# 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

畠山正行[†] 金子勇[††]

オブジェクトに基づくシミュレーションシステムは，大きな特長を持つにもかかわらず，従来のモデリング方法との差異による違和感やプログラミング（実装）の難しさ等の実用技術上の難点のゆえにその普及ははかばかしくない．そこで本研究では，これらを克服すべくUNIXシステム上にオブジェクトシミュレーションシステムを構築する新規な方法論を提唱し，従来のシミュレーションシステムやUNIX環境との整合性が良くプログラム言語に依存しない，実用性の高いシステム構築方法の確立を目指す．そのために，UNIXファイルシステムのディレクトリをオブジェクト単位ととらえ，その構成要素であるファイルへのアクセスを同一ディレクトリ内からのプロセスに制限する．その目的実現のため，我々はオブジェクトの構成とUNIXシステムとの対応関係を見直してわずかなシステムを加えた．それを用いることでカプセル化および情報隠蔽化されたプロトタイプベース・オブジェクトの構成要件を満たし，オブジェクトとしての振舞いや相互作用条件も満たした汎用的なオブジェクトシステムとして再構成した．これに基づいてオブジェクトシミュレーションの実際例を構築・駆動させ，当初の目標をほぼ達成した．著者たち自身の使用経験によれば，従来のシステムと比較しても実装の簡潔性，分かりやすさ，使いやすさ，表現能力は良好である．結論として，すぐにでも実装・実行可能なほどに実用性の優れたオブジェクトシミュレーションシステムがUNIXシステムを再構成することで，実際に構築され，高い有効性や有用性が検証された．

# Re-constitution of UNIX System toward Object Simulation System

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

The Object-oriented systems have often been developed, but, have not widely been applied and also not accepted especially to the domain users in the fields of the natural science or of the technology. Then, in this paper, we aim at establishing an implementation methodology for a new Object simulation system by which the domain users can easily and simply realize and drive the Object simulations. We have devised and proposed first a new fundamental idea to regard a directory of the UNIX file system as the aggregation Object unit. The second new idea is to restrict the access privilege to each object (=directory) from the UNIX processes. The Object system implementation methodology based on these two ideas has been developed and established by changing the view of the UNIX system, and by adding some constraints to realize the encapsulation and the information hiding of the Object, and finally by re-constituting the Object simulation system. This methodology has been applied to the wind tunnel flow simulations. The results are fine. The implementation procedures are very simple and practical. We can conclude that the implementation methodology we have proposed has successfully been established, and the implemented example system has verified the usefulness and simplicity for the domain users.

## 1. はじめに

本研究の目指すものは，特定のプログラミング言語に依存せず従来のシミュレーション環境との整合性も良いシンプルで実用性の十分高いUNIXシステム上でのオブジェクトシミュレーション（オブジェクトを基本単位とし，オブジェクト間の相互作用により対象世界のシミュレーションを現出させる方式全般を指す広い意味を持たせた用語）の実現方法論とその実装・実用例である．

UNIXシステムは現在，コンピュータシミュレーションが行われる最も一般的な駆動環境となっている．こ

† 茨城大学工学部情報工学科
Department of Computer and Infomation Sciences, Faculty of Engineering, Ibaraki University
†† 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

れはワークステーションのみならず，スーパコンピュータその他においてもそうである．しかしUNIX上での現状のシミュレーションの実現法は依然として従来からの手続き型のプログラミング言語を用いて作成したシミュレーションプログラムの実行に終始しているものも多い．したがって，モデル表現精度や，シミュレーションの実現・駆動，シミュレーションシステム（本論文では1つのシステムとして構成されたシミュレーションのプログラム群を指す）の駆動形式等々に問題の残る実現方法に基づいていることも少なくなく，UNIXシステムが持つシミュレーションを駆動させる環境としての能力を十分利用していないように考えられる．

本研究では，あらかじめ,「シミュレーション」の議論と実現を明確・確実にするためにシミュレーションの対象世界を流体流れのような自然現象の再現シミュレーションに限定する．そのような自然世界のシミュレーションにおいては現象を構成する「もの」のモデル化は,「もの」の構造とその構造から選択的に構成される機構，およびその機構を起動させて形成される振舞いによって構成されるものであると我々は考えている．そういった観点からすると，構造や機構を記述するのではなく振舞いの現出結果を記述する手続き型のプログラムによるシミュレーションは，いくつかの欠点を持つといえる．たとえば，その記述・駆動方式をはじめとして対象世界のモデル表現精度，トライ&エラーに対して柔軟なシミュレーションシステムという観点，UNIXシステムとの整合性という点，等々から見て，ドメインユーザ（本論文では情報科学・工学以外のドメイン（専門分野）のコンピュータユーザを指す．以下同様）の満足できるシミュレーションシステムではない．

このような欠点を解決する対策として,「もの」の複雑な構造を精度良く簡潔に表現できるモデリングおよびプログラミングのパラダイムとしてオブジェクト指向があり，我々もこれを用いたシミュレーションシステムを構築しいくつかの良好な結果を出している[1)~4)]．しかし，大規模・複雑な物理現象シミュレーションにも適用が可能で，シミュレーションシステムの諸構成の変更が容易である，等の多くの長所も報告[1)~4)]されているにもかかわらず，オブジェクト指向について熟知しているユーザに対して以外はそれらの長所に見あう普及とは程遠いのが現状である．我々の指摘できる一般的な原因はオブジェクト指向の概念や理論あるいはシステム構築法等の理論的な側面の問題ではなく，むしろ実装・実現方法の分かりにくさ・煩雑さ・違和感等であるのではないか，という点である．

それらに対しても我々は，多少は不完全でも平易にオブジェクトシステム（オブジェクトを基本単位としたシステム全般を指す広い意味を持たせた用語）が構築・駆動できる方法を構想・実現してきた[1),2),4),5)]．そのようなオブジェクトの持つべき条件とは，対象世界の相似なモデル化ということである．いい直すと，各モデル化単位がカプセル化（付録の用語説明参照）されかつ適切に情報隠蔽（付録参照）されてオブジェクトと定義でき，そのモデルの静的側面（すなわち構造）と動的側面（すなわち時間的変動・振舞い）を1セットでオブジェクト内部に定義でき，そのカプセル化単位と対象世界のモデル化単位とが1対1対応の関連を付けることができ，最後にそれらが相互作用をしながら駆動（振舞い）できる，ということである，と我々は考えた．ただし，オブジェクト指向を単にプログラミングテクニックとしての利用だけではなく，モデリング方法論として用いて構成したモデルを基盤として，オブジェクトシミュレーションシステム（オブジェクトシステム上で駆動されるシミュレーションシステムを指す）の実現方法論を議論し構想してきた[1),2)]．

そのような経緯を受け，本論文においてはまず次の2章において，改めてドメインユーザの要求記述を試みる．その結果をふまえて，3章においてそれらの要求を実現するオブジェクトの構成について2つの側面からの重要な指摘を行う．すなわち，まずUNIXのファイルシステムの概念や仕組みをほとんどそのまま利用し，それらの位置付け・意味解釈・見方を変更し，わずかな実装上の制約と専用の関数やメソッドを実現することでUNIXのファイルシステム上にプロトタイプベースのオブジェクトを基本構造単位として定義・表現することが可能であることの指摘を行う．そして，もう1つはこうして定義されたオブジェクトの相互作用や振舞いの側面についても，UNIXのプロセスに対してわずかな制約付加と簡潔な特定の実装手順を設けることで，ほとんどそのままオブジェクトの相互作用や振舞いとして解釈して実現できることを見いだせたことを指摘する．以上の2側面を総合し，UNIXシステム特にファイルシステムとプロセスに対する見方・考え方の転換と制約の付け方に基づく再構成の方針を考察・提案する．4章ではその方針に基づくオブジェクトの実装方法を提案し，5章では実装法に対応するオブジェクトモデルを考察する．6章でシミュレーションシステムとして実装し，その実装・駆動例を考察する．7章は本研究の実装方法論の実現を評価する．

