ICOT Technical Report: TR-180

TR-180

並列オブジェクト指向による大規模システムの実現
—SIMPOS共有ファイル・システムの開発—

真野忠志，斉藤慎一，小松光雄
（ピーコンシステム）
吉田かおる，近山　隆
（ICOT）

May. 1986

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

並列オブジェクト指向による大規模システムの実現

――ＳＩＭＰＯＳ共有ファイル・システムの開発――

吉田かおる　　真野忠志　　斎藤慎一　　小松光雄　　近山隆
（（財）ＩＣＯＴ）　　（ビーコンシステム（株））　（（財）ＩＣＯＴ）

概　要

SIMPOSは逐次型推論マシンPSI のためのオブジェクト指向型プログラミング／オペレーティング・システムである。本論文はSIMPOSの共有ファイル・システムの設計・開発を通して、並列オブジェクト指向により大規模システムを実現する場合の特質について述べている。

SIMPOS共有ファイル・システムは分散化システムへの適応を目的として設計された。外部記憶上に格納されたプログラム、データおよびオブジェクトの資源は、単一サイト内のみならず PSI間ネットワーク(PSI-NET) を介して複数プロセスからアクセスされる。

このような共有資源は並列オブジェクト・モデルを使って自然に抽象化できる。並列オブジェクトのもつメッセージ通信機構は競合するアクセスの排他制御を容易にし、内部のメッセージ管理機構により共有環境を自由に設定できるためである。本システムでは並列オブジェクト間の通信機構に非同期メッセージ交信を採用することにより、オブジェクトの独立性を高めることができた。また多重継承機構により、実装規模を小さくして機能の拡充・拡張が容易に図れたが、一方システム保護の立場から検討すべき課題が残された。

したがって（並列）オブジェクト指向はモデリングの道具としてのみならず、システム開発の生産性・保守性を高める有効な手法であることを報告し、問題点についても述べている。

1．序論

逐次型推論マシンPSI は単体のパーソナル・マシンとして機能するばかりでなく、LAN に結合され、さらにゲート・ウエイを介してネットワーク(PSI-NET) を構成するワーク・ステーションでもある。それぞれのPSI に外部記憶やプリンタなどの入出力機器が設備され、外部記憶にはプログラム、データおよびオブジェクトといったソフトウエア資源をファイルとして格納している。ネットワークに結合された PSI間でこれらの機器を共用したり、ソフトウエア資源を格納・転送することができれば、システムはさらに幅広く利用されるだろう。特に、分散環境下において共同でソフトウエアを開発したり応用システムを運用する場合、このような資源の共有は必要不可欠である。

SIMPOSは PSIのために開発されているオブジェクト指向型プログラミング／オペレーティング・システムである。そのプロセス管理はマルチ・プロセッシングを支援しており、他のPSI からのファイル入出力要求をファイル・サーバ・プロセスが受取ると、他のユーザ・プロセスと同様にそのサイト内のファイル・システムへ要求を出す。単一サイト内では要求の由来に関わりなく統一的に処理される。

SIMPOSの旧版ファイル・システムは単一サイト・単一プロセスからのアクセスを仮定したものであった。本論文で述べる共有ファイル・システムは分散環境に適応するように再設計したローカル・ファイル管理システムである。資源の共有環境を支援するため並列オブジェクトの概念を導入している。並列オブジェクトは実世界を反映しやすく、そのメッセージ通信機構により競合するアクセスの排他制御を容易に実現でき、内部のメッセージ管理機構により共有環境を自由に設定できるためである。

また一般にファイル入出力はユーザに複雑な操作を要求しがちである。そこでシステムの機能を拡充し、できるだけ簡易なユーザ・インタフェースを提供したい。多重継承機構によるクラスの構造化はこのような

機能の拡充・拡張を可能にする。複雑な機能を基本的な機能の組合せで表現できる上、その機能に対するシステム全体の実装規模が小さくなるので、虫の所在が局所化し生産性・保守性は向上する。このように継承機構により構築・拡張作業は容易になるが、完成したシステムを保護する立場からは問題が生じる。開発者と利用者との間に境界がないので破壊につながるということである。そこで境界を設けるしくみが必要となる。

