プロトタイプベースオブジェクトファイルシステム

金子 勇[†] 畠山正行[††]

本論文ではまず，オブジェクト指向の大規模シミュレーションのためにはオブジェクト指向実行環境が必要であるとの考えから出発し，そのためのオブジェクト指向ファイルシステムの開発の必要性を指摘する．そこで，現用の階層構造を持つファイルシステムをオブジェクト指向化する方針とその具体的な方法論として，UNIX のファイルシステムに対してプロトタイプベースオブジェクトの概念を用いることを提案した．そしてこれを実現するために，ファイルシステムのディレクトリをオブジェクトとするカプセル化，ディレクトリ単位の情報隠蔽を実現するアクセス制限機構，ディレクトリの合成をともなう委譲リンク，セルフディレクトリの概念を新たに提案し，実際に UNIX ファイルシステムとの上位互換性を持つ PBO-FS（Prototype-Based Object File System）の設計と実装を行った．また，この PBO-FS をオブジェクト指向による流れの数値風胴シミュレーションに応用し，実用的なシミュレーション環境が実現可能であることを確認した．この PBO-FS 自身とその応用例を他の研究と比較した結果，プロトタイプベースオブジェクトの概念をファイルシステム上で実現するという本研究の手法が新規なものであるとの結論を導き出した．

Prototype-Based Object File System

ISAMU KANEKO[†] and MASAYUKI HATAKEYAMA[††]

The starting point of the present paper is based on the idea that the execution environments should be developed on the object-oriented file system for the large scale simulations of the physical phenomena. Then, we have developed an extended hierarchical file system that will realize the concept of the prototype-based object. For this purpose, the access restriction mechanism to realize the encapsulation and the information hiding of the object, the delegation link to share the common parts among the objects, and the concept of the self-directory have been introduced and implemented on the UNIX file system. We call this extended (upper-compatible) UNIX file system the Prototype-Based Object File System (PBO-FS). This file system has been applied to some problems such like the object-oriented numerical wind tunnel flow system. The results are satisfactory and valid as the object-oriented execution environment. This PBO-FS and/or its application examples have been compared with other studies and that no other similar concepts or systems like the PBO-FS have been found out. Then we can consider that this PBO-FS is a new object-oriented file system with the concept of the prototype-based object.

1. はじめに

我々の研究グループではここ数年来，オブジェクト指向モデリングパラダイムに基づいた流れのシミュレーションのモデル化とその実現の研究を行ってきた．これらの研究の基本方針は，対象世界のモデル化，システム分析・設計から，実装・実行に至るまでのすべての過程にオブジェクト指向の方法論を一貫して用いるということであった．そしてこの方針に基づき，実際にいくつかの実例を実現・評価してきた[1)~4)]．

しかし，我々が対象としている複雑かつ大規模な計算が必要とされるシミュレーションでは，現行の一般的なオブジェクト指向の方法論やそのための種々のシステムが必ずしも効率的に働くとは限らないことが分かってきた．たとえば，インタプリタ言語により構成されている Smalltalk 環境[5)]などでは実行速度は遅く，連立方程式解法等のライブラリも揃っておらず，実用性は低い．逆にコンパイラ型のオブジェクト指向言語を用いたシミュレーションシステムは現状で多少普及しつつあり，クラスライブラリ等の整備によりシステマティックに作成されつつあるが，多くのプログラムやデータファイルからなるシミュレーションシステム

† 茨城大学大学院理工学研究科情報・システム科学専攻
Department of Information and System Science, Graduate School of Science and Engineering, Ibaraki University
†† 茨城大学工学部情報工学科
Department of Computer and Information Sciences, Faculty of Engineering, Ibaraki University

全体の管理などはオブジェクトという単位等とは対応しない旧態依然たる管理が行われているものが多い[6].

このような大規模な流れのシミュレーションでは，多数のプログラムやデータ群がそれぞれ関連を持って生成される．オブジェクト指向で一貫してシミュレーションを行う際には，これらプログラムやデータ群の管理もオブジェクト単位で行われることが望ましい．これを実現する理想的な環境は，これらシミュレーションに用いる各データやプログラム群を，オブジェクト単位で管理，運営することが可能なシステムである．現行システムでこれに適したシステムとしては，オブジェクト指向データベース管理システム[7]（OODBMS）をあげることができる．

我々もOODBMSを用いてシミュレーションシステムを構築してきたが[1)~4)]，DB内にプログラムを格納し，これを直接実行できないことが問題になった．すなわち，処理速度面の要請の高いシミュレーションではプログラムをC++等のコンパイラ型言語で記述する必要がある．しかし，コンパイラ型オブジェクト指向言語を基盤としているOODBMSでは，クラスメソッドの実行コードはDB内部ではなくアプリケーションプログラム側にリンクする形で扱われている[8]．なおコンパイラ型ではなくインタプリタ型の言語を用いるOODBMSを用いることで，クラスメソッドをDB内に格納し直接これを実行することも可能であるが，本研究の対象としているのは流れのシミュレーションのように大規模計算をともなうものであるため，コンパイラ型言語を用いることを前提としている．

そもそも実行プログラム群を実際に管理しているのはファイルシステムである．また試行錯誤的な運用が行われるシミュレーションシステムでは，プログラム変更の容易さから，DBMSよりもファイルシステムの方が実行コードおよびデータの管理の両面においても優れている場合が多い．そのため現状でシミュレーション時のプログラム・データ管理の基盤となっているファイルシステムそれ自身において，プログラムをオブジェクトのメソッドの形で扱えることが最も良い解決法であると思われた．

しかし現状でオブジェクト指向ファイルシステムと呼ばれているものは，以下の2つの方式のどちらかである．

- ファイルシステムのインタフェース部が抽象データ型となっているため，オブジェクト指向ファイルシステムとなっているもの[9)~11)]
- 永続オブジェクト管理機構上にファイルシステムを構築したことにより，オブジェクト指向ファイルシステムとなっているもの[12)]

しかしこれらは我々が必要としたオブジェクト指向ファイルシステムとは異なる．そこで我々は，これらとは異なる方針のオブジェクト指向ファイルシステムを提案する．それは，オブジェクトしてディレクトリを仮定し，ディレクトリ内のプログラムをオブジェクトのメソッドとするファイルシステムである．またオブジェクト指向の中でもプロトタイプベースオブジェクトモデル[13),14)]が，UNIX File System（以下，**U-FS**）と構造的な類似点を持つことに注目し，このモデルを用いてファイルシステムをオブジェクト指向化する．

これらの概念は現行のUNIXの上にライブラリを構築することでも実現可能であることが分かっているが[15)]，この方式では以下の章で述べるオブジェクトの情報隠蔽の点で不完全という問題点がある．そのため本研究ではU-FSをオブジェクト指向化するためにどのような機構が具体的に要求されるのかを考察・提案するとともに，このオブジェクト指向ファイルシステムの実装とその検証，有効性について検討する．

