＰ

ＥＳＰはＰＲＯＬＯＧベースのオブジェクト指向プログラミング言語であり、ＳＩＭＰＯＳの記述に使われる。ＥＳＰのクラス定義は次に形式で与えられる。

```
class   クラス名
     [マクロ・パンク定義]
has
     [継承定義 ;]
     {クラス・スロット定義 ;}
     {クラス・クローズ定義 ;}
[instance
     {インスタンス・スロット定義 ;}
     {インスタンス・クローズ定義 ;}]
[local
     {ローカル・クローズ定義 ;}]
end.
```

クラス定義は５つの部分から成る。マクロ部の説明は省略する。

継承部はこのクラスが継承する要素クラスのリストを与える。自クラスも含めて、これらのクラスのクローズ定義は、リスト内の順にＯＲ結合される。自クラスの位置も指定できる。

クラス部はクラス・オブジェクトを定義する。クラス・オブジェクトはこのクラスのオブジェクトを生成したり、クラスで共通の情報を保持したりするために

使われる。クラス部はさらにスロット定義とクローズ定義から成る。
インスタンス部はオブジェクトのテンプレートを定義する。そのスロット定義部ではオブジェクトがもつべきスロットを、クローズ定義部ではオブジェクトが受けつけるべき操作をそれぞれ定義する。
ローカル部はこのクラス定義内だけで呼べるＰＲＯＬＯＧ述語を定義する。

３．システムの構築

オペレーティング・システムが提供する機能は、クラスとして定義され、そのオブジェクトとして利用される。ここでは、どのようなクラスでオペレーティング・システムを構築しているかを概説する。

３．１　システムの構成要素

オペレーティング・システムが定義するクラスのオブジェクトをシステム構成要素と呼ぶことにする。オペレーティング・システムの基礎部分は、最小限の機能をもつ基本構成要素のみから作られる。追加機能はこれらの基本クラスの上に提供される。簡潔で一貫性の高いオペレーティング・システムにするためには、何を基本構成要素に選ぶかが重要である。まずシステムを表現する処理モデルを導入し、このモデルを実現するようなクラスを定義することにする。このモデルは、プログラム実行と入出力という２つの面をもつ。

（１）実行モデル

システムの実行主体をプロセスと呼ぶ。複数のプロセスが存在し得る。各プロセスは、その実行環境の下でプログラムで与えられた手続きを実行する。プログラムは、オブジェクトへの操作やＰＲＯＬＯＧ述語の呼び出しとして記述されている。必要ならば、プロセスは他のプロセスと通信し合って共同で処理を進めることもできる。また、オブジェクトの集まりも保持できる。

このモデルに対し、次のクラスが提供される。

- process　－　実行主体
- stream　－　オブジェクトの流れ（プロセス間通信）
- pool　－　オブジェクトの収納
- world　－　実行環境

また、ハードウェア資源を表現するために、processor やareaといったクラスが定義されているが、アプリケーション・プログラムで直接利用されることはない。

（２）入出力モデル

プロセスは、外界と通信するために入出力を実行する。ユーザと会話するためにウィンドウが、ディスク中にデータを保存するためにファイルが、そして他のマシンと通信するためにネットワークが、それぞれ使われる。入出力操作は、ウィンドウ、ファイル、およびネットワークを表現するオブジェクトへの操作として実現される。

ＰＳＩの各入出力装置もオブジェクトとして表現される。disk、keyboardといったクラスが定義されている。

３．２　システム構造

オペレーティング・システムはそのシステム構成要素を３層に分類している。つまり、入出力メディア・システム、スーパバイザ、およびカーネルである（図３．１参照）。入出力メディア・システムは論理入出力をサポートし[7][8][9]、スーパバイザは実行モデルをサポートし[6]、カーネルはハードウェア資源を管理し、物理入出力を制御する。下層で定義されたシステム構成要素は上層で利用されるが、その逆は許されない。このような階層的アプローチにより、システムの構造が明確になる。