さらにシステム更新の際にまず要求されることはユーザ・インタフェース・レベルでの旧版との互換性である。この問題はオブジェクト指向が最も得意とするものである。オブジェクトの機能（仕様）はそれへの操作群で定義でき、互換性とは操作群の形式を保つことを意味する。機能を拡張するには新たに操作を追加すればよいし、同じ仕様で機能を変更するには操作本体を実装しなおせばよいからである。

その結果、新ファイル・システムは短期間にして実現できた。今までに開発した応用プログラムにも新たな開発にも同様に利用されている。これはオブジェクト指向モデルの有効性を示すに充分な材料であろう。

2. SIMPOSでのオブジェクト

まず、SIMPOSの記述言語ESP のオブジェクト指向実行メカニズムとSIMPOSにおけるその利用形態について述べ、SIMPOSでのオブジェクトの概念を明らかにする。

一般に、オブジェクトの概念はデータの抽象化と制御の抽象化が一体となって生れてきた。オブジェクトは内部状態である属性（スロット）群とそれに対する操作（メソッド）群で定義される。オブジェクトへのアクセスはすべてこれらの操作を通して行なわれ、オブジェクトは内部データを保護するカプセルである。データのみの抽象化とはオブジェクトが自己完備である点で異なる。前者の場合、抽象化データを制御するコードが外部に分散するが、後者の場合はオブジェクトの内部ですべて制御するのでコードは分散しない。オブジェクトが実行環境を持たない場合、従属オブジェクトと呼ばれる。

オブジェクトの特徴づける属性群および操作群の枠組はクラスで定義される。クラスの継承とは親クラスからの枠組の共有を意味する。特に複数クラスを親クラスとする多重継承機構は、基本機能の組合せから複雑な機能を容易に実現させる。

狭義のオブジェクトはさらにその独立性が加味される。オブジェクトは独自の実行環境をもつプロセスである。オブジェクトへのアクセスは同期機構を備えた通信経路にメッセージとして送られ、オブジェクトはそれ自身のモニタとして機能する。これは独立オブジェクトと呼ばれる。競合するアクセスはメッセージの通信経路上で順序づけられるので、内部は排他制御から解放される。

SIMPOSはオブジェクト指向の論理型言語ESP で記述されているとともに、ESP プログラミング環境へと結合している。ESP ではメソッドが述語であり、スロット値設定などは核言語の組込述語で吸収し、論理と副作用の融合した世界を築いている。多重継承を実現している上、継承されるメソッドには選択肢を追加したり前デモン、後デモンを結合できるため、その組合せは階層的、差分的制御を可能にしている。ESP は論理型プログラミングの簡素性と多重継承によるオブジェクト指向のすぐれた記述力をその特質としている。

言語としてオブジェクトの独立性は明示していない。SIMPOSにおける一般クラスのオブジェクトは従属オブジェクトに相当する。アクセスが競合する場合、各メソッドにロック機構を埋込み排他制御している。

後者の独立オブジェクトはオペレーティング・システムのレベルで提供している。クラスprocess およびその系統クラスのプロセス・オブジェクトがそれであり、従属オブジェクトに実行環境を提供する。プロセス間の同期・通信機構を提供する種々のクラスが用意されており、通信経路は静的に設定している。

このようにSIMPOSは擬似並列環境を備えた逐次システムである。

3．基本設計

一般にマルチ・プロセッシングを支援するオペレーティング・システム核はその主な役割がプロセス間の同期と排他にあり、最小単位のセマフォによる同期機構からストリームなどの通信機構に及ぶ。このようなプロセス周辺の問題解決はそのまま並列オブジェクト・モデルに適用される。

SINPOS共有ファイル・システムを開発する第一目的は“ファイル共有環境の支援”である。

システム開発の立場では、複雑な共有環境を安全に効率よく制御することが必須である。複雑さの主要部分は共有資源の排他制御であり、これをできるだけ簡素なモデルで容易に実現したい。

ユーザの立場では、ファイル入出力操作ができるだけ簡単であってほしい。物理的な格納形式を意識したり、煩しい準備操作を施すことは避けたい。このため、システム内の機能を強化すべきである。また旧版までに開発した応用プログラムはそのまま利用したい。つまりユーザ・インタフェース・レベルで旧版と互換性を保つことが望まれる。

