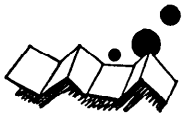
解　説

オブジェクト指向計算の現状と展望†



米　澤　明　憲††

1. はじめに

「オブジェクト指向」における「オブジェクト」という概念は，1960年代後半から1970年代初期にかけて，計算機科学のいくつかの分野で独立にかつほぼ同時期に，異なった形態をとって生まれた[1]．たとえば，

- K. Nygaard らによる離散事象のシミュレーションのための言語 Simula,
- A. Kay の理想のマルチメディア・パーソナル計算機 Dynabook とそのユーザ言語 Smalltalk (Smalltalk 80 の前身),
- C. Hewitt らによる並列計算モデルと知識プログラミングのための ACTOR 形式,
- M. Minsky の人工知能の知識表現のためのフレーム形式,
- J. Dennis らによる OS における資源保護のためのケイパビリティ概念,

などを考案・設計する際，中心的な役割をはたした概念が，今日「オブジェクト」と呼ばれるものに相当する．

このように広い背景と古い歴史をもち，80年代に入り急速に脚光を浴びるようになった「オブジェクト指向」について，以下筆者の私見を混じえつつ，その要点を直観的かつインフォーマルに述べることにする．(より精密な議論は別の機会に譲りたい．) 標題において，「オブジェクト指向プログラミング」とせず，「オブジェクト指向計算」としたのは，「オブジェクト」に基づく考え方が，ただプログラミングスタイルに留まらず，後に触れるように，計算機による問題解決や情報処理に関連する基礎から応用までの広い分野に浸透しつつあるからである．

2. オブジェクト指向とは

なにが「オブジェクト指向」を特徴づけるかは，この考え方に対する力点の置き方や，応用分野の相違によって意見が分かれるが，唯一の共通点は，「オブジェクト」という概念の存在であろう．「オブジェクト」とは，あるデータ（群）とそれを使用するための基本的な操作・手続群をひとまとめにし，その合併体を一つのモジュールとみなしたもの，と考えるのが最も広い解釈である．したがって構成しようとするシステムをオブジェクトの集りとして表現しようという考え方が「オブジェクト指向」の基本となる．

ここでいうシステムとは，単なるプログラムやソフトウェアシステムを考えるのではなく，計算モデル，ソフトウェアの設計，シミュレーションモデル，CAD/CAM における要素モデル，人工知能における知識の表現形式など，広く解釈すべきものである．また，上で，「オブジェクト指向」は考え方だと述べたのは，筆者なりの意図がある．研究者の間では，特にプログラミング言語の専門家の間では，いわゆる「オブジェクト指向言語」を特徴づける言語機能は，なにとなにで，なにがなければ「オブジェクト指向言語」とは言えない……のような，議論が華やかである[2]．このような議論は，新しいソフトウェアの概念を積み上げてゆく上できわめて重要であるが，一般のソフトウェアの構築してゆく上で，“これこれこのような機能をもった「オブジェクト指向言語」でプログラムを書かなければ，「オブジェクト指向」プログラミングをしたことにならない”というような傾向が広がることは，好ましくないことだと筆者は考えている．

3. オブジェクト指向の目差すところ

システムを構成する場合，従来の考え方では，その要素となるモジュールとして，「データ」と「手続」という2種類のものを分離して考える傾向が強かった．これに対して，オブジェクト指向では，**2.** で述

† Perspectives and Projections of Object-Oriented Computing by Akinori YONEZAWA (Dept of Information Science, Tokyo Institute of Technology).

†† 東京工業大学理学部情報科学科

べたように,「データ」と「手続」を分離せず,データとそれを操作する基本的な手続群を一体のものとみなす.

では,なぜこのような考えがでてきたのであろうか? それは計算機ソフトウェアの構築一般に対する二つの大きな課題をいかに克服するかに関係する.一つは「複雑さの壁をいかに乗り越えるか」,もう一つは「対象世界のモデルをいかに自然に計算機の中に表現するか」[3] という問題である.そしてこの二つの問題は,当然のことながら,互いに深い関係にある.