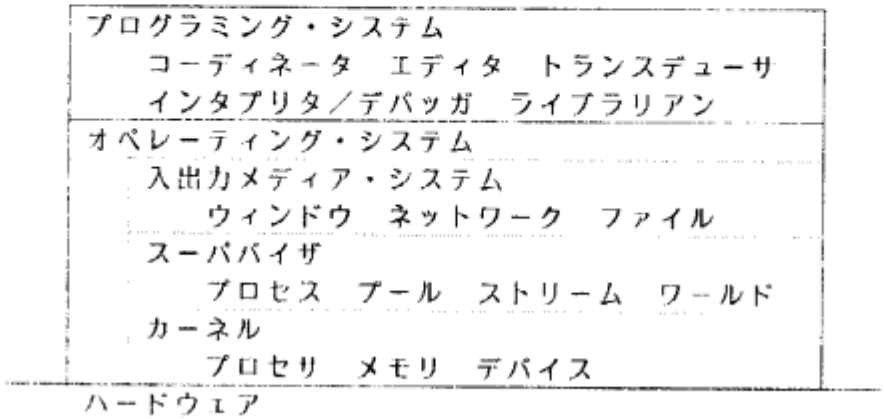

図３．１　ＳＩＭＰＯＳの構造

４．実現例

ここでは、ＳＩＭＰＯＳがオブジェクト指向的アプローチでどのように構築されるかを示すために、実現例をいくつか上げる。

４．１　単純なクラス

システム構成要素の簡単な例としてクラスfileを説明する。このクラスはディスク中のファイルをアクセスする機能を提供する。そのクラス定義の一部を下に示す。

```
class file has
attribute
        pool-of-files;
:create(Class, File) :- !,
        :new(Class, File);    %オブジェクトを生成する。
:make(Class, File, Pathname) :- !,  %新しいファイルを作る。
        :new(Class, File),
        :create-file(File, Pathname);
:open(Class, File, Pathname) :- !,  %既存ファイルを開く。
        :retrieve(Class, File, Pathname),
                %順路名で物理ファイルを検索する。
```

```
        :open(File);
instance
attribute
        region,
        opened := no,
        disposed,
        io-status;
:create-file(File, Pathname) :- ... ;
        %与えられた順路名で物理ファイルを作る。
:open(File) :- ... ;        %ファイルを開く。
:close(File) :- ... ;       %ファイルを閉じる。
:dispose(File) :- ... ;     %ファイルを捨てる。
end.
```

ファイルは上のようにopenやclose といった操作を受けつける。また、ファイルの状態を表わすために、region、opened、disposed、io-status といったスロットをもつ。一方、クラス・オブジェクトはmakeやopenという操作を受けつけ、開かれたファイルを管理するためにスロットpool-of-files をもつ。

なお、実際のクラスfileはas-permanent-object 、as-resource 、with-lock などのクラスを継承している。

４．２　継　承

継承を利用して付加機能をもつ種々のクラスを基本クラスから容易に定義できる。

（１）単一継承

ＦＳＰは多重継承をサポートしているが、単一継承でシステム構成要素（つまりクラス）を定義していくことが可能であり、また容易である場合も多い。単一継承では、新しいクラスはスーパー・クラスを継承し、それを改変することで定義すれる。この改変は、新しい操作（クローズやスロット）を追加し、既定義の操作をオーバライドすることでなされる。

継承の役割りは、仕様と実現では異なる。仕様では外部定義が意味をもち、操作がサブ・クラス（スーパー・クラスを継承したクラス）で同じであると考えられれば、その操作はスーパー・クラスで与えられるべきである。一方、その操作の実現はサブ・クラスごとに異なることもあり、そういう場合には各サブ・クラスで内部定義を与えることになる。

例えば、クラスbinary-file はクラスfileを継承して定義される。

```
class binary-file has
nature
        file;
instance
```

```
:read(Binary-file, Buffer, File-marker) :- ... ;
        %ファイル・マーカー位置のレコードをバッファに読込む。
:write(Binary-file, Buffer, File-marker) :- ... ;
        %バッファ中のレコードをファイル・マーカー位置に書出す。
:tap(Binary-file, File-tap) :- !,
        :create(#binary-file-tap, File-tap, Binary-file) ;
        %ファイルのタップを作る。
```

このクラスはクラスfileで定義された操作をオーバライドせずに、readやwriteなどの新しい操作を追加している。なお、これらの操作は外部的にはむしろクラスfileで与えられるべきものである。

（２）多重継承

多重継承ではシステム構築に異なるアプローチをとる。まず、必要な機能をいくつかの抽象クラスに分割し、それらから実体クラスを作る。実体クラスとはオブジェクトを生成できるクラスであり、抽象クラスは実体クラスに機能を提供するものである。ただし、適当な抽象クラス群を定めるのは必ずしも簡単ではなく、実際には単一継承と多重継承を併用する。

多重継承の使われ方を見るのには、ウィンドウ・システムが良い例となる。種々のタイプのウィンドウはウィンドウ構成用クラスを多重継承して定義される。例えば、ウィンドウに、ボーダ（境界枠）をもつもの、ラベル（名札）をもつもの、さらにボーダもラベルももつものがあるならば、基本ウィンドウ・クラス、ラベルをもつクラス、およびボーダをもつクラスを構成用クラスとして定義する。必要な機能をもつウィンドウはこれらのクラスを継承して定義される。

クラスbare-window はすべてのウィンドウ・クラスが直接・間接に継承すべき基本クラスである。

```
class bare-window has
nature
        with-display, as-inside, * ;
:create(Class, Parameters, Window) :-
        :new(Class, Window),
        :initialize(Window, Parameters),
        :start(Window) ;
instance
attribute
        width, height ;
:refresh(Window) :- !,
        :get-size(Window, Width, Height),
        :clear-rectangle(Window, 0, 0, Width, Height) ;
:initialize(Window, size(Width, Height)) :- !,
        Window!width := Width,
        Window!height := Height ;
end.
```