このような目的を達成するには、以下のような方針でオブジェクト指向機能を適用するのが良い。

(1) 共有資源は独立オブジェクトとする。

メッセージ通信によるモニタ機構がもたらす利益は、外界からの介入が一通信経路に集約されるため内部は排他制御から解放される点である。もしセマフォによるロック機構で従属オブジェクトを固めた場合、すべての操作（メソッド）に注意を払う必要があり、クラス継承で追加・結合した操作が穴となりかねない。そこで外部記憶上のファイル実体を独立オブジェクトで抽象化し、これを物理ファイル・オブジェクトと呼ぶ。

次に資源の共有・専有に関する環境の規約を設定しなければならないが、制御機構はその規約にとらわれないものにする必要がある。これはメッセージの受取とサービスを切分けることにより、メッセージ管理で吸収できる。また共有環境では互いに待ちに入るデッドロック状態がありうるため、この解消法を用意しなければならない。デッドロック要因を削除するにはメッセージの追越しが必要である。これはメッセージに優先順位を設けることで統一的に実現できる。

(2) 共有資源へのアクセス単位もオブジェクトとする。

資源アクセスに際し、その行為者との権利関係に応じてアクセス可否を決定する保護機構を組込む必要がある。物理資源には行為者のアクセス権を提示し、行為者には資源アクセスの操作を提供するというケーパビリティ機構を用いると保護機構を簡素で統一的に実現できる。そこで、このような論理的なアクセス単位をオブジェクトで実現し、論理ファイル・オブジェクトと呼ぶ。

(3) 制御を司どるものはオブジェクトとする。

オブジェクトの自己完備性を利用すると、制御はオブジェクト自身が決定してくれる。例えば、受信したメッセージの内容を加工してそのメッセージを返還する場合、戻り先をメッセージに問合せてそこへ送信してあげるのではなく、メッセージ自身に帰らせたい。メッセージがオブジェクトであればよい。

(4) 多重継承機能を活用する。

共通に利用する機能を部品クラスとして抽出すると、システム全体の実装規模は縮小する。また機能を意味的に分解することにより、検証の単位は小さくなる。各部品クラスが保証されれば、継承クラスでは組合せの相互作用を考慮することに集中できる。例えば、再利用するオブジェクトはシステム全体で何種類もある。再利用の機能を提供するクラスを提供し、これを継承してもらうだけで同様の機能の分散化を防ぐことができる。

ユーザ・インタフェースの互換性と機能拡充は、多重継承機構を利用して論理ファイルのクラス構成を工夫することで同時に実現できる。

現在、SIMPOSのプロセス環境には制限があり、並行に走ることのできるプロセスすなわち独立オブジェクトの数は有限個（６３個まで）である。この制限を考慮して、以上の基本方針に次のような変更を加えた。

(1)'物理ファイルの数は未定であるので従属オブジェクトとし、ファイル・マネージャ・プロセスをすべての物理ファイルのモニタとする。さらに、論理ファイルと物理ファイル間の通信経路は論理ファイルとファイル・マネージャ間の通信経路で共用して、モニタリングの負荷を軽減する。