## 2. オブジェクトおよび駆動環境への要求記述

ドメインユーザの目指すシミュレーションへの要求を，著者たちやその周囲のドメインユーザの議論と経験のまとめ[6),7)]として再検討する．ドメインユーザとシステム構築者の間には，相互にまず相手の要求仕様あるいは可能な機能の全貌の提示を要求し，それが分かった後で自分の解答のフルセットを相互に相手に示せる（あるいは示したい）という「双すくみ状態」となっている場合が多い．それゆえにシミュレーションシステムについてのシステム構築者に対するドメインユーザの要求記述の全貌はいまだ見えていない．そこで，筆者たちが構築してきたシミュレーション駆動をさせる環境[3),8)]とシミュレーション対象世界の設計・構築・駆動の実績と経験から以下のようなオブジェクトシミュレーションシステムに対する要求記述を，可能な限り第三者的な立場から抽出・抽象化した．まず，UNIX 環境に対しては

(1) 現行の UNIX 上での実行形態をほとんど変えない上位互換のオブジェクトシミュレーション計算環境であること．
(2) モデル化とプログラミング（実装）方法が簡単で，従来の手続き型の方法との違いが少なく，実装上の制約が緩やかで，使いやすいシステムであること．
(3) 新規なプログラミング言語の回避，あるいはプログラミング言語からの独立性．

上記 (3) については，ドメインユーザは新規言語の修得に対しては保守的なので FORTRAN，C 言語，せいぜい C++，およびその混合使用または UNIX 上のユーザ指定言語が望ましい．そのようなシミュレーション駆動環境の中でシミュレーション対象世界の各オブジェクトおよびその要素に対しての条件を述べる．

まずドメインユーザの日的にとっては，生成結果のオブジェクト単位モジュールがオブジェクトの条件を満たしていればよく，生成方法の違い等は必須な条件ではない，ということである．したがって，本論文におけるオブジェクトの持つべき条件は広義のオブジェクトシステム（付録：用語説明参照）でよく，狭義のオブジェクト「指向」[9)]システム（付録参照）やプロトタイプベースオブジェクト指向システム（付録参照）が必須条件ではない，という点である．もちろん，利用できる言語，実装および駆動の実用性，効率，利便性等を考慮に入れると，付加的な条件要素がそれぞれのシミュレーションの目的や事情によって付加・導入される可能性は当然考えられる．

オブジェクトに対する以下の諸条件もドメインユーザとシステム構成者との議論から数年の経験と 1 章のまえがきの議論を含めてまとめたものである．

(4) 分離記述・格納されている属性（データ）とメソッドの明示的な一体化（カプセル化）．
(5) 各モデル化単位オブジェクトでの属性（データ）の適切な情報隠蔽化．
(6) 各オブジェクト単位と対象世界のモデル化単位とが 1 対 1 対応していること．構成要素についても，また，モデル化単位間の相互関連についても同様に 1 対 1 対応していること（構造の相似性）．
(7) 個々のオブジェクト単位での起動・駆動・制御・停止．さらには個々のオブジェクトの振舞い，およびオブジェクト間の相互作用の結果としての振舞いが，対象世界の対応するそれらと相似なスタイルであること．これはメッセージパッシング概念の必要性を指す（振舞いの相似性）．

## 3. ファイルシステム上のオブジェクト構成方針

### 3.1 オブジェクト構成への基本アイデアとそのモデル概念構成

本論文ではまず UNIX システムに対する「見なし方，観点」に新たな 2 つのアイデアを適用し，UNIX システムに対する見方を変更する．第 1 には

**（1）ファイルシステムにおけるディレクトリをカプセル化されたオブジェクトとしてとらえること．**

すなわち，UNIX の通常ファイルをオブジェクトのデータ構造（属性）部分と見なし，実行可能ファイルをオブジェクトのメソッド部分に対応する要素と考え直す．そうすると，最も下の枝のディレクトリは最小単位オブジェクトであり，その上位のディレクトリはなんらかの意味での複合集約階層構造オブジェクトを表現していることになる．

以上の考え方を基に**図 1** に現状の UNIX システムをオブジェクトシステムと見なしたときのオブジェクトモデル表現を試みた．図 1 におけるディレクトリを Object，通常ファイルを Data，実行可能ファイルを Method，UNIX のプロセスが Process に対応させることができる．またファイルシステムにおけるディレクトリの階層関係はオブジェクトの集約構造に，そしてシンボリックリンクはオブジェクト間の関連表現に対応する．また，ディレクトリとはファイルやディレクトリの再帰的な集約であり，ある抽象名を持つ単位オブジェクトである．そのため，この抽象名としてオ

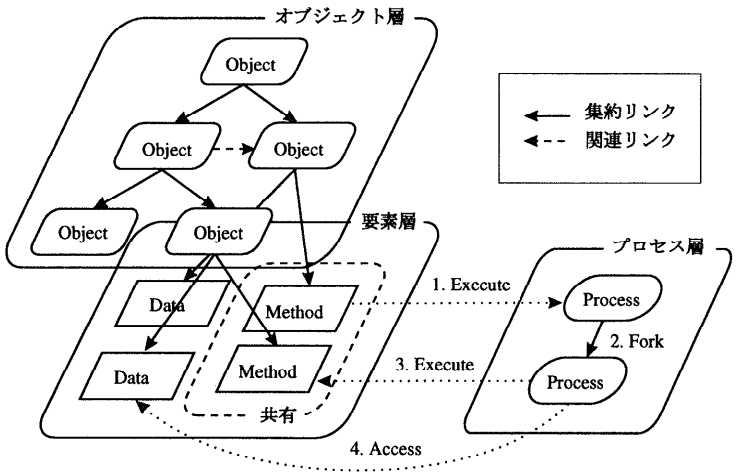


図1 UNIXシステムが持つオブジェクトモデル
Fig. 1 Object model mapped on UNIX system.