以下において，まず2章でオブジェクト指向ファイルシステムの持つべき条件について考察し，それから導かれるファイルシステムに追加すべき新たな要素について述べる．3章で実際にこの考えに基づくファイルシステムの設計と実装について述べる．そして4章でこのファイルシステムの特徴を議論・考察するとともにシミュレーションその他への応用例を示し，5章以降でそれらに対する他の研究との比較，考察や結論，今後の展望等を述べる．

2. オブジェクト指向ファイルシステム

本章ではオブジェクト指向ファイルシステムが持つべき条件と，それを実現する際の実用性からくる制約について述べ，これら各条件の充足方法を提案する．

2.1 オブジェクト指向ファイルシステムの成立条件

一般的にオブジェクト指向であるためには，カプセル化，情報隠蔽，メッセージパッシングによる相互作用，クラス，継承の概念が含まれていなければならない．そのため本研究では，オブジェクト指向ファイルシステムの成立条件を次のように定義する．

(1) データとメソッドをひとまとめにしたオブジェクトを表す構造体がファイルシステム上に存在する（カプセル化）．
(2) そのオブジェクト単位のアクセス制限が実現されている（情報隠蔽）．
(3) プログラムの実行はオブジェクトに対するメッ

セージの送信という形で行う（メッセージパッシング）.

(4) オブジェクトは雛形とするオブジェクトから生成される（クラス）.

(5) クラス間の要素包含関係を扱うことができる（継承）.

また前章のシミュレーション環境として用いる制約から，本研究では以下の3つの条件も満たされていることが望ましい.

(6) 従来ファイルシステムであるU-FSとの上位互換性を保っている.

(7) 既存のプログラムやデータファイル等をそのまま再利用できる.

(8) プログラムの記述が，任意の言語で行える.

これらの各条件を，どのようにして満たすかについて，以下それぞれ列挙する．また，これらの条件充足の検証は3.5節で行う.

2.2 カプセル化とセルフディレクトリ

本研究ではすでに1章で述べたように，オブジェクトとしてディレクトリをあてることで(1)のカプセル化の条件を解決する．これによりデータファイルがオブジェクトの属性となり，プログラムファイルがオブジェクトのメソッドとなる．さらに次節のディレクトリ内の情報隠蔽を行うことで，複数のファイルから構成されるディレクトリの内部構造が隠され，オブジェクト単位での格納・管理と運用が可能になる.

このように本論文で主張するオブジェクト指向ファイルシステムでは，オブジェクト管理システムといえるのはディレクトリから上の階層である．ディレクトリより下の階層では通常のファイル単位での管理システムとなる.

次にこのカプセル化の方針により，プログラム駆動時にプロセスがどのオブジェクトのメソッドとして働いているかの情報がプロセス内に必要となる．この「プロセスが実行している実行可能ファイルを含むディレクトリ」をオブジェクト指向言語におけるSelfの概念に基づき，セルフディレクトリと名付ける．またこれを基底とする新たなパス名記法を図**1**のように提案する.

なお，以下の図ではファイル名の後ろに記号を付加することでその種類を明示した．ディレクトリに“/”，実行可能ファイルに“*”，シンボリックリンクに“@”，後で述べる委譲リンクに“::”を付加している．また通常ファイルは記号を省略することで表すこととした.

2.3 情報隠蔽

従来のU-FSでは，ユーザ権限による保護を考えなければファイルシステム内の任意のファイルに任意のプログラムからアクセスできる．しかし，ディレクトリをオブジェクトと考えると，このままではオブジェクト指向ファイルシステムが持つべき条件(2)のオブジェクト要素の情報隠蔽が成立していない．そのためファイルシステムに新たなアクセス制限を付加することでディレクトリ内部要素の情報隠蔽を実現する.

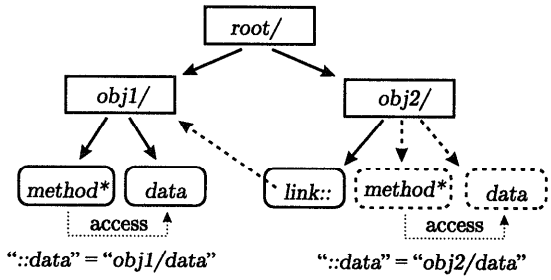

図1 セルフディレクトリ
Fig. 1 Self directory.

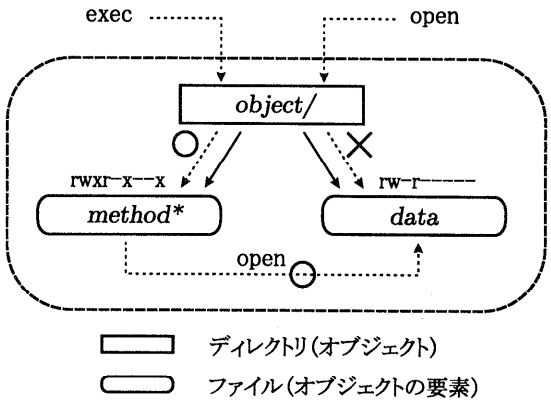

図2 ディレクトリのカプセル化と情報隠蔽
Fig. 2 Encapsulation and information hiding of directory.

具体的には，図**2**に示すようにデータファイルへのアクセスを同一ディレクトリに存在する実行可能プログラムからのみ可能とする．この新たなファイルアクセス制限機構は図2のrwxのような形で新規付加することで行う．このアクセス権限情報（`rwx r-x --x`）の意味はそれぞれ3ビットごとに，セルフディレクトリ内からのアクセス，2.6節で述べる委譲リンクを用いたアクセス，そしてその他のオブジェクトからのアクセス情報である．この3ビットはそれぞれ読み出し，書き込み，実行に関する可否である．このような権限情報を用いて各ファイルの情報隠蔽を行う.

図2のファイル“data”の場合，同一ディレクトリ内のプログラムからのみアクセスが可能である．他の

ディレクトリ内のプログラムから生成されたプロセスがファイル “data” へアクセスする際には，対象ディレクトリ内にあるファイル “data” を扱うプログラム “method” を実行することで行う．これによりファイル内容が情報隠蔽される．

2.4 メッセージパッシング

本研究では，オブジェクトへのメッセージパッシング（相互作用）をディレクトリ内のプログラム起動手順として定義する．これにより，条件(6)〜(8)を満たすことが可能となる．

2.5 オブジェクトの生成

ディレクトリをオブジェクトと見なす本研究の考えからは，クラスに相当する概念をファイルシステム上に導き出すことはできない．そこでプロトタイプベースオブジェクトモデルをファイルシステムに導入し，これを解決する．

プロトタイプベースオブジェクトモデルでは，インスタンスとクラスを別のものとはせず，ともにオブジェクトとして扱う．プロトタイプベースオブジェクトモデルにおいては，新しいオブジェクトの生成は他のプロトタイプ（典型例）とするオブジェクトをコピーすることで行われる．そのためコピーベースのオブジェクト指向と呼ばれることもある．

このプロトタイプベースオブジェクトモデルをファイルシステムに導入すると，オブジェクトの生成は，他のディレクトリのコピーとなる．しかし，オブジェクトの生成を実際にディレクトリのコピーで行うことは効率上問題があるため，本研究では次節で述べる委譲の概念をオブジェクトの生成に用いる．それは，新たにオブジェクトを生成する場合，ディレクトリを1つ作成しその内部にプロトタイプディレクトリに対する委譲リンクを作成することである．これにより，オブジェクト要素のコピーを省略する．