(2)'論理ファイルの数も未定であるので従属オブジェクトとし、それぞれの主体はユーザ・プロセスとする。

以下に共有資源のモデルと本ファイル・システムのシステム構成図を示す。

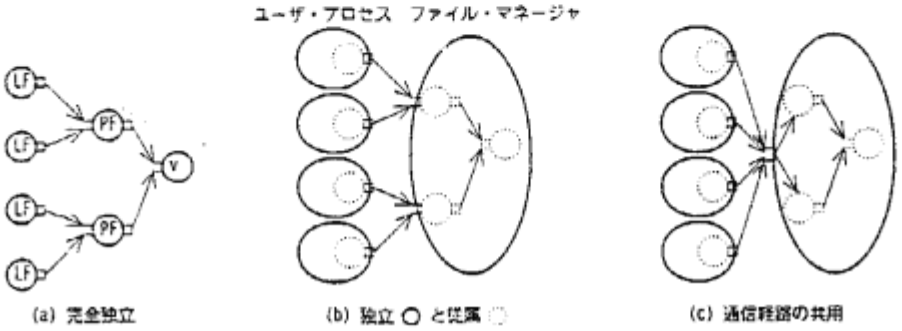


図 3-1　共有資源のモデル化

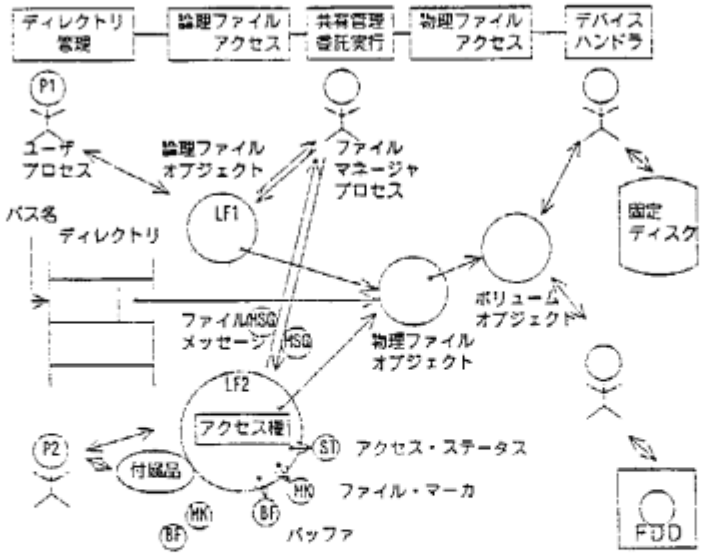


図 3-2　SIMPOS共有ファイル・システムの構成図

4．メッセージ通信機構

本システムではオブジェクトの独立性を高めるため、論理ファイル（ユーザ・プロセス）と物理ファイル（ファイル・マネージャ）の交信に非同期メッセージ通信機構を採用した。

4.1 非同期メッセージ通信

要求メッセージの送信と、それに対する返答メッセージの受信が独立である場合、これを非同期メッセージ通信と呼ぶ。要求メッセージを送信したオブジェクト（プロセス）は任意の時点でその返答メッセージを受信することができ、その間に別の操作を行なうことができる。別の要求メッセージをまた送信する操作も含む。

このように受信順序を送信順序に独立にするためには、送信事象と受信事象を関連づける手段が必要である。本システムでは、ファイル・メッセージがそれぞれの認識番号（ID）を持ち、論理ファイルが受信時にその認識番号をキーとして受信チャネルをアクセスすることで実現した。なおファイル・メッセージは再利用管理され、その認識番号はメッセージの確保（生成または再利用）時に与えられ、返還時に無効となる。

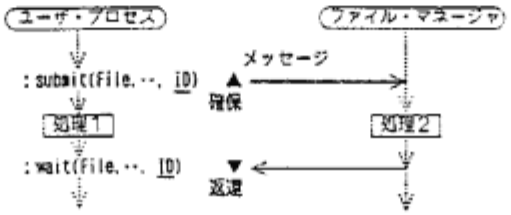


図 4.1 非同期メッセージ通信

本システムでは文字入出力を簡易にするためのユーティリティ・クラスを提供しているが、そこではこの非同期メッセージ通信をダブル・バッファリングに利用している。その結果、同期メッセージ通信のみでの実装に比べて平均２～３割の性能向上が得られた。

4.2 同期メッセージ通信

要求メッセージの送信直後に返答メッセージの受信を行う同期メッセージ通信は内部で非同期メッセージ通信に展開している。

5．メッセージ管理機構

本ファイル・システムでは次のような共有環境を設定し、これをオブジェクト内のメッセージ管理機構で制御している。

5.1 共有環境の規約

共有の対象となるのはファイル全体である。したがって論理ファイルの単位で制御すればよい。

共有状態はファイルのオープン／クローズ操作により遷移する。排他しうる事象はオープン操作であり、４種類のオープン・モードを用意している。クローズ操作でその排他状態を解除する。逆に、排他されうる事象はオープン操作と出力操作である。その関係を以下に示す。

表 5.1 論理ファイルのオープン排他関係

物理ファイルの共有状態	論理ファイルのオープン・モード			
	input 入力共有／出力禁止	shared 入力共有／出力共有	output 入力共有／出力専有	exclusive 入力専有／出力専有
closed	○	○	○	○
input	○	×	×	×
shared	×	○	○	×
output	×	○	×	×
exclusive	×	×	×	×
操作制限	出力禁止	outputモードの時、出力排他		