クラスwith-border がウィンドウのボーダを描く機能をサポートする。ボーダはウィンドウの外枠を示す４本の線から成る。

```
class with-border has
nature
        as-inside, * ;
instance
component
        border-x, border-y,
        border-area-width, border-area-height ;
attribute
        border-flag := on,
        border-width := 2 ;
:draw-border(Window) :- ... ;      %ボーダを描く。
after :refresh(Window) :-
        Window!border-flag == off, ! ;
after :refresh(Window) :-
        :draw-border(Window) ;
end.
```

クラスwith-labelはラベルを描く機能をウィンドウに与える。

```
class with-label has
nature
        as-inside, *, as-label ;
instance
:draw-label(Window) :- ... ;       %ラベルを描く。
:clear-label(Window) :- ... ;      %ラベルを消す。
:refresh-label(Window) :-
        :clear-label(Window),
        :draw-label(Window) ;
after :refresh(Window) :-
        :refresh-label(Window) ;
end.
```

なお、ラベルの実体はクラスas-labelに定義されているが省略する。
ボーダもラベルももつウィンドウは次のクラス定義で定義される。

```
class window-with-border-and-label has
nature
        bare-window, with-border, with-label ;
end.
```

上の例で述語refresh がデーモンを利用している。クラスbare-window がウィンドウ領域をリフレッシュするプライマリ述語refresh をもち、クラスwith-label とwith-border がそれぞれ述語refresh の“after ”デーモンをもつ。したがって、このウィンドウをリフレッシュすると、まずウィンドウ領域がリフレッシュされた後に、ボーダとラベルがリフレッシュ（描き直し）される。

5．結　論

ＳＩＭＰＯＳの設計はＩＣＯＴで１９８２年の秋に始まり、その年度末に機能仕様が作成された。１９８３年６月に機能の詳細設計とインプリメンテーションを担当するソフトウェア・グループが結成され、数回の改訂の後、１９８３年度末にクラス仕様書が完成した。

これと並行して、ＦＳＰの要求仕様が議論され、１９８３年の夏に結論が出された。それに続いて、言語の設計とインプリメンテーションが始まった。ＥＳＰ支援システムは現在開発用システムで稼働している。これには、ＥＳＰクロス・コンパイラ、ＥＳＰクロス・リンカ、ＥＳＰシミュレータなどが含まれている。

ＳＩＭＰＯＳはクラス仕様書に従ってＥＳＰでコーディングされ、ＥＳＰシミュレータの上でクロス・デバッグされている。１９８４年３月にＰＳＩがソフトウェア・グループにリリースされたので、プログラムの一部はＰＳＩ上で実機デバッグもされている。ＳＩＭＰＯＳの暫定版は内部利用のために９月末にリリースされ、第１版は本年度末にリリースされる予定である。

ＳＩＭＰＯＳで得た経験では、オブジェクト指向的アプローチは、仕様設計や実現の面で労力を軽減するのに有効であったと考えられる。しかし、このアプローチの欠点は、動的な実行方式に起因するオーバヘッドであり、まだＳＩＭＰＯＳが稼働していない現時点で、実行効率の面でも満足のいくものであるかを判断できない。ただし、従来のオペレーティング・システムの多くが実行時にパラメタのチェックを余儀なくされていたことを考慮すると、本来的にはそういうチェックを必要としない（オブジェクト呼び出しで置換えられる）ので、オブジェクト指向的アプローチでもシステム全体の効率を大きく悪化させないものと期待している。

謝　辞

ソフトウェア・グループ全員に対して積極的にＳＩＭＰＯＳ開発に携わっていただいていることに感謝したい。また、ファイルとウィンドウのサブグループには、本論文のクラス定義例を提供していただいた。

文　献

[1] S.Uchida, et al., "Outline of the Personal Sequential Inference Machine PSI", New Generation Computing, vol.1, no.1, 75-79 (1983).
[2] T.Chikayama, "KL0 Reference Manual", ICOT TR（予定）.
[3] T.Chikayama, "ESP Reference Manual", ICOT TR-044 (Feb. 1984).
[4] S.Takagi, et al., "Overall Design of SIMPOS", ICOT TR-057 (April 1984). The Proceedings of Second International Logic Programming Conference (July 1984)に掲載予定。

[5] T.Hattori, et al., "SIMPOS: An Operating System for a Personal Prolog Machine PSI", ICOT TR-055 (April 1984).
[6] T.Hattori and T.Yokoi, "The Concepts and Facilities of SIMPOS Supervisor", ICOT TR-056 (April 1984).
[7] T.Hattori and T.Yokoi, "The Concepts and Facilities of SIMPOS File System", ICOT TR-059 (April 1984).
[8] T.Hattori and T.Yokoi, "The Concepts and Facilities of SIMPOS Network System", ICOT TR (予定).
[9] J.Tsuji, et al., "Dialogue Management in the Personal Sequential Inference Machine (PSI)", ICOT TR-046 (Mar. 1984).
[10] A.Goldberg and D.Robson, "Smalltalk-80: the Language and its Implementation", Addison-Wesley (1983).
[11] D.Weinreb and D.Moon, "Flavors: Message Passing in the Lisp Machine", A.I.Memo No.602, MIT A.I.Lab. (Nov. 1980).