2.6 継承と委譲

2.1節，2.2節の方針から，各ディレクトリ内にはそのディレクトリ内のファイルを扱うすべてのプログラムが集約される．ここで多くのオブジェクト間で共有可能あるいは他のオブジェクトで利用可能なメソッド（実行可能ファイル）や属性（通常ファイル）が多数存在する．オブジェクト指向では，このようなオブジェクト要素の共有に関して継承の概念を用いる．この継承の概念はオブジェクト指向にとって重要なものであるが，そのままではファイルシステムに導入することができない．しかしクラスベースではなく，プロトタイプベースオブジェクトモデルを採用することで，継承と同等の機能を委譲という形でファイルシステム上に導入できる．

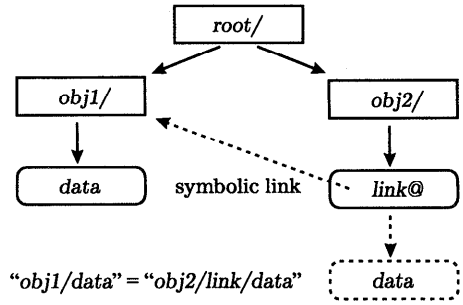

図3 シンボリックリンク
Fig. 3 Symbolic link.

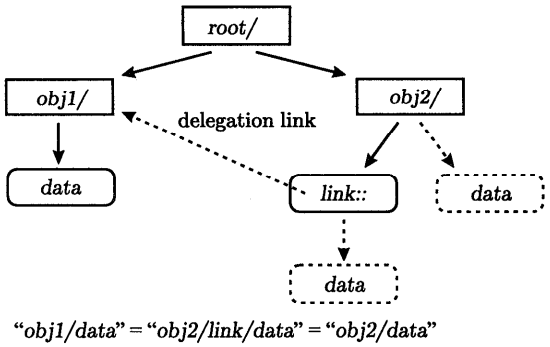

図4 委譲リンク
Fig. 4 Delegation link.

委譲とは，オブジェクトにメッセージ送信があったときに他のオブジェクトにそのメッセージを委託することで，継承と同様の働きを行うプロトタイプベースオブジェクトにおける概念である[13]．継承がクラス間で働いたのに対して，委譲はインスタンス間で働く．本研究ではこの委譲の概念を，シンボリックリンクの拡張である委譲リンクという形でファイルシステムに対して導入する．

委譲リンクの具体的な例を記述する．まず**図3**はU-FSのシンボリックリンクの場合である．“obj1/data” というファイルが存在する状態で他のディレクトリ “obj2” 内部に “obj1” に対するシンボリックリンクを “link” というファイル名で作成した場合，“obj2/link/data” で示されるファイルは “obj1/data” で示されるファイルに等しい．次に**図4**のようにこの “obj2/link” が委譲リンクの場合，シンボリックリンクと同様 “obj2/link/data” の名前で “obj1/data” にアクセスできるが，他に “obj2/data” というファイル名でも “obj1/data” にアクセスできる．これがシンボリックリンクと委譲リンクの違いである．すなわち委譲リンクを用いた場合 “obj1” 内のファイルはすべて “obj2” 内にも存在しているように扱うことがで

```
struct dinode {
        u_short         di_mode;        /*  0: IFMT and permissions. */
        short           di_nlink;       /*  2: File link count. */
        ino_t           inumber;        /*  4: inode number. */
        u_quad_t        di_size;        /*  8: File byte count. */
        struct timespec di_atime;       /* 16: Last access time. */
        struct timespec di_mtime;       /* 24: Last modified time. */
        struct timespec di_ctime;       /* 32: Last inode change time. */
        daddr_t         di_db[NDADDR];  /* 40: Direct disk blocks. */
        daddr_t         di_ib[NIADDR];  /* 88: Indirect disk blocks. */
        u_long          di_flags;       /* 100: Status flags (chflags). */
        long            di_blocks;      /* 104: Blocks actually held. */
        long            di_gen;         /* 108: Generation number. */
        u_long          di_uid;         /* 112: File owner. */
        u_long          di_gid;         /* 116: File group. */
        u_short         di_permit;      /* 120: File permissions. */
        short           di_sspare;      /* 122: Reserved; currently unused */
        long            di_spare;       /* 124: Reserved; currently unused */
};
```

図5 PBO-FS のiノード構造体
Fig. 5 i-node structured data of PBO-FS.

きる.

これにより，あるディレクトリ内のファイルにアクセスがあり，そのファイルがなかった場合に他のディレクトリに委託するという委譲の概念をファイルシステム上で表現することができる.

本章の結論としてU-FSに対して新たに提案する機構は，セルフディレクトリとそれに基づくパス名記法，ディレクトリ内部を情報隠蔽するための新規なファイルアクセス制限機構，ファイルを各ディレクトリ間で共有可能とする委譲リンク，の3点である.

3. PBO-FS の設計と実装

本研究では2章の考察を基に，新たなファイルシステムの設計と実装を行った．このファイルシステムを本研究では**PBO-FS**（Prototype-Based Object File System）と呼ぶこととした．そしてこのPBO-FSの実装を4.4BSD Lite（FreeBSD）のUFSを基盤として行った.

3.1 ディレクトリを情報隠蔽するアクセス権限

2.4節の考察を基にファイルアクセス権限の拡張を行った．ファイルアクセス制限としてはアクセス権限リスト[16]やケーパビリティ[16]によるものが考えられる．ここで，従来のU-FSでは，プロセスとファイル間のユーザID（UID）一致，グループID（GID）一致，UIDもGIDも不一致，それぞれに対して読み・書き・実行可能を表す9ビットの情報を各ファイルごとに付加することでアクセス制限を行っている.

PBO-FSではこのU-FSの概念と，C++言語[17]などで見られる，自身のオブジェクト内，継承先のオブジェクト内，その他のオブジェクト内からのアクセスを分けて制限することを組み合わせ，情報隠蔽実現のためのアクセス権限（情報隠蔽アクセス権限）を新たに設けることとした．すなわち，同一ディレクトリ内の実行可能ファイルを実行しているプロセスからのアクセス（セルフアクセス権限），委譲リンク検索によるアクセス（委譲アクセス権限），その他からのアクセス（その他のアクセス権限），それぞれの読み・書き・実行可能を表す9ビットのアクセス権限情報を新たにファイルに付加する．実装の主要な点は以下のとおりである.

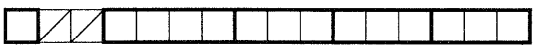

図6 iノード構造体のdi_permitの内容
Fig. 6 Array of di_permit of i-node structured data.