複雑さを克服する手段として,オブジェクト指向においては,(1)密接に関連するデータと手続はひとまとまりの個体として定義し,かつ一度個体として定義したものは,(2)それがどのように実現されているかを知らずともその個体の機能を使うことができる,というオブジェクトの性質を利用する.(1),(2)はソフトウェア工学,プログラミング方法論における,抽象化,抽象データ型,情報隠蔽などの概念のもつ性質に対応する.また,関連する機能や知識およびその知識の操作手続を一つのまとまったモジュールとして定義するために使われるフレーム形式も(1),(2)の性質をもつ.

計算機によって離散事象をシミュレートするための高級言語 Simula が,オブジェクト指向の発端の一つとなったことは **1.** で触れた.シミュレーションは,対象世界の挙動に関し,必要な情報を得るのに十分な精度で,対象世界のモデルを計算機に表現することから始められる.その際,対象世界や必要とする情報が複雑になればなるほど,対象世界の構造を自然な形でモデル化することがますます必要となる.このためにSimula ではオブジェクトという概念が考案されたわけである.

一方,人工知能の基本問題の一つは,われわれ人間がもつ世界に対する諸知識をいかに使い良い形で効率的に計算機の中に表現するかである.これに関して,われわれが世界を知覚・認識する際に無意識に使っているのと同様な区分け方に従って個体や概念を区分け分節し,それらのおのおののもつ機能や性質をそれぞれのまとまりとして表現するのが,最も都合が良さそうだというのが,フレーム形式の提唱理由である.これも,結局は世界に対する自然なモデル化という要請が基本となっている.

さらに,この数年データベースにおける概念モデル(データモデル)の分野においても,オブジェクト指向の諸概念を取り込む研究が活発になっている[4,5].これは,従来の関係データベースの枠組では,実世界のモデル化が不自然にならざるを得ないという点が認識されはじめたからであろう.

以上から,「オブジェクト指向」の目差すところは,次のように要約される.計算機による問題解決や情報処理を試みる対象領域を,「オブジェクト」という概念を用いてできるだけ自然な形で計算機内にモデル化し[3],その際に獲得される「自然さ」によって,ソフトウェアシステムの複雑さの障壁を乗り越え,より高度な問題解決・情報処理の能力を手にすること.

ここでいう「自然さ」とは,システムの設計者が対象世界に対してもつイメージや,設計者が頭の中に作り上げるモデルに登場してくる物理的なものや概念的なものと,システムを構成するオブジェクト群との間に1対1に近い対応関係がとれることを意味する.

4. オブジェクト指向の具体的な要素

オブジェクト指向計算を特徴付ける要素をもう少し具体的に述べてみる.しかし,以下の要素がすべて備わっていなければ「オブジェクト指向」とはいえないという狭い観点は推奨できナイことをあらかじめ断わっておく.同様に,各要素についても,以下で説明するような性質や機能をそれぞれすべてもっていなければならないというわけではない.さまざまな変形がある.

オブジェクト メッセージを受理すると活性化し,メッセージによって指定された仕事をする自立的な手続的モジュール (procedural entity, 算体と訳されることもある).オブジェクトは,内部記憶をもつことが一般的で,記憶の内容がオブジェクトの状態に対応する.よって記憶内容の変化にともなって状態も遷移する.

より具体的にいえば,オブジェクトは,**2.** で触れたように,データとそれを操作する手続群の合併体である.しかし,上での「受理」「自立的」などの用語から分かるように,送られてきたメッセージを受け取り指定された仕事を開始するか否かは,オブジェクト自身の判断,すなわち現在のオブジェクトの状態とオブジェクト自身に備わっている機能(あるいは定義されている手続群)によって決定される.また,オブジェクトの記憶内容の参照・更新は,オブジェクト自身が(自分に定義された手続を用いて)行うだけで,ほかのものが勝手に外からアクセスすることはできない.