表1 UNIXシステムとオブジェクトシステムの構成要素の対応関係
Table 1 Relationships between UNIX system and Object system.

| UNIXシステム | オブジェクトシステム |
|---|---|
| UNIXディレクトリ | 集約オブジェクト |
| 実行可能ファイル | メソッド |
| データファイル | 属性 |
| UNIXプロセス | オブジェクトの振舞い |
| プロセス間通信 | オブジェクト間相互作用 |

ブジェクト名をあてる．あるディレクトリ下でデータとメソッドを一体化させるには同一ディレクトリ内で図1のようにリンクを張るか，関連を記述すればよい．これを実装すれば，前章の(4)や(6)の条件によく適合する．これにより，結果としてデータとメソッドが情報隠蔽化されるように意識的に注意を払ったプログラムを作ることでオブジェクトシステムが実装可能である．**表1**にUNIXシステムとオブジェクトシステムの構成要素に対する基本的な対応関係を示す．

次に，各ディレクトリがオブジェクトの定義を満たす資格（情報隠蔽化）を得させるために，

**（2）そのオブジェクトへのアクセスを同一ディレクトリ内の実行可能ファイルから生成されたプロセスのみに制限する．**

これが本論文で提案する第2の最も核心となるアイデアである．このアイデアが実現すると，オブジェクト単位の駆動（振舞い）の側面であるUNIXのプロセスがそのプロセスを立ち上げたオブジェクト（ディレクトリ）に対してのみデータアクセスできるようになるので，情報隠蔽化したオブジェクトが実現する．しかしこの考えは現状のままのUNIXシステムでは実現困難であるので，なんらかの変更を提案する．他のディレクトリ（オブジェクト）のデータファイルをアクセスする際には直接アクセスできず，オブジェクトにメッセージを送り，そのディレクトリ（オブジェクト）内部の実行プログラムを起動して実行するように制限する手段を講じる必要がある．

以上の2つの基本アイデアおよびそれに連なる必要な構成が実現されれば2章の条件(4)～(7)が実現され，したがってオブジェクト間の相互作用方式が実現し，カプセル化と情報隠蔽化も実現し，オブジェクト単位ごとの振舞いが確立する．そこで次に，このような方針を基にしてオブジェクトに関する概念をファイルシステムへ導入することを個々に検討していく．

### 3.2 プロトタイプベースオブジェクトモデルの提案

UNIXのファイルシステムをオブジェクトシステムと見なす本研究の視点からは，オブジェクトをディレクトリに制限しているため，もしも狭義のオブジェクト指向[9]のクラスの考え方・見方・構築をこのファイルシステム上で実現するとすれば，ある程度の別途新規システム，つまり現状のファイルシステムとは独立にクラスを管理するオブジェクト管理システムを作成する必要が生じる．この方向の考え方はドメインユーザの要求記述（2章の(1)の条件）とは相容れない．

そもそも，ドメインユーザがオブジェクトを作る基本的な方法は，むしろ他のオブジェクトのコピーを作りそれを出発点にして目指すプログラムを作ることが多い．ところで，このような方法はオブジェクトのモデルの中でクラスを考えないプロトタイプベースモデ

ルを用いていることに近い．さらには2章の要求条件(2)を満たす必要からも本研究のオブジェクトの基本型はプロトタイプベースモデルとした．以下ではこの基本モデルを前提に考察を進める．

### 3.3 メソッドの共有

プロトタイプベースのオブジェクトモデルを仮定すると，オブジェクト間のメソッドをどのようにして共有するのかについて考える必要が出てくる．これに関して，プロトタイプベースオブジェクトモデルの研究[10),11)]から，メソッドの共有に関して大きく分けて2つの考えがあることが分かっている．すなわち，Kevo[10)]のようにオブジェクト生成の際にメソッドまでもコピーしてしまう方法と，Self[11)]のようにメソッド検索のためのリンクを用いる方法である．

そのため，図1のモデルにおけるメソッド共有法として，メソッドに相当する実行ファイルをハードリンクにより共有する方法と，プログラムを検索するためのパスをオブジェクト（=ディレクトリ）ごとに設定する方法が考えられる．図1はハードリンクによるメソッドの共有を表現しており前者である．後者のモデルを採用するには，4.2節で述べる我々の提案する委譲機構（委譲リンク）の新規実装をUNIXに導入することで容易にオブジェクトシステムを実現できる．

### 3.4 オブジェクトの振舞い

オブジェクトの振舞いに関する最も基本的な方法として，オブジェクト（構造表現）層とそのオブジェクトにアクセスするプロセス（振舞い駆動）層とに分離するモデル表現法が考えられる．図1の再構成表現はこれであり，あるプロセスがメソッドを実行中に他のオブジェクトのメソッドを呼ぶ場合には，そのプロセス自身が他のオブジェクト内のメソッドを直接解釈する，もしくは新しいプロセスを生成したうえでメソッドを解釈する（図1内の1~3）．もう1つの振舞いモデルとしては，プロセスをオブジェクト内部に集約されているものと考えることもできる．このモデルの場合，すべてのオブジェクト内にはプロセスが存在し，このプロセス間の通信によりオブジェクトの振舞いが行われるものとしてモデル化できる．しかしこのモデルの場合，カプセル化という点では優れているがプロセス数が多くなりすぎるという問題が発生する．

そのため，本論文では「オブジェクトの活性」という概念を導入し，非活性時のオブジェクトは内部にプロセスが含まれていないが，活性化することで内部にプロセスが生成されるものとする．この方法により不要なプロセスを省略する．このモデルの場合プロセスの生成をどのように考えるかが問題となるが，メソッドが活性化することでプロセスが生成されるという考えと，オブジェクトが活性化した時点でメッセージを解釈するプロセスが生成され，このプロセスがメソッドにアクセスするという2種類のモデルが考えられる．

### 3.5 オブジェクト間の相互作用（通信）

本研究ではストリームに基づくメッセージモデルを採用した．ストリームモデルはストリームと呼ばれる通信路（リンク）を通じてメッセージを送る[12)]．返答用には別にストリームを並行に設けるのでシミュレーション（本研究では流体の振舞いの再現）の際の大量のデータの高速転送を容易に実現できる．

### 3.6 オブジェクトの構成

実際に，いくつかの議論・考察と，具体的な構成要素の選択組合せのほとんどを機能確認のために部分的に試作として実装した．その結果と経験から得られた知見から，次の選択組合せに実現の可能性と有効性・有用性を見い出した．すなわち，表1のような対応関係に加えて，

(1) オブジェクトモデルの基本型はプロトタイプベースオブジェクトモデルである．
(2) メソッド共有にはメソッド検索パスを用いる．
(3) プロセスはオブジェクトの活性化概念を採用する．
(4) 通信モデルはストリームモデルとする．

このモデルを図に表現すると，**図2**のようになる．UNIXファイルシステムのオブジェクトモデル再構成において，他にも幾種類かの良好な組合せも存在するが，より多様なシミュレーションに適応・応用可能なのではないかと考えたのがこの組合せである．モデル化する対象世界によっては上記以外の構成モデルも採用可能である．オブジェクトの活性化の概念とその実装方法は4.3節で述べる．

以上の考察から，以下では本節で提案した構成のオブジェクト実装モデルを採用した．

## 4. オブジェクトの実装方法

### 4.1 情報隠蔽化

データファイル（=通常ファイル，属性データファイル）を情報隠蔽化するためには，あるデータファイルにアクセスする（できる）プロセスを，そのプロセスの元となるメソッドファイルと同一ディレクトリにあるデータファイルに限ることができるような仕掛けが必要である．これについてはファイル（=属性データ）操作のためのfopenの代わりとなる関数（具体的にはOB_fopen）を提供することで実現している[13)]．ファイルへのアクセスが，すべてOB_fopenを用いる

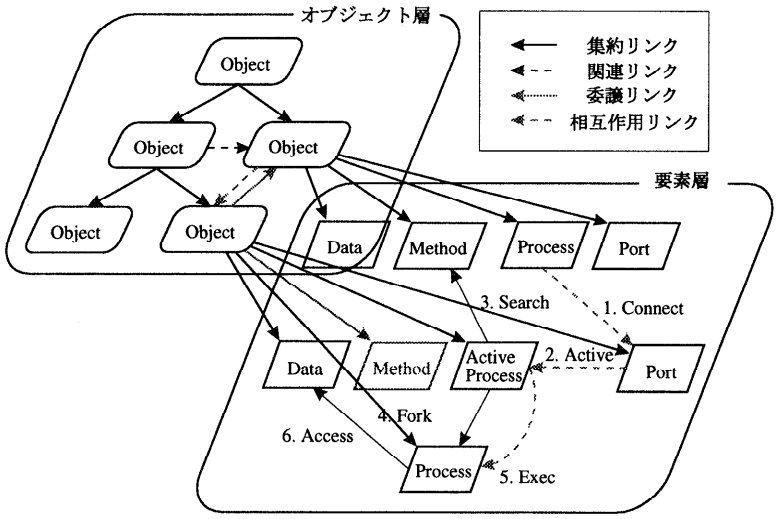


図2 UNIXシステムのオブジェクトシステムへの再構成
Fig. 2 Re-constitution of UNIX system toward Object system.