(1) U-FSのiノード，VFS[18]のvノードへのアクセス権限情報di_permitの追加（**図5**，**図6**）.
(2) カプセル化アクセス権限を変更するシステムコールchper（change permission）の追加（**表1**参照）と，これを参照するシステムコール（stat, fstat, lstat）の変更.
(3) ファイル作成時の情報隠蔽アクセス権限マスクを設定するシステムコールpmask（permission mask，表1）を追加.
(4) open, execveなどの各システムコールにおいて情報隠蔽アクセス権限チェックを追加.
(5) 情報隠蔽アクセス権限を変更するコマンドchperの追加と，これを参照するコマンドlsなど

表1 PBO-FSのシステムコールとライブラリ関数
Table 1 System calls and library functions of PBO-FS

関数定義	機能
int chper (char *path, int permit)	情報隠蔽アクセス権限を変更する.
int pmask (int mask)	ファイル生成時の情報隠蔽アクセス権限のマスク変更.
int dellink (char *spath, char *dpath)	委譲リンクを作成する.
char* getself (char *buf, int size)	セルフディレクトリを取得する.
int chself (char *path)	セルフディレクトリを変更する. Root 権限でのみ使用可.

の変更.

なお, このアクセス権限を変更する権限に関しては, 従来のユーザ権限に従うものとした. アクセス権限の変更を, そのディレクトリ内のプログラムに制約するという設計も考えられるが, 従来のU-FSにおける各ユーザの書き込み権限がそのユーザから自由に変更できるのと同様に, 利便性を重視してこのようになっている.

また, 2章では通常ファイルに関するアクセス権限についてのみ述べたが, このアクセス制限は実行ファイルに対しても同様に働く.

3.2 委譲リンク

3.2.1 委譲リンクの設計

2.6節で述べたように, PBO-FSにおける委譲機構はU-FSのシンボリックリンクを拡張することで実現されている.

U-FSのシンボリックリンクと委譲リンクとの違いは, ディレクトリ内にファイルが見つからずそのディレクトリ内に委譲リンクが存在している場合, 委譲リンク先のディレクトリの検索も行うことである.

委譲リンクは以下に述べるディレクトリの合成が起こる点でシンボリックリンクとは異なる概念である. まず**図7**のように委譲リンク先のディレクトリに委譲リンクが存在する場合は, その委譲リンク先のディレクトリに関しても再帰的に検索が行われる. そのためobj3/に "data1" と "data2" が合成されたように見える. さらに**図8**のように図7のデータファイル名を2つともdataとすると, 委譲リンク元と委譲リンク先に同一名のファイルが存在することになるが, 委譲リンク元のファイル "obj2/data" が優先され委譲リンク先のファイル "obj1/data" はオーバライドされ見えなくなる. これはオブジェクト指向における多段継承に対応している. ここでオーバライドされてしまったファイルに対しては, 委譲リンクがシンボリックリンクとしても働くので, これをたどることでアクセスが可能である.

次に, **図9**のように1つのディレクトリ内に複数の委譲リンクが存在している場合にもディレクトリの

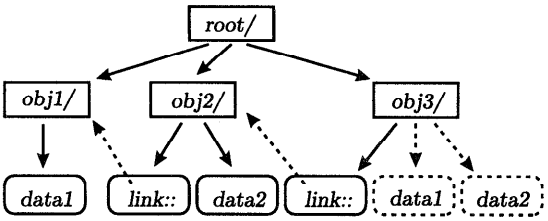

図7 多段の委譲リンクによるディレクトリの合成
Fig. 7 Dirctory composition using multi-stage delegation links.

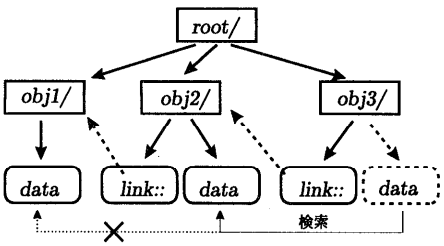

図8 委譲リンクによるファイルのオーバライド
Fig. 8 File override using delegation links.

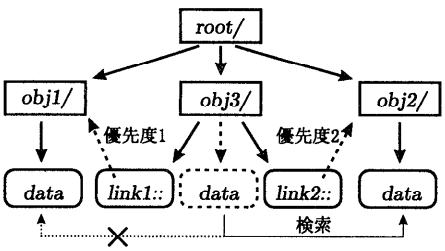

図9 多重の委譲リンクによるディレクトリの合成とオーバライド
Fig. 9 Directory composition and override using multiple delegation links.

合成とオーバライドが起きる. これはオブジェクト指向における多重継承に相当している. この場合委譲リンクごとに持つ検索の優先値の大きい順に検索することで, 名前空間の衝突を解決する. また, すでに述べたように委譲リンク先の委譲リンクの検索も同時に行われるが, これは深さ優先検索 (委譲リンク先のディレクトリ内を検索してから優先度の小さい委譲リンク

先を検索）で行うこととする．

またPBO-FSでは，openシステムコール時の委譲リンク検索はcreateオプションが指定されていない場合にのみ行う．fopen(“path”, “w”)などの，ファイルを新規作成する場合は委譲リンクの検索を行わない．

例として図4の状態において，“obj2” 内に新たに“data” という名でファイルを作成した場合は“obj2”の中に実際にファイル“data” が生成される．そのため，ファイルの内容をすべてメモリ上に読み込んだ後，同一名で新たにファイルを作成し書き込みを行うと，結果的に書き込み時コピー動作となる．

PBO-FSの他の設計としては，委譲リンク作成直後は“obj1/data” の方を読み，“obj2/data” への書き込みが起こった時点で自動的に“obj1/data” を“obj2/data” にコピーし，以降“obj2/data” にアクセスするという，一般的な書き込み時コピーを行うことも考えられる．しかし，U-FSとの互換性と，実装の容易さから，PBO-FSではこのような設計とした．そのためPBO-FSでは，ファイルへの追加（fopen(“path”, “a”)）などの場合には，ファイル内容のコピーをユーザプログラム側で行う必要がある．

また，委譲リンク生成の権限に関してであるが，2.3節で述べたアクセス権限により保護が行われる．通常ファイルの生成もこれと同様である．

3.2.2 委譲リンクの実装

委譲リンクはdellinkシステムコール（表1）により作成される．この際，委譲リンクであることを識別するための情報と委譲優先度情報がディレクトリのエントリとiノードに追加される（図6のd_linkとd_no）．PBO-FSではU-FSとの上位互換性確保の理由により，委譲優先度d_noを4ビットに制限している（図6参照）．また委譲優先度0の委譲リンクは用いない．これにより委譲優先度は1から15であるが，初めに検索が有効となるのは数字が大きい方からとする．委譲機構の実装主要点は以下のとおりである．

(1) 委譲リンクを張るシステムコール dellink（delegation link）の追加とこれを利用できるようにコマンドlnを変更．

(2) 委譲リンクを検索するために，カーネル内部のパス名解釈関数namei等を変更．

PBO-FSでは，dellinkシステムコールにより委譲リンクが新たに作成される場合，委譲リンクが作成されるディレクトリ内で使われていない最も小さい委譲優先度の値が使われる．これにより委譲リンクが存在しないディレクトリに新たに委譲リンクを作成すると，作成順に委譲優先度が1，2，...となる．先に述べたように委譲リンクは優先度が大きいほど先に検索されるので，後に作成された委譲リンクが以前に作成された委譲リンクより先に検索に使用される．