5.2 メッセージ管理

規約に基づく環境はメッセージに対するサービス順序により制御される。

(1) 優先順位とデッドロック解消

サービスには、優先と通常の２種類の順位があり、メッセージに指定されている。

- 優先メッセージ

  マネージャ・チャネルから受信後、直ちに処理される。

- 通常メッセージ

  順序制御の対象となる。受信時にサービス可能ならば、その処理を行う。一方、現在、排他要因がありすぐにサービスできない時は、対応する物理ファイルのメッセージ・キューにメッセージを追加する。

  ただし、応答性を上げるために、排他状態にあった場合のメッセージ取扱方法をメッセージに指定できるようにしている。その取扱方法には“排他解除待ち”と“要求取下げ”の２種類があり、“要求取下げ”の場合はメッセージを直ちに返信する。

優先順位はオブジェクトの状態検査やデッドロックの解消に役立っている。デッドロックとはメッセージが通信経路（メッセージ・キューも含む）上に詰った状態である。メッセージにサービス順位がなく均一である場合、デッドロックの要因を取除くためにまたメッセージを送れば状況はさらに悪化する。つまり、メッセージの流れを制御するためのメッセージは別の通信経路から来る必要がある。優先順位は論理的に別の通信経路を設定することを意味する。

デッドロックの解消はファイルの後始末操作で吸収している。ファイルの後始末操作の内容はクローズ操作のそれと類似するが、優先メッセージである点が異なる。デッドロックを起した場合そのユーザ・プロセスを殺すとそれの所有するファイルは資源管理において自動的に後始末され、排他状態は解除される。

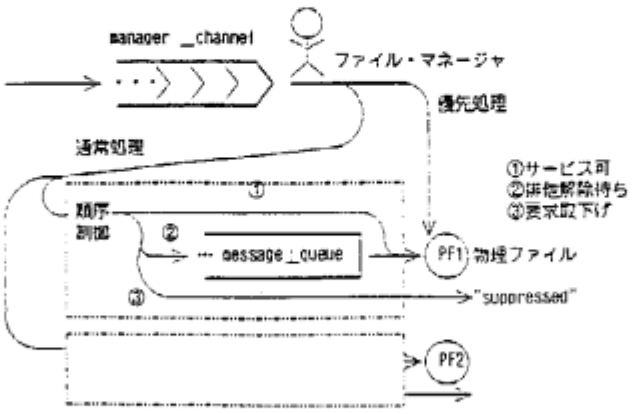


図 5.2-1 メッセージ管理

(2) 到着順制御

サービス順序は、オープン・モードの静的な関係のみならず、メッセージの到着順による動的な状態により決定する。すなわち規約と因果関係をもとにして排他状態は連鎖する。連鎖要因として、自分（論理ファイル）の排他状態が自分自身を抑制する場合と他の排他状態が自分を抑制する場合の２種類がある。前者は非同期メッセージ通信を、後者はオープン・モードの組合せを採用したことから生じる現象である。これらの因果関係は各物理ファイルのメッセージ・キューで管理している。

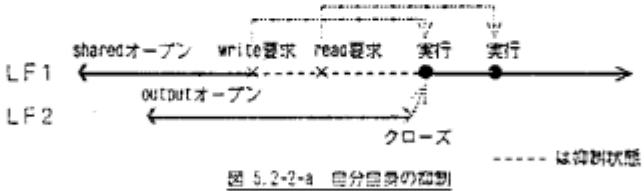


図 5.2-2-a 自分自身の抑制

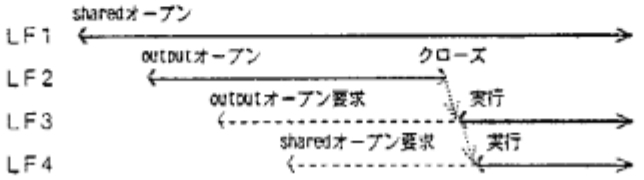


図 5.2.2-b 他からのオープン抑制

6. 自己完備性

自己完備性とはオブジェクトがそれ自身の内部状態を制御する知的実体であることを意味する。これは論理ファイルの高機能化やシステム全体の制御の簡素化に役立っている。その効果的な例を以下に示す。