ように実装されていれば，ファイル（オブジェクトの属性・データを格納してある）は情報隠蔽化が保証される．

さらにいえば，3章の実装モデルと本章の以降の節で示す実装方法に準拠したシステムの設計と構築を行い，かつ，それに対応して整備されて提供されているメソッド群や十数個のOB_call関数群[13]を用いて構築されていることにより，ほぼ完全な情報隠蔽化が実現する．なお，いまここで記述した構築に関する実装方法は4.2節以降の方法の実現法も含んでいる．

### 4.2 委　　譲

メソッドの共有はプロトタイプベースオブジェクトモデルにおける委譲の概念を導入することで実現する．これはディレクトリごとにメソッド検索パスを持たせることで実現される．すなわち，あるディレクトリ内の実行可能ファイルを実行する際にそのファイルがなかった場合，他のディレクトリ内の実行ファイルを，元のディレクトリにあったと見なして実行する機構である．これはUNIXの実行ファイル検索パスと同様の考えにより，容易に実装できる．具体的には，UNIXでは環境変数PATHで表される実行ファイル検索パスをプロセスごとに持つことになっているが，これをディレクトリごとに持つというルールをつねに実装ユーザが守って実装する．この検索パスを委譲リンクと称する．

この，ディレクトリ単位に実行ファイル検索パスを持つという考えは，前節で述べた通常ファイルが“各ディレクトリ内で”の情報隠蔽が実現できていなくてはならないという制約に対しては厳密には問題が起こる可能性がある．しかし，委譲機能により検索されたメソッドがアクセスする属性ファイル（通常データファイル）は検索元のオブジェクト（ディレクトリ）のみである．したがって，委譲機能によって情報隠蔽化が破れるわけではないという意味において，この委譲機能は論理的には許容できる実装である．この委譲機能は実装面から見た本研究における最も重要なアイデアであることを強調しておく．本研究の実装においては，メソッドの実行は後で述べるアクティブプロセスにより行われ，このアクティブプロセスが委譲リスト（実体は特定の名前のファイル）を参照し，その中から実行可能なファイルを検索することで委譲を実現している．

### 4.3 プロセス

オブジェクトの振舞いは，メソッド（=実行ファイル）をプロセスに実行させることで実現される．実際にメソッドを実行するのは，同一オブジェクト内のアクティブプロセス（図2参照）から生成されたプロセスである．アクティブプロセスはメッセージを解釈するプロセスであり，このアクティブプロセスは起動されるオブジェクトごとに生成され，オブジェクトの活性を表す．

メソッド起動の手順は，まずオブジェクトの要素であるプロセスが他のオブジェクトへ接続要求を行うことから始まる（図2での1. Connect）．もし，接続先のオブジェクトが活性化している，すなわち，内部にアクティブプロセスが存在している状態ならばただち

に接続を行う．オブジェクトが活性化していなければ，オブジェクト内部のアクティブプロセスを作るためのメソッドであるアクティブメソッドからアクティブプロセスが生成されて（2. Active），このプロセスにストリームが接続される．もし，アクティブメソッドが見つからない場合，他のメソッドと同様に委譲により処理される．そのあと，呼び出し側のオブジェクトがメソッド起動の要求を行い，その結果，呼び出された側のプロセスでメソッドが検索され（3. Search），新しいプロセスがアクティブプロセスのコピーにより生成される（4. Fork）．このプロセスによりメソッドが実行されて（5. Exec），その結果がストリームを通して呼出し側に返される（6. Access）．メソッドの起動は基本的に実行ファイルの実行であるが，アクティブプロセス内の関数の実行もメソッドの起動と見なすことができる．これによりプログラムの実行と関数の実行がともにオブジェクトのメソッドの起動として扱うことが可能となる．

### 4.4 生　　成

オブジェクトは，プロトタイプとするオブジェクトをコピーすることで生成される．集約の要素オブジェクト（=サブディレクトリ）が存在する場合，これも再帰的にコピーされる．関連を表すシンボリックリンクが存在している場合，リンク先のみコピーする．また，オブジェクト生成時はデータのみコピーし，メソッドはコピーしない．ただし，作成されたオブジェクトの委譲先にはオブジェクトのコピー元の委譲情報が追加されるため，新しく生成されたオブジェクトもコピー元と同様のメソッドを有する．これは，オブジェクトごとに持つメソッド検索パスの合成により実現されている．

### 4.5 名前付け

このモデルにおけるオブジェクトの指定はファイルシステムとまったく同一の方法によって行われる．集約による構造は木構造であり，すべてのオブジェクトの名前は，root オブジェクトからの集約名の合成で一意に決定することができる．また，1つのオブジェクトに対して，関連名（=シンボリックリンク）を用いることで複数の名前を付けることができる．これらはすべて UNIX の機能をそのまま用いる．

### 4.6 オブジェクトの実装と利用の部分例

このモデルの実装はメソッド記述の際に用いる C 言語のライブラリと，テンプレートとなる各種のオブジェクトの生成により実現されている[5]．ライブラリの基本的な機能は，他（もしくは自身）のオブジェクトのメソッド起動であり，

```
FILE *fp=OB_Fcall("r", "Object", "aMethod");
```

などの形で実装する．この例はオブジェクト Object のメソッド aMethod を起動し，ストリームの形でやり取りすることを意味する．この関数は fopen ライブラリ関数と同様の形で使用する．これらにより，オブジェクトのメソッド起動（プログラムもしくは関数の実行）やファイルなどの UNIX リソースへのアクセスをすべてオブジェクトのメソッド実行として行う方式が実現する．また 2 章で述べたように，ファイルへのアクセスはすべてメソッドと同一のディレクトリのファイル（データファイル）に制限されるが，それはこれらの関数により実現されており，これら以外の関数を用いてのデータファイルへのアクセスを禁止する．

メソッドの作成は基本的に実行プログラムの作成，もしくはアクティブメソッドにリンクする関数の生成であり，従来の開発環境を用いて行う．ライブラリ以外に，プロトタイプオブジェクト（後の図 6 の Prototype_directory）が作成されており，オブジェクトの生成や削除などの共通なメソッドはこれらのオブジェクトが保持している．そして，これらのプロトタイプオブジェクトのメソッドを起動することにより，新しいオブジェクトを生成する．ここで作成されたオブジェクトも委譲によりコピー元のメソッドを共有しており，これらを基に新たなオブジェクトの生成やメソッドの追加を行うことができる．

また，ユーザがオブジェクトを利用するために専用のシェルが作成されており，

```
# object/method argument1 argument2...
```

などとして用いる CUI のシェルや，OB シェルという名の GUI システムが利用できる（後の図 7 に実例）．なお多少の制限があるが，従来のシェルもオブジェクトの操作に使用できる．これは，メソッドの起動が基本的にプログラムの実行に相当するためである．