あるオブジェクトの具体的な挙動を定義するには,

そのオブジェクトがどのようなメッセージをどのような状態のとき受理し，そのときどのような手続が起動されるか指定してやればよい．同様に，オブジェクトが内部状態をもつ場合，状態がどのような要素で構成されるか指定してやればよい．「オブジェクト」概念の使われる目的にしたがって，その挙動を記述する言語も，いろいろなレベルや種類がある．ある種の代数による数学的記述，ソフトウェア設計用言語による記述，計算機による実行されるプログラミング言語による記述などである．

オブジェクトの動作　活性化したオブジェクトは，一般にどのような動作の実行が可能であるか？まずオブジェクトにメッセージを送ること，これはほかのオブジェクトとの唯一の相互作用の手段として必要である．オブジェクト間のメッセージのやりとりの方式は，複数のオブジェクトが同時に活性化し得る場合は種々のものが考えられる[7]．また，オブジェクトが新たなオブジェクトを生成できる機能は，動的に構造が変化してゆく対象世界をオブジェクトの集まりとしてモデル化する上で重要であると考えられている．

条件分岐，繰り返しなどの通常の命令型プログラムの制御構造や，算術演算・リスト処理の基本演算などがオブジェクトの基本的機能としてあらかじめ備わっていると通常は仮定する．しかし，これらの制御構造や基本演算は，はじめに何種類かの基本オブジェクトの存在を仮定すれば，それらのオブジェクト間のメッセージのやりとりと，オブジェクトの動的生成の組み合わせによってすべて構成できる．

メッセージ　オブジェクトを活性化し，依頼する仕事を指定する標識 (tag) を含んでいるのが，メッセージである．メッセージは，このほかに依頼する仕事を遂行する上で必要な情報，たとえば演算のための引数なども含むことができる．特に，オブジェクトの名前をメッセージに含めて送ることができる．

クラスとインスタンス　一般に同じような性質や機能をもったものの集まりをクラスと呼ぶが，オブジェクト指向においても，同じ内部記憶の構造と同じ手続群をもつオブジェクトは，同じクラスに属すると考えられる．一つのクラスを指定したとき，そのクラスに属するオブジェクトをそのクラスのインスタンスと呼ぶことが多い．オブジェクトが新たにオブジェクトを生成できるという機能を仮定すると，あるクラスに属するインスタンスを生成する機能をもつオブジェクトをそのクラスに対応する（あるいは，そのクラスをモデル化する）オブジェクトを考えることが多い．

インヘリタンス(継承)　一般に，より上位の概念がもつ性質や機能は，より下位の概念によって受け継がれる．オブジェクト指向においても，下位のクラスに属するオブジェクトは，上位のクラスを特徴づける性質や機能を有するのは当然である．そこで，下位のクラスのオブジェクトを定義するとき，その上位のクラスの名前を指定することにより，その上位のクラス（のオブジェクト）の性質や機能を自動的に受け継ぐ(inherit)ことにすれば，あとはその上位のクラスのオブジェクトはもたないが下位のクラスのオブジェクトを定義するのに必要な機能や性質を付加的に記述するだけでよい．このようなオブジェクト（のクラス）の定義法を許す機構をインヘリタンスあるいは継承と呼ぶ．また互に上下関係のない複数のクラスを，あるクラスの上位のクラスとして指定する機構を多重継承と呼ぶ．このような機構は，対象世界に登場する概念間の上下関係を，オブジェクトのクラス間の関係としてモデル化しそれを簡潔に記述するのに有効である．また，継承機構はモジュールの再利用という観点からも重要である．