例 6-1 内蔵道具の動的生成

ユーザとのインタフェースである論理ファイルは入出力に必要な道具としてバッファおよびマーカを内蔵する（次節の例１参照）。入出力操作にはこれらの内蔵する道具オブジェクトも外部に用意した道具オブジェクトも同様に利用できる。内蔵の道具を用いない場合にもそれを準備することはオーバヘッドを伴なう。また入出力データの大きさにとらわれずに操作したい。すなわち内蔵する道具は必要時に生成し、バッファは自動的に伸縮してほしい。このような動的生成機能や自動伸縮機能をオブジェクト自身が判断して行なうことにより、ユーザは繁雑な準備操作から解放され、オブジェクトも無駄に生成されなくなる。

メソッド :buffer(File, ^Buffer) はファイルの内蔵バッファ・オブジェクトの問合せ機能を果すとともに、メソッドが最初に呼出された時にバッファ・オブジェクトを生成している。内蔵バッファを利用する入出力メソッドは内部的にこのメソッドを呼出している。また入出力操作の内部ではバッファへのデータ格納のためのメソッド :put _data(Buffer,String) を呼出すが、ここではバッファ・オブジェクトが文字列の長さに合せて自らバッファ領域を拡張している。

7．多重継承

本ファイル・システムのクラス構成図を付録１に示す。

多重継承機構により、複雑な機能を基本的な機能に分解することができる。これは検証を容易にするだけでなく、オブジェクトの役割が多面化し、結果として実装規模（コード量）の減少につながる。興味深い使い方を以下に示す。

例 7-1　再帰的構造

簡易操作のためのユーティリティとして論理ファイルに付属品を取付けられるようになっている。ファイル・アクセサ・オブジェクトはその１種類であり、内部にバッファ・オブジェクト（データ格納庫）とマーカ・オブジェクト（位置情報）を装備し、これを利用した直接データ入出力や順次アクセス機能を提供している。論理ファイル・クラスはこのファイル・アクセサ・クラスを継承しており、論理ファイルがそれ自身への付属品であるという再帰的構造を形成している。したがって、あえて付属品を取付なくとも論理ファイルがすでに付属品の機能を提供する。（付録２参照）

例 7-2　参照と継承

クラスvolumeはクラスdevice_portを参照ではなく継承している。つまりボリューム・オブジェクトはデバイス・ハンドラとの通信経路を保持するのではなく、それ自身が通信経路となっている。これにより、メッセージ返信操作を直接ボリュームに対して行なうことができる。参照で実現した場合、その通信経路を取出す操作を外部で行うか、通信経路への操作群をボリュームで再定義するようになるため、コード量が増加する。

8．開発過程および性能

SIMPOS共有ファイル・システムは、ファイル共有環境の支援を主たる目的として従来のファイル・システム［小松84］から再設計・再構築された。ユーザ・インタフェースの点で従来とほぼ互換性を保つように設計されており、ユーザのためにユーティリティとして提供している一部のクラス群は旧版のものをそのまま活用し、それを除いた内部制御部分が実際の開発部分である。システム全体がオブジェクト指向論理型言語ESP で記述されている。SIMPOSには、オブジェクトを辿ったりメソッド単位のスパイ機能をもつデバッギング・ツールが整備されていることも作用しており、その開発コストは設計１ヶ月、実装・デバッグ３ヶ月の約１０人月と非常に低い。これはオブジェクト指向モデルがシステム開発の有効な道具であることを裏づける事実である。

以下に現版での実装規模と性能を示す。

デバイス・ハンドラとの複数回のプロセス切替が性能低下に影響しており、このインタフェースを変更し性能改善を図る予定である。

表 8-1　SIMPOS共有ファイル・システムの実装規模

	内部制御部分	システム全体*
クラス数	65	74
メソッド/述語数	1142	1428
クラス・メソッド	110	132
インスタンス・メソッド	651	681
ローカル述語	381	615
行数	約10K	約15K