## 5. オブジェクト実装モデル

### 5.1 オブジェクトの構造モデル

以上の設計結果を総合的にまとめると，図 2 の表現するオブジェクトシステムを UNIX システム上に次のようなオブジェクトモデルを表現するシステムとして再構成できることが分かる．

オブジェクト ＝ データ＋メソッド＋相互関係リンク
　　＋ポート＋プロセス
相互関係リンク ＝（集約，関連，委譲
　　または相互作用の）リンク

- オブジェクトは**図 3** に示すように，その内部に構成要素として任意個数のスロット（=個別のファ

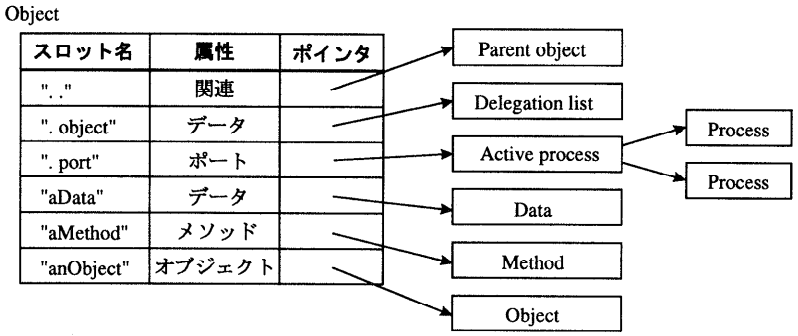


図3 オブジェクトの構造モデル
Fig. 3 Structure model of Object.

イル）を持つ．スロットはスロット名とスロット属性，ポインタを持つ．スロットには，集約スロットと関連スロットがある．

- 集約スロットの種類にはオブジェクト（サブディレクトリ），データ，メソッド，ポートがあり，オブジェクトの構成要素（ファイル）である．
- 集約スロットはオブジェクト生成時に再帰的にコピーが行われ，関連スロットはポインタのみがコピーされる．
- データはオブジェクトの属性をその内容として記述・表現し，任意長のバイトデータで構成する．
- データの一種として委譲リスト（ファイル）があり，アクティブプロセスがメソッドを検索する際に参照する．
- メソッドスロット（ファイル）はプロセスによって実行可能なプログラムである．
- ポートスロットはプロセスとのインタフェースを担当し，オブジェクト間相互作用を司る．

### 5.2 オブジェクトの振舞いモデル

- プロセスはメソッドを解釈実行することによってスロットやオブジェクト要素に対してアクセスを行う．
- データとメソッドへのアクセスおよびスロットの変更はアクティブプロセスもしくは，そのコピーにより生成されたプロセスに制限する．ただし，その例外として，委譲リスト先のオブジェクトに対してメソッドの読み出しのみ許可する．
- アクティブプロセスは，ポートスロットへのアクセスがあった際に生成され，その消滅はアクティブプロセス自身により行われる．
- プロセスはプロセスのコピーにより生成される．
- オブジェクト間相互作用（通信）モデルはストリームモデルである．

### 5.3 オブジェクトシステム

ドメインユーザから見ると，本システムのオブジェクトは図2のとおりUNIXのファイルシステム自体の構造に対象世界の構造をほぼ1対1対応でそのままマップする方法が実現されていることが分かる．また，各ディレクトリ内部にカプセル化されたオブジェクトの基本構成要素モジュールをスロットという形で持ち，複合集約階層オブジェクトはディレクトリの階層構造で表現できることが分かった．実際に次節でその例を示すが，これらの点は対象世界である物理世界のイメージ（認識モデル）を持ちつつ作業にあたっているドメインユーザにとっては，対象世界との相似イメージが容易に沸き起こり，従来のシミュレーションシステムには見い出すことが困難であった大きな特長を持つことが分かる．

その意味で，このオブジェクトモデルおよびオブジェクトシステムはマクロに見れば（各スロット内部をミクロに見なければ）対象世界に対する相似な構造モデルおよび構造化システムを近似的に表現しているといえよう．この構造化オブジェクトをオブジェクトベース（OB）オブジェクトと呼ぶことにする．さらに本論文で提案した実装モデルはOBモデルと呼び，このオブジェクトモデルで構成されて駆動するシミュレーションをOBシミュレーションと呼ぶことにする．

## 6. OBシミュレーションの実装例

前章までに提示された実装法およびモデルを用いてオブジェクトシミュレーションシステムを構築するには，以下のようなシステム構成の要件を満たして実装することが必要である．

- 本論文で提案するオブジェクトのモデルとその構築方式に準拠したシステム設計と構築手順・スタイルを用いること．
- オブジェクト実装内部に専用のメソッドや関数

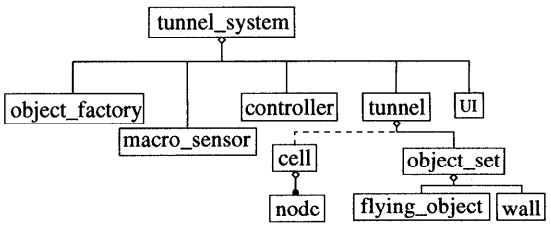


図4 数値風洞オブジェクトシミュレーションシステムの構成図
Fig. 4 Constitution of numerical wind tunnel Object simulation system.

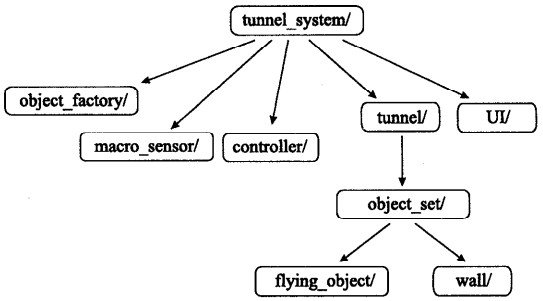


図5 数値風洞オブジェクトシミュレーションシステムの実装図
Fig. 5 Implementation of numerical wind tunnel Object simulation system.

（OB_fopen，OB_Fcall 等十数個程度を提供）を用いること．

本章では，これまでの実現方法の正当性を綜合的に示し，その実際を検証するために，ある特定の対象世界に対して適用し，実装を行った．実装は一例のみではなく，いくつかの実装と駆動例[3),5),8),13),14)]があり，それらの複数の実装・実現の過程，実装経験の背景をふまえて典型例[3),8)]を以下に記述する．

対象世界は二次元の連続流の数値風洞の世界である．**図4** に実装された数値風洞シミュレーションシステムを示す．この図は実際の数値風洞のかなり省略された主要部のみを OMT[15)] 類似の表記法を用いて示してある．矩形の枠はクラスを意味せず，プロトタイプベースのオブジェクトを示す．

対象世界である数値風洞全体 tunnel_system が，風洞と飛行物体の定義作成工場である object_factory，計測装置である macro_sensor，風洞本体である tunnel，風洞の制御装置 controller，ユーザの代理あるいはインタフェースである UI 等の集約から構成されていることを示している．tunnel 本体の下の object_set は飛行物体と風洞壁のセットであり，現在複数例作成されている．

tunnel 本体に集約されている cell とそれに集約されている複数（多数）の node は，風洞を流れる流体の物理量の変化をその基礎支配方程式から離散的に数値計算をするために設計モデル作成時に新たに加えられたオブジェクトである．この 2 つは別の意味でも特別なオブジェクトで，OB モデルではなく狭義のオブジェクト指向モデルに基づくクラス階層で実装されている．そう実装した理由は，図 4 において，cell は（たとえば）1800 個，node は約 20000 個以上になるからである．このような多数のオブジェクトを個々にオブジェクトとしてディレクトリに定義するのは種々の側面からみても現実的ではない．また，オブジェクトの実装を一部このようにしても数値風洞全体としての駆動上の不都合は起こらない．そこでこの部分については狭義のオブジェクト指向モデルをそのまま用いてクラス定義を行い，C++言語から生成されるメモリ上のみの言語オブジェクトとして実現した．したがって，OB モデルの特長はこの部分に限っては適用外である．この点の詳細な理由付けと考察は後の 7.3.2 項で述べる．