5. オブジェクト指向における論点

オブジェクト指向に関する研究や実践は現在いろいろな角度から試みられ，そのうちのいくつかは，本特集の解説記事として紹介されている．しかし今回，種種の事情で取り上げられなかったものあるいは，重要性のわりに扱いが軽かったものも多い．オブジェクト指向ソフトウェア設計法，オブジェクト指向データベース，並列オブジェクト指向計算，プログラミング教育，ロボティクス分野などでの応用などがそれである．以下，これらに簡単に触れるとともに，その他オブジェクト指向計算に係わるいくつかの論点をあげておく．

（オブジェクト指向設計法）　M. Jackson の JSD[8] なども考え方として，オブジェクト指向に基づいた設計法である．ADA をベースに考えた設計法や設計の実例なども報告されている[9]．これらは，インヘリタンスを考慮していないが，それを含めた種々のいわゆる「設計法」が今後多数提唱されると思われる．

（オブジェクト指向データベース）　柔軟性や表現力の拡大を狙って，データモデルの設計のために，オブジェクトやクラス/インスタンスの考え方が，導入されつつある[10]．このほか，オブジェクト指向データ

ベースへのアクセス言語としてどのようなものを考えるかなどの研究が始まっている[11].

(並列オブジェクト指向計算)[14] 並列計算・分散処理における複雑さの壁の克服および，対象世界自身のもつ並列性を素直にモデル化し表現するために，複数のオブジェクトが同時に活性化することを仮定した，研究が行われている．計算モデル，記述言語，アーキテクチャ，プログラミング環境，応用などさまざまのレベルで議論が活発で，国際ワークショップも筆者らで計画されている☆.

(強い型付けと型なし) Lisp, Actor, Smalltalk の流れをくむオブジェクト指向言語では，一般にプログラムの中にあらわれる名前などが型付けされていない．一方，Simula, CLU 系統の言語は強い型付けがされている．型付けがある方が良いか否かは，専門家の間でも意見が分かれる．しかし一般的にいえることは，強固な設計のもとに多人数で大きなソフトウェアを組み上げるときには，型付けされた言語が安全だろう．一方，プロトタイプとして柔軟にソフトウェアを作ってゆく上では，型による強制は障害となる．本特集の近山氏の報告は OS の開発に型のないオブジェクト指向言語を用いた例として大変興味深い.

(実行速度) "オブジェクト指向言語で書かれたプログラムは実行速度に問題がある" というような懸念をときおり耳にするが，これはあまり心配がない．現在オブジェクト指向言語と呼ばれるものが多数あるが，実行速度において通常の命令型言語と変わりがないものもあれば，数倍程度遅いものもある．遅いものについては，改善する研究が本格的に行われている．(昔，実行速度が遅いといわれた Lisp 系の言語の実行速度の飛躍的な向上の歴史を思い起こしていただきたい.) また，オブジェクト指向の考え方を取ることによって得られるソフトウェアシステムの設計レベルでの質の向上は，現時点での速度上の問題を上廻るものと考えられる．さらに，一度作成されたシステムの速度を改良することは比較的容易である．(同じく近山氏の報告を参照されたい.)

(制約プログラミング) 解決したい問題やその解法を記述する場合，登場するもの同士の間に成立しなければならない制約条件を用いると便利な場合がある．このような場合，オブジェクト指向の考え方だけで表現することは不自然なことがある．これに対して，制約条件をオブジェクトとして記述する方法など，制約の表現とオブジェクト指向の考え方をどのようにうまく融合するかという研究が活発になっている[12].

(オブジェクト指向に関する出版物) 本稿では，論文・記事などの引用は最小限にとどめたので，比較的入手しやすい論文集・会議録などをまとめて掲げておく．参考文献の 13)～20) を参照されたい.

6. おわりに

60年代末から70年代にかけて「構造化/的プログラミング」というある種のスローガンのもとに，理論的研究から，新しい言語の設計…ソフトウェア開発の組織・経営論に至るまでさまざまな研究や実践が試みられた．そしてこの過程で，短いプログラム作りを楽しむ人から，大規模なソフトウェア開発に従事する人々までの広範な「プログラマたち」に一致して定着してきた意識は，「個々の手続やルーチンは，使っているアルゴリズムやその表現が明確になるようにわかりやすく書かねばならない」ということであろう.