また従来のシステムコールは，委譲リンクによる検索が行われるシステムコール群と行われないシステムコール群に区別される．これはシンボリックリンクが有効なシステムコールと無効なシステムコールが存在していることと同様である．また前節ですでに述べたようにopenシステムコールに関しては，createオプションが指定されていない場合にのみ，委譲リンクの検索を行うものとしている．

次に委譲リンクがループ状に作成されている場合の処理であるが，委譲リンクは特殊なシンボリックリンクとして実装されているため，シンボリックリンクと同様の対策法で行われている．

まずU-FSのシンボリックリンクでは，リンク先がリンクだった場合にこれをある回数以上検索すると，“Too many levels of symbolic links”エラーが発生する．ここでシンボリックリンクではリンク先を順に追っていかないと，このエラーが発生しないが，委譲リンクではファイルが見つからない場合，自動的に次のリンク先を検索する．そのため，委譲リンクのループが存在しているパス内で存在しないファイル名を検索するとこのエラーが発生する．

3.3 セルフディレクトリ

U-FSではファイル検索の開始点として，ファイルパス名が“/” で始まる場合にはプロセスごとに持つルートディレクトリを用い，その他の場合はプロセスごとに持つカレントディレクトリを用いる．これに加えてPBO-FSでは，名前が“::” で始まった場合にプロセスごとに持つセルフディレクトリから検索を行う．

ここで，セルフディレクトリは2.2節で述べたように，プロセスが実行しているプログラムファイルが存在するディレクトリである．しかし，例外として，委譲リンク検索によるファイル実行で生成されたプロセスでは，実際に実行可能ファイルが存在する委譲リンク先のディレクトリではなく，検索に用いたファイル名に相当する委譲リンク元のディレクトリをセルフディレクトリとする．それは委譲リンク先のファイルが実際には委譲リンク元にあるものとして扱われるからである．この例を図1に示す．“obj1/method” を実行しているプロセスにおいて“::data” で表されるファイルは“obj1/data” である．しかし，“obj2/method” を実行しているプロセスから見ると“::data” は”obj2/data” となる．図1では“obj2内” に“obj1” への委譲リンクが存在しているため，プログラム“method” が実際に

は “obj2” 内に存在しないにもかかわらず実行可能である．この場合でも “::data” はファイル “obj2/data” を表す．ここでもしファイル “obj2/data” もなかった場合，再び委譲リンクの働きによりファイル “obj1/data” がアクセスの対象となる．

このセルフディレクトリの主要な実装点を以下に示す．

(1) Self directory をプロセスエントリに追加．
(2) プログラム実行の際に Self directory が変更されるようにシステムコール execve を変更．
(3) Self directory が “::” の名で参照できるようにカーネル内部のファイル名解釈関数 namei 等を変更．
(4) Self directory 名を参照するライブラリ getself (get self directory) と，変更するシステムコール chself (change self directory) を追加 (表1)．

セルフディレクトリの実装は U-FS のカレントディレクトリの実装と同様の手法で行われている．カレントディレクトリとの違いは，セルフディレクトリの更新が，他のプログラムを実行したとき (execve システムコール実行時) にのみ行われることである．

3.4 オブジェクト指向ファイルシステムの成立

本節では，2.1 節の条件 (1)〜(8) が PBO-FS で充たされているかを検証する．

まず従来の U-FS におけるディレクトリをオブジェクトとし，セルフディレクトリの概念を新たに導入したことで，(1) の条件は充たされている．また，ディレクトリ内部を情報隠蔽するためのアクセス権限を新たに導入したことで，(2) の条件も充たされている．(3) の条件は，ディレクトリ内のプログラム起動をオブジェクトへのメッセージパッシングと見なすことで充たされている．また (4) のオブジェクトの生成，ならびに (5) の継承に相当する機能は，委譲リンクを導入することで実現されている．(6)〜(8) の条件はオブジェクトの要素を従来の U-FS のファイルシステム要素に当てるという (1) と (3) の充足方法により充たされている．PBO-FS には表 1 のシステムコールやライブラリ群が追加されているが，情報隠蔽のためのアクセス権限をすべて公開と設定し委譲リンクを用いなければ PBO-FS は U-FS として用いることができる．以上から 2.1 節で示した条件はすべて充たされており，PBO-FS はオブジェクト指向ファイルシステムとしての条件を充たしているといえる．

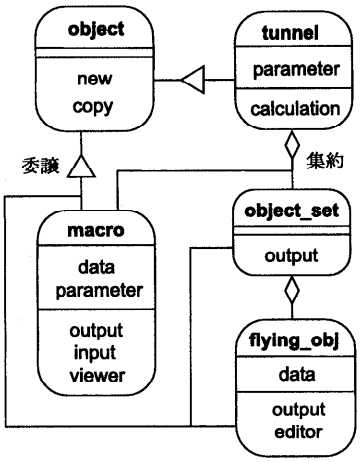

図 10 数値風洞オブジェクトモデル図 (OMT 類似記法)
Fig. 10 Object model diagram of object-oriented numerical wind tunnel using OMT like notation.

4. PBO-FS の応用とその特徴

4.1 シミュレーション環境としての応用

本節では PBO-FS の有用性を示すために，シミュレーション実行環境として用いた具体例を示す．我々がシミュレーション対象としたのは，数値風洞である[4]．このようなシミュレーションシステムでは，計算のための各種パラメータ，初期データ，形状データ，また，それらを扱うための各種エディタ，可視化ツール，前処理・後処理用のデータコンバータ，そして計算を行うための計算プログラムなどがファイルの形で構成されている．PBO-FS 上では，これら各ファイルをオブジェクトのデータやメソッドと見なし，各々の関連を考慮したうえでファイルシステム上に配置する．

数値風洞シミュレーションシステムの簡略化したモデル図を**図 10** に示す．なおこの図は OMT[19] に似た記法を用いているが，各オブジェクトはすべてクラスではなくインスタンスであり，各オブジェクトごとにデータやメソッドを持つ．また OMT の継承表現は委譲を意味している．

図 10 のモデルを PBO-FS 上で実現すると**図 11** のような構造となる．tunnel/が数値風洞本体を表すディレクトリであり，風洞のパラメータを表すファイル “parameter” とシミュレーションの計算プログラム “calculation” を要素として持っている．また macro/は風洞内の流れを表す物理世界の分布のデータを格納・保持するオブジェクトであり，内部にはそのデータファイルとそれを扱うためのプログラムを要素として持っている．object_set/は風洞内に置かれる物体の

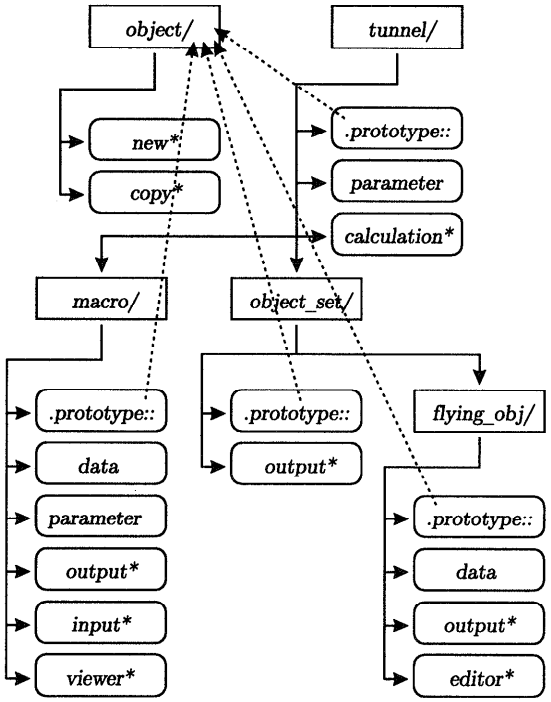

図 11 数値風洞システムディレクトリ構成図
Fig. 11 Object-oriented numerical wind tunnel system on PBO-FS.