**図5** には，図 4 の数値風洞の設計モデルを OB オブジェクトとして UNIX ファイルシステム上に実装した例を示した．この図でも主要部以外はかなり省略されている．実際に本数値風洞用に定義され用意された OB オブジェクトの数は細かくいうと約 70 個，平均的に稼動する風洞内の OB オブジェクトの数は，たとえば 11 個から 15 個である．

図 5 から強調すべき重要なことは，図 4 の集約階層構造図がそのまま図 5 に示す UNIX のディレクトリ構造にほぼ相似な構造に実装（マップ）されているという点である．各オブジェクトの実装された UNIX のディレクトリ構造が図 4 の数値風洞システムの構造と相似でかつ 1 対 1 に対応した構造で作成できていることは両図の比較から一目瞭然である．ただし，クラスとして定義・実装された cell と node は，図 5 ではすでに tunnel 内に実装されていてディレクトリ構造には表現されていない．

このように対象世界との相似構造を本システムではシステム側で標準的にサポートしている．ドメインユーザにとってはこのような違和感の少ない対象世界の構造のマップが構築でき，（後で図 7 に示すように）簡単に可視化もできるというのは大きなメリットと考えられる．

さらに実装の詳細を説明するために，**図6** に flying_object 周辺の実装状況を示す．flying_object は属性を（データファイルとして）data，parameter，その他を持ち，メソッドを（各々を実行可能ファイルとして）viewer，output，その他を持つ．この構成は図 2 のオ

ブジェクトの構造モデルの一実装例でもある．これらの中で，メソッドファイルについてはflying_object内に実体としてカプセル化されたものももちろんあるが，viewer，outputの例のようにPrototype_directory内に共有し，4.2節で提案した委譲リンクによって実装と同等の効果を持たせたものもある．

flying_objectの実行可能ファイル検索パス（メソッド検索パス）は．objectに格納されており，アクティブプロセス（4.3節参照）が．objectの委譲リストを検索する．そしてアクティブプロセスは，図6の破線矢印に沿ってPrototype_directory内のflying_object，さらにたどってobjectにメソッド検索を行い，検索目的のメソッドを探し出して起動する．起動されたプロセスはflying_object内の属性（data，paranmeter）にアクセスしてflying_objectとしての実行（振舞い）を開始する，という手順になる．Prototype_directoryは全システムにおいてしばしば共通的に用いられるメソッドをこのオブジェクト内に置いておくことで，システム実装の効率化を図っている．

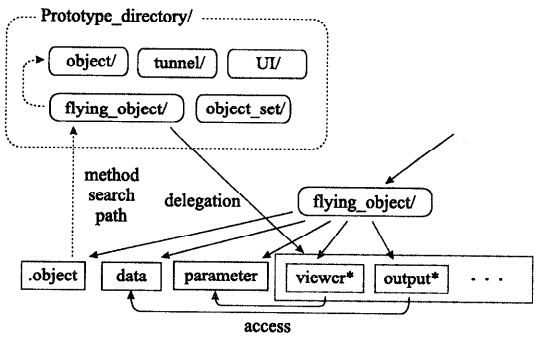


図6 数値風洞オブジェクトシミュレーションシステムの実装詳細図
Fig. 6 Detailed implementation of numerical wind tunnel Object simulation system.

図7にはオブジェクトシミュレーションの駆動例を示してある．この図7の内部の左上はオブジェクトシミュレーションシステムを駆動するためのUIオブジェクトのメソッドであるGUIベースのOBシェルが立ち上がっており，左下には同じくCUIベースのOBシェルも起動されてオブジェクトの操作を行っている．GUIベースのOBシェルはPrototype_directoryオブジェクトとobject_factoryオブジェクトの現在の実装状況を表示している．右上にはシミュレーション結果表示と分析ツールを使った可視化分析を行っていることを示している．左は流体中の物体である斜め板の直前方のnodeオブジェクトの配置状況をチェックしている様子である．右下はシミュレーション結果の等高

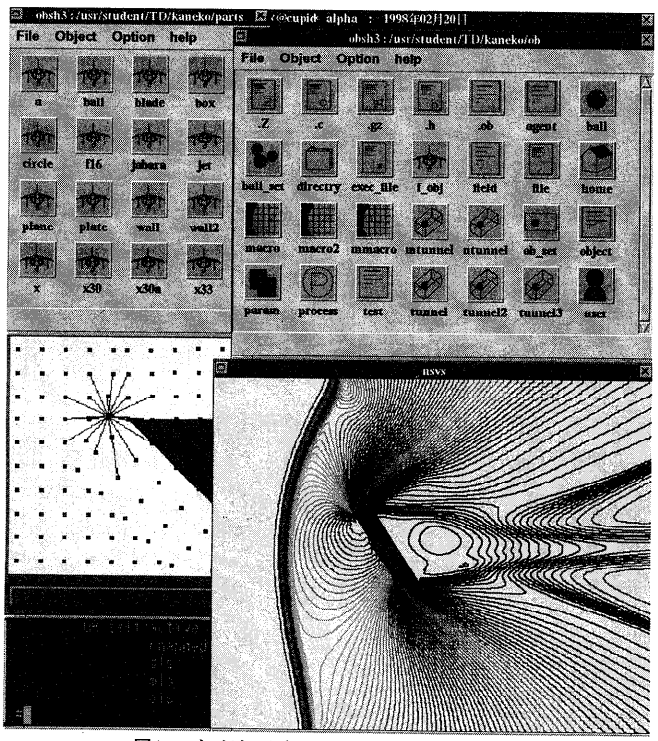


図7 オブジェクトシミュレーションの駆動例
Fig. 7 A representative example of Object simulation.

線表示である．これらは対象世界の駆動を支援する統合化された環境内で駆動している[3),8)]．

## 7. 考察と評価

### 7.1 OBシミュレーションシステムへの要求記述実現への評価

数値風洞のシミュレーションシステムを実装し使用した経験と結果から2章で述べたシミュレーションシステムへの要求記述の条件(1)～(7)が本研究の方法論の実装によりどの程度実現されているのかについて考察する．

駆動環境については，実装した環境が従来環境と同じUNIXでありディレクトリへの見方の読み替え概念さえ確実であれば，条件(1)はドメインユーザから見て十分満たしていると考えられる．プログラミング（実装）方法論の簡単さの条件(2)については新規のプログラミング言語やプログラミング技術の修得は不要であり，十分満たされている．

現行のUNIX計算機環境との整合性に関しては，従来環境であるファイルシステムとプロセス機構の見方を変化させただけであるため，緩やかな実装や従来環境との整合性があるだけではなく，従来シミュレーション環境そのものであるといえる．そのため，各種のUNIX計算機との混在に関しても問題はない．