オブジェクト指向は，直観的なモデル化，あるいはそのモデルの直接的な表現に適した枠組である．このオブジェクト指向という一つのスローガンのもとに，80年代および90年代前半，さまざまな研究と実践が行われ，「複雑さの克服とそれによる問題解決/情報処理能力の高度化のためには，対象世界の自然なモデル化とその直接的な表現が必須である」という意識が，ある規模以上のシステムの設計・実現に関与する人々の間に浸透するであろう.

今後，計算機ハードウェア/アーキテクチャの進歩にともない，莫大な計算力と記憶量が安価に供給され得ることが予想される．このとき，計算機システムは，それぞれの応用分野での「対象世界」の広義のシミュレーション装置あるいは仮想対象世界を与えるものとして活用することが可能になる．そして，この装置を動かすソフトウェアは対象世界のモデルを表現したものである．この観点に立つと，オブジェクト指向および並列オブジェクト指向というものの考え方・見方の重要性が再認識されるであろう.

☆ 1988年9月米国 San Diego にて，OOPSLA '88 の直前に開催される予定である.

参考文献

1) Yonezawa, A. and Tokoro, M.: Object-Oriented Concurrent Programming: An Introduction, in 14) pp. 1-7 (1987).
2) Wegner, P.: Dimensions of Object-Based Language Design, in 17) pp. 168-182 (1987).

3) 米澤明憲：オブジェクト指向型プログラミングについて，コンピュータソフトウェア，Vol. 1, No. 1, pp. 29-41 (1984).
4) 増永良文：マルチメディアデータベース総論，情報処理，Vol. 28, No. 6, pp. 671-682 (1987).
5) 小島 功，植村俊亮：マルチメディアデータベースのためのデータモデリング，ibid., pp. 683-693 (1987).
6) Stefik, M. and Bobrow, D.: Object-Oriented Programming—Themes and Variations, AI Magazine, Vol. No. (1985).
7) 米澤明憲他：オブジェクト指向に基づく並列情報処理モデル ABCM/1 とその記述言語 ABCL/1，コンピュータソフトウェア，Vol. 3, No. 3 (1986).
8) Jackson, M.: System Development, Prentice Hall (1983).
9) Booch, G.: Object-Oriented Development, IEEE Trans. on SE, Vol. SE-12, No. 2 (1986).
10) Beech, D.: Groundwork for an Object Database Model in 15).
11) Bloom, T. and Zdonik, S.: Issues in the Design of Object-Oriented Database Programming Languages, in 18).
12) Boring, A.: Constraints Hierarchy, in 18).
13) 鈴木則久（編）：オブジェクト指向，共立出版 (1985).
14) Yonezawa, A and Tokoro, M (Eds.): Object-Oriented Concurrent Programming, The MIT Press (1987).
15) Shriver, B and Wegner, P.: Research Directions in Object-Oriented Programming, The MIT Press (1987).
16) Cox, B. J.: Object-Oriented Programming—An Evolutionary Approach, Addison Wesley (1986).
17) Proc. 1986 ACM Conf. on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '86), Special Issue of SIGPLAN Notices, Vol. 21, No. 11 (1986).
18) Proc. OOPSLA '87, Special Issue of SIGPLAN Notices, Vol. 22, No. 12 (1987).
19) Bézivin, J. et al.: (Eds.): ECO OP'87-European Conference on Object-Oriented Programming, Lecture Notes in Computer Science, No. 276 Springer-Verlag (1987).
20) Peterson, G. E. (Ed.): Tutorial: Object-Oriented Computing, Volumes 1 & 2 IEEE Computer Society Order Numbers 821, 822 (IEEE Catalog EH 0257-6, EH 0264-2) (1987).

（昭和63年2月8日受付）