(*) デバイス・ハンドラおよびディレクトリ管理は除く。

表 8-2　１ページ入出力操作の性能

（単位はmsec）

	cache on	cache off
ファイル・システム内		
①入出力内部制御	15	15
論理ファイル内	1．5	
プロセス切替と排他制御	9	
物理ファイル	3	
ボリューム	1．5	
②ページ初期化時（出力）	40	
デバイス・ハンドラ内		
③定常時（プロセス切替含）	6	22
④ページ初期化時（出力）	77	

（注）出力操作は :write/3 、入力操作は :read/3 で測定した。

9．結論

本論文では、ファイル共有環境を支援するためのローカル・ファイル管理システムの設計・開発を通して、（並列）オブジェクト指向がシステム開発にもたらす効果を探ってみた。実世界の複雑な制御はオブジェクト間のメッセージ通信・管理機構により簡素にモデル化される。開発の生産性・保守性は極めて向上する。

一方、システムを完成した後、システムを保護する立場からはオブジェクト指向特有の問題が浮び上ってくる。

- メッセージのコピー問題

  現在、論理ファイルが要求メッセージを送る場合、メッセージの内容はコピーせずそのまま送っている。バッファ・オブジェクトやマーカ・オブジェクト等がその内容であり、コピーとは新オブジェクトの生成のみならず、旧オブジェクトの保持する内容、例えばバッファ領域のコピーまでをも意味する。コピー方式を採用すると、構造化されたオブジェクトの場合、コピーが表層オブジェクトだけに止まらないからである。もしメッセージにロック機構を配備するとしても、コピー同様に表層オブジェクトに止まらない。

  本システムでは非同期メッセージ通信にもとずく入出力操作を提供しており、メッセージ送信後ユーザ・プロセスが走れるため、既に送信したメッセージ中のオブジェクトを操作可能であるという危険性を残している。このような問題への対処はユーザの責任としており、必要ならば送信前に内容のコピーを行ってもらう方針をとっている。

- クラス継承問題

  オブジェクト指向の特徴の一つに、クラスの階層性がある。SIMPOSの環境下ではOSを構成するクラスはユーザにも提供され、さらに継承可能である。このようにユーザとOSとの間に境界がない開放型OSの場合、ユーザが容易に各々のプログラミング環境を拡大できるが、システムの安定化は容易ではなく重要な問題である。

  ファイル・システムは外部記憶という外界と接触しており、既に格納されたファイルのクラス再定義に伴い、外部記憶全体の検索が必要となる。現在、このような探索機能は行っておらず、クラス再定義を制限している。OS内のクラス管理に関する今後の検討課題である。

また本システムは擬似並列でしかも独立オブジェクトと従属オブジェクトの混在する環境であった。すべてが独立オブジェクトである環境では次のような新たな問題が生じる。

- 優先制御

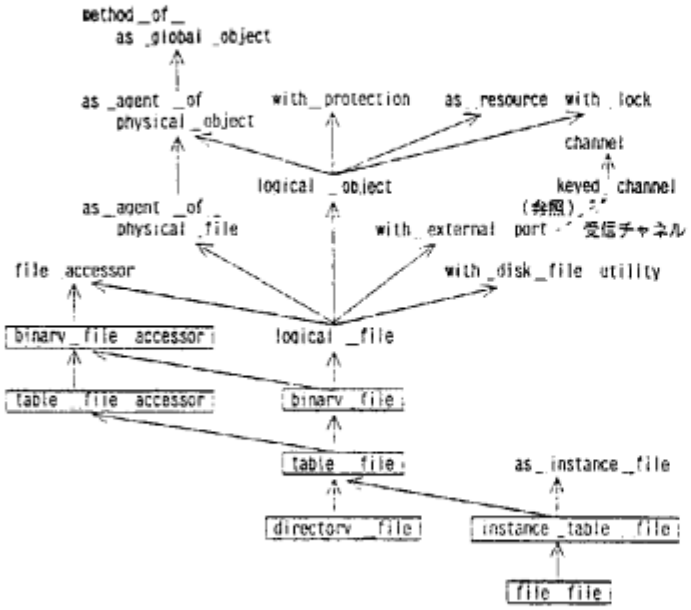
  本システムではボリューム・オブジェクトは従属オブジェクトとしており、物理ファイルと論理ファイルからアクセスされる。論理ファイルからは物理ファイル同様に独立オブジェクトとしての顔をみせており、物理ファイルには従属オブジェクトとしての側面をみせている。すなわち、物理ファイルからのアクセスは論理ファイルからのそれより優先しなければならない。独立オブジェクトだけの世界でオブジェクト間の交信関係がネットワーク構造を形成した場合、このような“親からメッセージが子孫に伝わるまで子孫は他人からのメッセージを受付けない”といったオブジェクトの相互関係に基づく優先制御を行なうことが難題の一つと考える。