条件(3)のプログラミング（記述）言語からの独立性についてであるが，ファイルシステムが特定の言語から独立しているのと同様に，本研究の考え方でもメソッドはプロセスが解釈可能なデータ列であればよく，メソッドの記述が特定の（あるいはドメインユーザにとって新規な）言語に依存する形にはなっていない．したがって，言語の混在（言語からの独立性）は何の問題もなく実現している．

条件(4)～(7)の実現について述べる．3.1節のアイデアがすでに実現されていることは3.2節以降4章までの記述で明らかである．したがって，オブジェクトモデルの枠組みが十分理解され，UNIXシステム内の要素とオブジェクトの対応関係が十分理解されていれば，オブジェクトを簡単に実装できることは実装理論上からもまた本研究での十分な実体験からも検証された．したがって，オブジェクトの実装は簡潔に実現された．実際上，4章および5章のオブジェクトシステム構築の簡潔なルールを守ることはドメインユーザにとってもまったく容易であり，これらを守れば各オブジェクトに対する条件(4)～(7)はその緩やかな制約であることも手伝って十分実現され，実装には不満・不便・不自由はほとんどなかった．

次にオブジェクト構造モデルは各オブジェクトごとのカプセル化を実装でき，シミュレーションに必要とされる要素および構造（主として集約階層構造）を格納できるディレクトリをオブジェクト基本構造として持っている．したがって，集約・集合・各種関連を持つ複雑な物理的な対象世界に対しても相似な構造を構築できることが分かる．したがって，構造モデル（およびその実装方法の4章）は，OBシミュレーション実現に適当かつ柔軟な構造を提供していることが分かる．この構造モデルは開発途上のメソッドやデータを臨機に管理するのにも都合良くできているのでユーザの工夫に応じて便利に使える．

オブジェクト振舞いモデルはすべてUNIXのプロセスに簡単に対応づけられる．つまり，その振舞いモデル駆動は従来からの手続き型シミュレーション駆動の各々に対応関係をとることが原理的には必ず可能である．決定的な違いは，その駆動（振舞い）をオブジェクト単位にすべて区別できる，つまり振舞いがオブジェクト単位での構造化を実現している点である．これはオブジェトシミュレーションを別の面からみて，オブジェクト単位でのシミュレーションシステムの変更を柔軟に可能にしている理由でもある．

相互作用（通信）モデルは高速なストリームモデルであり，シミュレーション向きに改良された適切なモデルおよび実装であることが分かった．

### 7.2 実装駆動例からの評価

本オブジェクトのモデル基本型はプロトタイプベース（コピーベース+委譲機能）である点に対しては，従来においては採用例の少ない基本型であるが，シミュレーションドメインにおいてはむしろ従来からのプログラム開発方式との差が少なく継続できる実装向きのオブジェクトモデルであり，ドメインユーザには使いやすく適切な基本型の選択であった．ドメインユーザ（著者の1人および同研究室内の複数の流体シミュレーションユーザ）用のシミュレーションシステムとしては構築に簡潔で使いやすいとの結論を得た．

さらには要求事項ではなかったが，オブジェクト，データ，ファイル間の関連の管理作業において，従来は「ユーザの注意のみ」で管理が行われてきた計算モジュールとデータの関連を，オブジェクトとしての形で「システムとして」の管理が実現し，各オブジェクト間の計算モジュール（メソッド）の共有を委譲によって管理できるようになったことが新たに加わったメリットとしてあげられる．また，実装のオブジェクト構造と対象世界のそれが従来よりも相似性が高いゆえに自然なオブジェクト構造配置が実現していること

で，シミュレーションの特徴であるトライ&エラーの作業の中でのオブジェクトに対してはもちろん，各種ファイルやデータ等の管理に対してユーザの使い勝手を大きく向上させ，違和感を減少させていることが見い出された．

以上の議論を総合すると，本シミュレーションシステムは自然世界のシミュレーション用のプロトタイプベースのオブジェクトモデルとしては明確によく定義された，ある程度汎用的にシミュレーション世界に適用可能なモデルおよび実装方法であると評価・断定してよいと考える．したがって，本論文の目標であるOBシミュレーションシステムへの再構成は実現されたと結論できる．

### 7.3 本OBオブジェクトシステムの問題点について

#### 7.3.1 情報隠蔽化の不完全性

本オブジェクトシステムにおいては，オブジェクトの情報隠蔽化はシステムとして無条件に保証されているものではなく，本オブジェクトシステムのために実装されて提供している多くのOB関数群やOBメソッド群[13]をその与えられたシンプルなルールどおりに使うという条件下ではじめて保証される性格のシステムである．情報隠蔽化機能の無条件の保証は本来OSがサポートする機能である．

しかし，オブジェクトシステムによる情報隠蔽化のこのような実装上の特性はドメインユーザにとっては（少なくとも現状では）何の障害も生じていない．したがって，事実上の情報隠蔽化機構として十分機能している．実際のシミュレーション駆動には支障がなく，トータルでシミュレーションの目的が達成されるのならば許容範囲内にあると考える．

#### 7.3.2 狭義のオブジェクト指向に基づく実装の併用

すでに6章で実際に利用したように，プロトタイプベースオブジェクトシミュレーションシステム中に狭義のオブジェクト指向のオブジェクトの実装を一部分の要素に併用している．しかしそれらが一部であり，対象世界の全体構造が一貫してOBモデルで記述されるならば許容してよいものと考える．そもそもシミュレーション用のオブジェクトモデルにはその生成の方式は無関係であったという併用の許容を補強すべき理由もある（2章）．もちろんシミュレーションシステム構成に不都合を生じさせるということでもあれば別ではあるが，現状では異種オブジェクトモデルを併用することの実用上の悪影響は見い出されていないので現在のところ問題は出ていない．併用の際に加わる多少の煩雑さは全システムを狭義のオブジェクト指向パラダイムで作成した場合[1]に比べればずっと少なくて済む．

#### 7.3.3 ディレクトリをオブジェクトとしたことにより生じる問題点

この最重要な実装アイデアは，その反面として，OBオブジェクトの構成や個数，サイズ（粒度）に実際上ある程度制限が生じてしまうことや，現在ではともかく将来的には何か詳細・複雑な構成の必要なオブジェクトの実装が困難になる場合を必ずしも否定できない．しかし数十個程度以下のオブジェクトにおさえるモデリングおよび実装上のテクニックや，オブジェクト活性の概念の利用，階層構造オブジェクトの活用でかなりの領域のシミュレーションをカバーできることも確かであろう．ドメインユーザとしては1章の第3段落で述べたオブジェクトシステム構築上の問題点が解決されるメリットの方がはるかに大きいと考える．

### 7.4 従来のシミュレーションシステムとの比較

#### 7.4.1 対象世界を制限したシミュレーションシステムとの比較

シミュレーションを行う対象世界がすでにある程度解かれた問題などであれば，流体分野におけるDEQSOL[16]のような専用システムを用いることが考えられる．このような専用システムは，その適用範囲内ではしごく便利で効率も良く使いやすい．しかし，いずれも手続き型シミュレーションであり，対象世界の「もの」との対応関係の了解性は低い．さらには，本研究の方法論のような汎用性はなく適用可能範囲が狭く，その範囲外での利用はいかに工夫しても困難である．