集約であり，flying_obj/は集約されているオブジェクトの 1 つである．これらは各々データやパラメータを持ち，それらを外部に出力するプログラムを有している．

またディレクトリ内部を情報隠蔽するために，**図 12** のように PBO-FS におけるアクセス権限を，通常ファイルはセルフディレクトリ内からのアクセスのみ可能とし，実行可能ファイルはディレクトリ外部からでもアクセス可能とする．そしてファイルアクセスはセルフディレクトリ相対のパス名を用いて行う．これにより，データファイル（通常ファイル）へのアクセスが同一ディレクトリ内の実行可能ファイルに制限される．

また 2.5 節で述べたように，オブジェクトの生成は委譲リンクを用いることで行われる．例としてこの数値風洞で実際に計算を行う際には，**図 13** のように空ディレクトリを 1 つ作成しその内部に tunnel/に対する委譲リンクを作成することで行う．図 13 の例では a_tunnel/の生成がこれである．この操作は頻繁に行われるため，このためのプログラム “new” を初めから用意しておき，すべてのオブジェクトの委譲リンク先となる object/を用意してそこへ委譲リンクを設けることで（図 11 参照），このプログラムを共有する．

プログラム “new” により新たに作成されたディレクトリ内部には，委譲リンクのみ存在するが，委譲リンクの働きにより tunnel/と同一に扱うことが可能であり，この tunnel/内のプログラムを実行することができる．そしてこのプログラム内部でパラメータファイル “parameter” の作成を行うと，委譲リンク先の tunnel/ではなく委譲リンク元の a_tunnel/のディレクトリに新たなファイルが生成される（図 13 参照）．このファイルの読み込みは委譲リンク先から行われるので，委譲リンク先のファイルが自動的にテンプレートとして働く．

また，複数の委譲リンクを用いることで，複数のオブジェクトを組み合わせた新たなオブジェクトを生成することが可能である．このように PBO-FS を用いることで，一時的に計算プログラムやデータファイル

```
$ cd tunnel/macro
$ ls-alFp              // 情報隠蔽アクセス権限を表示(1列目)
drwxrwxrwx lrwxr-xr-x  ... .prototype:: ->[1] ../../object
-rwxrwxrwx -rw-r--r--  ... Data
-rwxrwxrwx -rw-r--r--  ... parameter
-rwxrwxrwx -rwxr-xr-x  ... output*
-rwxrwxrwx -rwxr-xr-x  ... input*
-rwxrwxrwx -rwxr-xr-x  ... viewer*
$ cat data                    // catからアクセス可
1000 0 0 0 1000
        .
        .
$ chper 750 data parameter    // ディレクトリ外からのアクセスを禁止
$ ls-alFp                     // 変更を確認
drwxrwxrwx lrwxr-xr-x  ... .prototype:: ->[1] ../../object
-rwxr-x--- -rw-r--r--  ... data
-rwxr-x--- -rw-r--r--  ... parameter
-rwxrwxrwx -rwxr-xr-x  ... output*
-rwxrwxrwx -rwxr-xr-x  ... input*
-rwxrwxrwx -rwxr-xr-x  ... viewer*
$ cat data                    // catからアクセス不可
cat: data: Permission denied
$ ./output                    // outputからはdataへアクセス可
1000 0 0 0 1000
        .
        .
```

図 12 ディレクトリの情報隠蔽
Fig. 12 Information hiding of directory.

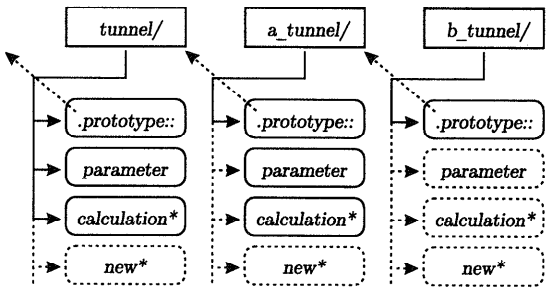

図 13 “new” による新たな “a_tunnel” の作成
Fig. 13 Generation of new object “a_tunnel” using the “new”.

を変更して試行錯誤的にその結果を探るというシミュレーションに特徴的な処理を容易に行うことが可能となっている．

4.2 オブジェクト指向シミュレーション実行環境

前節では一般的なシミュレーション環境としての典型例を述べたが，1章で述べたように，PBO-FS開発の発端はオブジェクト指向シミュレーション実行環境の必要性から生じている．そこで本節では，このオブジェクト指向シミュレーション実行環境としてのPBO-FSの応用について述べる．

まずPBO-FSは，集約や委譲を表現することができるため，シミュレーション対象世界の構造をファイルシステム上にある程度表現することができる．そのため，「複雑なシミュレーション対象世界と相似な構造」をファイルシステム上に再現するという応用が可能である．図10と図11の関係はこの簡単な例である．そしてこの相似構造を表現可能であるという側面と，PBO-FSのU-FSの上位互換という側面から，複雑なシミュレーション世界の高い記述性と構造了解の良さ，開発の容易さ，従来環境からの移行性と互換性が得られている．

また，PBO-FSはあくまでファイルシステムであるため，オブジェクトのメソッド記述言語はC++等のオブジェクト指向言語に限られない．Fortran言語等で作成したプログラムでもオブジェクトのメソッドとして利用が可能である．

これは各メソッドごとに異なった言語で記述されたオブジェクトシステム構成を許すものであり，従来の莫大なソフトウェア資産のオブジェクト指向システムへの移行を可能にする．この特性はシミュレーション分野に限らず重要な特性である．

前節および本節では，シミュレーションシステムへのPBO-FSの応用を述べた．以下の節ではより一般的なPBO-FSの応用例について述べる．

4.3 プログラミング環境のオブジェクト化

ディレクトリのオブジェクトとしての情報隠蔽を徹底すると，プログラムの各ソース群も，エディタやコンパイラなどの外部プログラムからアクセスすることができなくなる．これは，必要に応じてアクセス権限を変更することでも対処可能であるが，PBO-FSでは一般的に，外部プログラムを使うときのみ新たな委譲リンクを設定するという方法で行う．