オブジェクト指向モデルはある面で実世界に近く、実世界のなげかける並列・分散に関する潜在的な問題を浮彫りにする点で大変興味深いものである。

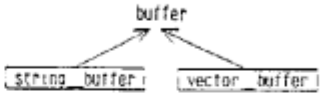
参考文献

[小松84] 小松光雅，真野忠志，小長谷明彦，服部隆　“SIMPOSのファイル・サブシステム”
情報処理学会第２９回全国大会，1984

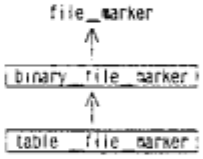
付録1　クラス構成図



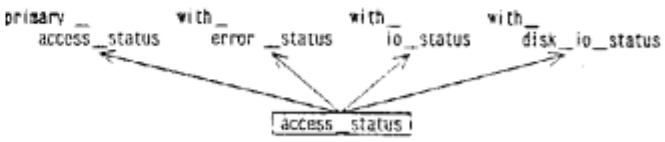
(a) 論理ファイルおよびファイル・アクセサのクラス構成図



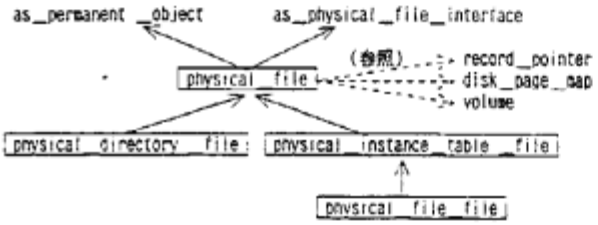
(b) バッファのクラス構成図



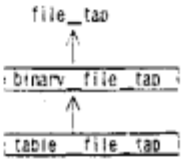
(c) ファイル・マーカのクラス構成図



(d) アクセス・ステータスのクラス構成図



(g) 物理ファイルのクラス構成図

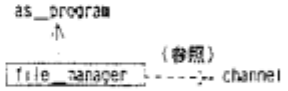


(e) ファイル・タップのクラス構成図

(h) ボリュームのクラス構成図

(f) 標準入出力ファイルのクラス構成図

(i) ファイル・メッセージのクラス構成図



(j) ファイル・マネジャのクラス構成図

付録２　プログラム例

例 A2-1 バイナリ・ファイルへの文字列追加（直接データ入出力）

```
:open(#binary_file, File, Pathname),          ファイルのオープン
:add _data(File, "a string"),                 文字列追加
:close(File)                                  ファイルのクローズ
```

例 A2-2 テーブル・ファイルからの順次読込み（順次アクセス）

```
:open(#table _file, File, Patname),
:move(File, n),                               第ｎレコード位置に移動
:read_next(File),                             第ｎレコードの読込み
:get _data(File, Record1),                    内蔵バッファからデータ取出し
        ・
        ・
:read_next(File),                             第（ｎ＋１）レコードの読込み
:get _data(File, Record2),                    内蔵バッファからデータ取出し
```

例 A2-3 ダブル・バッファリングによるバイナリ・ファイルへ文字列追加（非同期メッセージ通信）

```
:open(#binary_file, File, Pathname),
:put _data(File, string#"Taro"),              Fileの内蔵バッファへ文字列転送
:submit_add(File, ID1),                       追加要求１
:accessor(File, Accessor),                    FileにAccessorを取付ける
:put _data(Accessor, string#"Jiro"),          Accessorの内蔵バッファへ文字列転送
:submit_add(Accessor, ID2),                   追加要求２
:wait(File, ID1),                             委託１の結果を受取る
:put _data(File, string#"Saburo"),            Fileの内蔵バッファへ文字列転送
:submit_add(File, ID3),                       追加要求３
:wait(Accessor, ID2),                         委託２の結果を受取る
:wait(File, ID3),                             委託３の結果を受取る
```