#### 7.4.2 OODBMSを用いたシミュレーションシステムとの比較

通常のデータベースシステムがデータのみ管理しているのに対して，近年開発されたオブジェクト指向データベース管理システム（OODBMS）[17]では，オブジェクトのメソッドという形でプログラムも管理できる．しかし，少なくとも我々が構築したOODBMSを用いたオブジェクトシミュレーションシステム[1),2),4)]にはいくつかの点で問題がある．まず，記述言語がC++に制限され，OODBMS自体への理解とプログラミング（実装）技術までも要求されるという問題が重い．次に実行性能がデータベースへの入出力やトランザクション処理のために芳しくない．さらにはシミュレーションによく発生する計算モジュールや各モジュール自身の構成の頻繁な変更に向かない．したがって，本研究の提案するシステムに劣ることが分かる．

#### 7.4.3 オブジェクト指向OSとの比較

オブジェクト指向OS[18]は，プログラムやデータを

すべてオブジェクトとして生成・管理・駆動し，プログラムやデータを表すファイルをオブジェクトとして管理することも可能となり，オブジェクトの並行性の管理や，オブジェクトの永続性の管理などが統一的に実現される．従来システムの中では本研究の目標とする機能に最も近い．

しかし，逆に駆動環境の方までもオブジェクト指向で実現されることで，UNIXに馴れたユーザが操作上の違和感を持ちやすく，さらには使用すべきプログラミング言語も新規なものになる可能性が高いとなれば，ドメインユーザにとっては現状環境と非常に差違が大きく，実用性という点で我々の方法論に及ばない．また，オブジェクト指向OSはまだ研究開発段階に近く，一般的に広く用いられていないという欠点もある．

本章の結論としては，ドメインユーザに対する実用性の高いオブジェクトシミュレーションシステムという観点からみて，また，従来コンピューティング資産の継続性，違和感のなさ，容易な使い勝手を兼ね備えているか否か，などの諸点から見て，本方法よりも優れたオブジェクトシステムや環境は我々が感知した範囲内では見つけ出せなかった．

## 8. 結論と今後

### 8.1 OBオブジェクト再構成方法論に対する結論

本研究は，全体ではオブジェクトシミュレーションをUNIXシステム上にシンプルに再構成・実装するためのソフトウェアアーキテクチャの提唱である．その成果としては，まず第1にオブジェクトシミュレーションシステムをUNIX上に再構成して実装できることを実証したことである．次に，その簡潔性，分かりやすさ，使いやすさ，オブジェクトとしての表現記述能力は現在類例を見ないほど良好であることである．著者たち自身および他の複数のユーザたちの2年以上の使用経験においてもそのことは検証された．ドメインユーザとしては実装の簡単さと実用性の高さを最も重要度高く評価したい．したがって，本方法論は他の従来のオブジェクト指向方法論に比較して，いくつかのシステム上の不満足点も持つが，そのユーザ負担に対する有効性・有用性の比率が高い方法論であると結論できる．

### 8.2 応用実装されたOBオブジェクトシミュレーションからの結論

本研究に基づくオブジェクトシステムは，現在のところは十分に対象世界の構造化モデルの表現機能を持ったシステムとして構築できた．それゆえに，実現されたオブジェクトシミュレーションシステムも当初の目標をほぼ達成できた．実装例のオブジェクトシステムは本研究ではシミュレーションに対象を限定したが，実際は汎用的に他の分野のオブジェクトシステムとしても適用・実現でき，従来には見られない特徴を発揮できるとの見解をいくつかの試作システムから得ている．結論としては，従来シミュレーション環境との整合性も良いゆえに実用性の高いオブジェクトシミュレーションシステムが提唱され実際に構築評価され，その高い有用性が検証されたといえよう．

### 8.3 今後の研究展開

本論文で実装したのはオブジェクトシステムに構成できる可能性の1つにすぎず，まだ多くが未検討・未実装である．たとえば，クラスベースのオブジェクトシステムなどもそうである．現在そのいくつかの実装を検討中である．

## 参考文献


1) Hatakeyama, M., Kaneko, I. and Uehara, H.: Numerical Wind Tunnel Simulations for Arbitrarily Complex and/or Moving Test Bodies Based on the Object-Base Architecture and GUI, *Proc. 5th International Symposium on Computational Fluid Dynamics-Sendai*, Vol.1, pp.279–284 (1993).
2) 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).
3) 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, pp.27–32 (1995).
4) 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).
5) 金子　勇，畠山正行：オブジェクトベースモデルに基づくシミュレーション駆動機構の構築，情報処理学会第1回プログラミング研究会研究報告，Vol. 95, No.21, pp.1–8 (1995).
6) 畠山正行編：数値情報環境と実用システム開発，数値環境研究会（1989）．
7) 畠山正行：計算力学ユーザからの要求記述とその情報モデル構築の試み，第39回応用力学連合講演会講演論文集，pp.235–238（1989）．
8) 渡邊正雄：オブジェクト指向に基づく柔軟な連

続流シミュレーションの一貫駆動環境，茨城大学大学院理工学研究科情報工学専攻修士学位論文（1997）.
9) Wegner, P.: Concept and paradigms of Object-Oriented programming, *OOPS MESSENGER*, Vol.1, No.1 (1990).
10) 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).
11) 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).
12) Liskov, B., Bloom, T., Gifford, D., Scheifler, R. and Weihl, W.: Communication in the Mercury System, *Proc. 21st Hawaii Conference* (1988).
13) 金子 勇：オブジェクトベースモデルに基づくシミュレーション環境の構築，茨城大学大学院工学研究科情報工学専攻修士学位論文（1995）.
14) 畠山正行，宮崎大輔，鈴木俊人：エージェントモデルを用いた連続流の差分解に対する解適合格子の自動生成とその適用例，日本数値流体力学会第10回数値流体力学シンポジウム講演論文集，pp.222–223（1996）.
15) Rumbaugh, J., et al.: *Object-Oriented Moddeling and Design*, Prentice Hall (1991).
16) 佐川暢俊，金野千里，梅谷征雄：数値シミュレーション言語 DEQSOL，情報処理学会論文誌，Vol.30, No.1, pp.36–45 (1989).
17) Khoshafian, S.: *OBJECT-ORIENTED DATABASES*, John Wiley & Sons, 野口喜洋，小川東（訳）：オブジェクト指向データベース，*bit* 別冊，共立出版 (1995).
18) Mitchell, J.G., et al.: An Overview of the Spring System, http://www.sun.com/tech/projects/spring/papers/overview.ps.

## 付録 用語説明

**カプセル化** 属性とメソッドの明示的な一体化.

**情報隠蔽** 属性へのアクセス制限.

**広義のオブジェクト指向** 属性とメソッドの明示的な一体化（カプセル化），必要な要素に対する情報隠蔽，メッセージパッシングの概念に基づく外部との通信（相互作用）インタフェース，の3条件を満たす独立的な記述単位からなる体系概念．オブジェクトベースともいう.

**狭義のオブジェクト指向** 広義のオブジェクト指向にクラスと継承を追加条件とするオブジェクト指向．一般的によく知られている方のオブジェクト指向で，クラスをオブジェクト定義の核心概念とするという意味でクラスベースのオブジェクト指向と呼ばれることもある.

**プロトタイプベースオブジェクト指向** 上記の意味でのクラスベースオブジェクト指向に対して，クラスの概念を持たず，任意のオブジェクトをコピーすることにより新しいオブジェクトを生成するオブジェクト指向．コピーの元（ベース）となるオブジェクトをプロトタイプと呼ぶことからプロトタイプベースオブジェクト指向，あるいはコピーベースオブジェクト指向とも呼ばれる．継承類似の委譲という概念を含む.

(平成 9 年 4 月 14 日受付)
(平成 10 年 4 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.

**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.