具体的には委譲リンク先としてコンパイラやエディタなどのプログラミングツール群をグループ化しオブジェクトと設定したディレクトリを用意しておき，必要に応じて委譲リンクを作成する．この際，現状で利用しているツール群をそのまま使用することが可能である．このようにPBO-FSを用いることで，従来の各アプリケーション群をオブジェクト指向の方法論で扱うことができる．

4.4 PBO-FSのクラスベース的な特徴と機能の応用

委譲リンクはオブジェクト指向における実現値化（Instantiation）の働きをしているが，他に特化的（Specialization）な働きもする．例として，図13のように新たに“new”で生成したa_tunnel/の計算プログラムファイル“calculation”を変更するとa_tunnel/の振舞いが変更される．このa_tunnel/内で委譲リンクの働きにより“new”プログラムを実行するとa_tunnel/をプロトタイプとする新たなb_tunnel/を生成することができる．これはa_tunnel/がtunnel/のサブクラスとしての位置付けになっていることを意味している．クラスという概念を持たず，インスタンス生成とサブクラス生成を区別しないのはPBOの特徴であり，PBO-FSもこれと同様の性質を備えている．

4.5 PBO-FS上でのプログラムソースファイル管理

PBO-FS単体で，ファイルを単位とした簡単なバージョン管理が可能である．オブジェクトファイル（～.oファイル）やソースファイル，コンパイル指示ファイル（Makefile）が含まれているディレクトリに対する委譲リンクを持つディレクトリ内では，委譲リンク先のソースファイルを委譲リンク元でも参照することが可能である．ここで委譲リンク元でエディタによりプログラムソースファイルを読み込むと，委譲リンク先のファイルを読み込んでエディタが起動し，その後にファイル保存を行うことで委譲リンク元にファイルが生成される．この状態でタイムスタンプを参考にコンパイルを行うプログラムmakeを実行した場合，自動的に委譲リンク元と委譲リンク先を参照し，更新が行われたソースファイルのみ再コンパイルされる．これにより生成されたアプリケーションはそのまま実行することが可能であり，委譲リンク先のディレクトリでは何の変更も行われていない．これによりバージョン管理を行うことが可能となっている．以上の説明は単段のバージョン管理であるが，3.2節のように多段や多重の委譲を行うことで，より複雑なバージョン管理を行うことが可能である．

5. 議論と考察

5.1 PBO-FSの評価

まず，PBO-FSの設計と実装について評価を行う．

4.4BSDを基盤としたPBO-FSの実装はすでに終了し，実際に稼働している．ここで，このPBO-FSの設計・実装は，2章から導かれるシステム構成例の一例である．そのため3章で述べたようにアクセス権限機構や，委譲機構などに関して他の設計・実装方法が考えられる．しかし本PBO-FSの実装評価に関しては，実用性の観点から一応の水準に達していると考える．

次に，本PBO-FSが当初の目的であるオブジェクト指向シミュレーションの基盤環境として十分に機能しているのかの評価を行う．

まず4章の応用例とその特徴に関する記述から，PBO-FSではシミュレーション時におけるオブジェクトの構造とその振舞いをファイルシステム上にそのまま実装できていることが分かる．また，ディレクトリ内のプログラム起動という形として扱われるオブジェクトへのメッセージパッシングは，シミュレーション時のオブジェクト間の相互作用と振舞いをそのまま扱うことができる．そして委譲機能は，シミュレーションシステムにおける柔軟な変更を実現している．さらにU-FSとの上位互換性は，従来環境であるUNIXのユーザに継続性の良い環境を提供し，ユーザの心理的な不慣れ感を取り除いている．また，メソッドの記述に任意の言語が使えることなどから，従来のプログラムやデータファイルはわずかな変更作業のみで高い再利用性を実現し，莫大な過去のソフトウェアの継続性を実現している．

以上を総合すると，本PBO-FSはオブジェクトシミュレーションをはじめとするオブジェクトの管理システムとしての機能性に関して従来のUNIX実行環境に比べて高いといえる．

5.2 オブジェクトの粒度

PBO-FSをOODBMSのように言語システム内のオブジェクトを永続化させる機構として考えると，オブジェクトの粒度が問題になるように思える．しかし，PBO-FSはファイルシステムであり，その目的はプログラムをオブジェクトのメソッドとして扱うことである．そのためPBO-FSにおけるオブジェクトの粒度は，アプリケーションとそれによってアクセスされるデータ群である．プログラミング言語内で用いられるオブジェクト群の永続性管理に関しては，従来のファイルシステムと同様に，ファイル内にシリアライズすることで行う．

5.3 他の研究との比較・考察

初めに他のオブジェクト指向ファイルシステムとの比較を行う．まずオブジェクト指向ファイルシステムとしては，ファイルシステムのインタフェース部を抽象データ型として扱う研究があげられる[9)~11)]．また代表的なファイルシステムであるU-FSに関しても，すでに複数のファイルシステムに対応する必要性などから，カーネル内部においてこれと同様の概念が用いられている[18)]．これらの研究においては，オブジェクトはファイルもしくはその内部要素であり，メソッドはそれを操作するための各関数群である．本研究ではオブジェクトのメソッドはプログラムファイルであり，ディレクトリはプロトタイプベースオブジェクトである．このように本研究とこれらではオブジェクトの定義やその目的に違いがある．

ここで，近年OLE[20)]などの複合ドキュメント技術においてはファイル内部に仮想的なファイルシステム構造を考えるようになってきており，その観点からファイルが内部にディレクトリ的な構造を持つと考えれば，本研究の概念はファイル自身もしくはその内部構造に対しても応用可能である．

次に，別のオブジェクト指向ファイルシステムの研究としては，永続オブジェクト管理機構を用いて階層型ファイルシステムを構築するというオブジェクト指向OSのサブシステムとしての研究が存在する[12)]．本研究はこれらとは異なり，従来の階層型ファイルシステムに対する視点を変更し，3章で述べた3項目の機構を追加することで，オブジェクト指向ファイルシステムを実現している．これは手続型言語であるC言語の構造体をクラスと見なすことにより，オブジェクト指向の概念を導入したC++の手法[17)]と同様な方法である．PBO-FSでは，ディレクトリをオブジェクトとすることで従来環境との上位互換性と移植性を保っている．

次にPBO-FSの各構成要素に関する比較を行う．まず，ディレクトリをプロトタイプベースオブジェクトと見なすのが，本研究の基盤となる概念である．これに関連する研究としては，二村ら[21)]の研究においてもディレクトリ階層をオブジェクト階層と見なすという考えが示されている．しかしこれは，ハイパリンクの管理にディレクトリを用いたものであり，本研究とは直接関連しない．

また，PBO-FSの構成要素それぞれに関しては本研究と類似の概念を他研究に見出すことができる．例として，環境変数や機種に応じてシンボリックリンク参照先が変更されるAFS[22)]などである．また，ディレクトリの合成に関しては，ファイルシステムマウントの際にディレクトリの合成が行われる半透明ファイルサービス[23)]などがある．これらと委譲リンクは，ディレクトリの合成が発生するという点などで類似点が存

在するが，オブジェクト指向の概念を導入したものは見当たらない．ディレクトリをオブジェクトと見なしたうえで，委譲の概念をファイルシステムに導入するのは，本研究が初めてである．

以上のことから，ファイルシステムに対してプロトタイプベースオブジェクトの概念を適用するという本研究の方法論に対して，直接比較評価すべき類似な概念や実装が他に見当たらないのが現状である．

6. 結　　論

本研究ではまず，U-FS に対してプロトタイプベースオブジェクトの概念を導入可能であるということを指摘した．そしてそれを実現するための機構として，ディレクトリを情報隠蔽するアクセス権限，委譲リンク，セルフディレクトリ機構を新規に提案した．またこれに基づいて実際に PBO-FS の設計・実装を行うとともに，応用の典型例を示すことで，考察と評価を行った．結論として本研究での重要なことがらは，プロトタイプベースオブジェクトの概念を従来 FS である U-FS 上に導入し，新規なオブジェクト指向ファイルシステムとして実装可能なことを実際に示したことである．

7. 今後の課題と展望

PBO-FS の今後の研究課題としては，分散ファイルシステムへの対応や，ファイル単位での委譲などの，より多様で柔軟な利用法のための拡張，他分野での利用法の開発と普及，等をあげることができる．現在その実現のために，本 PBO-FS を商用 U-FS 上に移植作業中である．

参 考 文 献

1) 畠山正行，金子　勇：オブジェクトベース機構に基づく数値シミュレーション，情報処理学会第 51 回プログラミング研究会研究報告，Vol.94, No.51, pp.1–8 (1994).
2) Hatakeyama, M., Kaneko, I. and Uehara, H.: Numerical Wind Tunnel Simulations for Arbitrarily Complex and/or Moving Test Bodies Based on the Object-Based Architecture and GUI, *Proc. 5th International Symposium on Computational Fluid Dynamics-Sendai*, Vol.1, pp.279–284 (1993).
3) Hatakeyama, M., Kaneko, I. and Uehara, H.: DSMC Analyses for Highly Complicated and Interactive Flow Based on the Object-Based Mechanism and GUI Environments, *Proc. 19th International Symposium on Rarefied Gas Dynamics*, Vol.1, pp.1175–1181 (1994).
4) Hatakeyama, M., Akita, K., Hasegawa, Y., Takimoto, N. and Watanabe, M.: Object Based Numerical Wind Tunnel System with Integrated Support Environments, *Proc. International Conference on Computational Engineering Science (ICES) '95*, Hawaii, USA, Vol.1, pp.27–32 (1995).
5) Goldberg, A. and Robson, D., 相磯秀夫（監訳）：SMALLTALK-80—言語詳解，オーム社 (1987).
6) Computational Mechanics'95, *Proc. ICES'95*, Vol.1, pp.14–68 (1995).
7) Khoshafian, S.: *OBJECT-ORIENTED DATABASES*, John Wiley & Sons, 野口喜洋，小川束（訳）：オブジェクト指向データベース，*bit* 別冊，共立出版 (1995).
8) ONTOS DB3.0 マニュアル，ONTOS (1994).
9) Crawley, S.: Object-Based File System for Large Scale Applications., *Software Engineering Environments*, IEE Computing Series 7, Peter Peregrinus, London (1993).
10) Karpovich, J.F., Grimshaw, A.S. and French, J.C.: Extensible File Systems (ELFS): An Object-Oriented Approach to High Performance File I/O, *SIGPLAN Notices*, Vol.29, No.10, pp.191–204 (1994).
11) Smolik, T.: An Object-Oriented File System-An Example of Using the Class Hierarchy Framework Concept, *Oper. Syst. Rev.*, Vol.29, No.2, pp33–53 (1995).
12) Kato, K., Inohara, S., Narita, A., Chiba, S. and Masuda, T.: Design of the XERO open distributed operating system, *3rd Computer System Symposium Tokyo, Jpn, Journal of Information Processing*, Vol.14, No.4, pp.384–397 (1991).
13) Smith, R.B., Lentczner, M., Smith, W.R., Taivalsaari, A. and Ungar, D.: Prototype-Based Languages: Object Lessons from Class-Free Programming, *Proc. OOPSLA '94*, pp.102–112 (1994).
14) Ungar, D. and Smith, R.B.: Self: The Power of Simplicity, *Proc. OOPSLA '87*, ACM Press, also appeared in *SIGPLAN Notices*, Vol.22, No.12 (1987).
15) 畠山正行，金子　勇：オブジェクトシミュレーションシステムへの UNIX システムの再構成，情報処理学会論文誌，Vol.39, No.7, pp.2060–2073 (1998).
16) Tanenbaum, A.S.: *Modern Operating Systems*, Prentice-Hall (1992).
17) Stroustrup, B.: *The C++ Programing Language*, Addison-Wesley (1986).

18) Kleiman, S.R.: Vnodes: An Architecture for Multiple File System Types in Sun UNIX, *Proc. Summer Conference*, UNENIX Association, Atlanta (1986).
19) Rumbaugh, J., et al.: *Object-Oriented Modeling and Design*, Prentice Hall (1991).
20) Brockschmidt, K.: *Inside OLE*, 2nd Edition, Microsoft Press (1995).
21) 二村祥一，菊池正城，脊古裕司：ディレクトリ階層を利用した分散形ハイパーテキストシステムの設計と実現，情報処理学会研究報告，Vol.96, No.15 (CH-29), pp.19–24 (1996).
22) Morris, J.H., Satyanarayanan, M., Conner, M.H., Howard, J.H., Rosenthal D.S. and Smith F.D.: Andrew: A Distributed Personal Computing Environment, *Comm. ACM*, Vol.29, pp.184–201 (1986).
23) SUN Microsystems: SUN OS Reference Manual (1988).

(平成 9 年 6 月 25 日受付)
(平成 10 年 7 月 3 日採録)

畠山 正行（正会員）

1947 年生．1976 年東京大学大学院博士課程（航空学専攻）修了．工学博士．東京都立航空高専を経て，1990 年 10 月より，茨城大学工学部情報工学科専任講師，現在に至る．この間，超音速凝縮流れ等の非連続気体流れの力学の研究に従事．ついで，数値シミュレーションユーザ用の計算機環境，オブジェクト指向シミュレーション，エージェントを用いた数値シミュレーション手法の研究開発に従事．日本機械学会，日本航空宇宙学会，日本数値流体力学会，日本シミュレーション学会等各会員．なお，本プロトタイプベースオブジェクトファイルシステムは，著者らの研究室の web ページにて以下の URL で公開されている．OS は FreeBSD2.2.6（4.4BSD Lite）である．http://poseidon.cis.ibaraki.ac.jp/pbofs.html

金子 勇（学生会員）

1970 年生．1995 年茨城大学大学院博士前期課程（情報工学専攻）修了，同年同大学大学院博士後期課程（情報・システム科学専攻）入学，現在に至る．この間，オブジェクト指向に基づくシミュレーション環境の研究開発に従事．