第 8 章

オブジェクト指向プログラミング

8.1 オブジェクト指向概念

> *実世界のオブジェクト（対象）をそのまま*
> *プログラムに対応付けるとわかりやすい。*

最近，オブジェクト指向概念が種々のソフトウェア技術に応用され始めている。自立性の高いオブジェクトを組み合わせてプログラムを作ることにより，開発者にとっては高生産性，保守者にとっては拡張性と移行性の確保，利用者にとっては使い勝手の良さという効果をもたらしている。

本章では，オブジェクト指向概念を「使いやすいソフトウェア」と「作りやすいソフトウェア」を促すソフト生産技術の観点から位置付け，その基本技術と応用について述べる。

8.1.1 オブジェクト指向技術の発展の歴史

オブジェクト指向概念とその関連技術の発展は，概ね次の 3 期に分けて考えることができる。

(1) 基本概念創生期（1960 年代後半から 1970 年代半ば）

1960 年代後半にオブジェクト指向概念の重要な特徴であるデータ抽象化，ク

ラスからのインスタンス生成，クラスの階層と継承，などの機能を持ったシミュレーション言語Simulaがノルウェー計算センタで開発された。1970年頃，MITのC. Hewittが，メッセージパッシングを主体とした計算モデルであるActorモデルを提案した。1974年には，MITのB. Liskovが本格的な抽象データ型機能を持つ言語CLUを開発した。このうち，オブジェクト指向概念と関連の深い手続き型パラダイムの情報隠蔽やデータ抽象化技法についてはすでに5章で詳しく述べた。

(2) 言語開発期（1970年代後半から1980年代半ば）

オブジェクト指向言語の代表例としてXerox社のSmalltalkがある。1970年頃から人間の創造的活動をコンピュータが支援する方法を研究する過程で開発され，1980年にはSmalltalk-80として実用化された。その後，コンピュータのパーソナル化と相まって，オブジェクト指向への関心が急速に高まり，本格的なオブジェクト指向言語の研究開発が活発になった。そのほとんどは，既存の言語をベースにオブジェクト指向機能を導入したハイブリッド型またはマルチパラダイム型と呼ばれる言語である。その例として，Prologをベースとした ESP（ICOT），S-LONLI（日立），Lispをベースとした CLOS（ANSI標準案），Tao(NTT)，ABCL(東工大)，Cをベースにした C++(ATT)，Objective-C（Stepstone社）などがある。

一方，知識工学の分野でも，知識表現にオブジェクト指向概念を取り入れたエキスパートシステム構築ツールが開発された。

(3) 応用拡大期（1980年代後半以降）

1990年前後からオブジェクト指向概念の応用技術分野が急速に拡大している。その代表的なものとして，ソフトウェアの使い勝手の良さを決めるグラフィカルユーザインタフェースとその構築ツール，マルチメディアデータを扱うためのデータベース，システムの分散化に対応するオペレーティングシステムやネットワーク，などの分野がある。さらに，それらの開発支援のための設計技法や部品化再利用などのソフトウェア生産技術，あるいは，アプリケーションソフトウェアの共通プラットフォームとしてのオブジェクト管理システムの開発などが積極的に行われており，オブジェクト指向ベースのソフトウェア開

発環境が充実し始めている。

8.1.2　4つの主要概念

コンピュータで取り扱う問題に関連した概念的対象物をそのままプログラムの基本単位として表現できるオブジェクト指向概念は，以下のような4つの基本的な特徴（図8.1）を有し，情報処理や知識処理関連の多くの技術分野で広く利用されている。

① 分散協調型計算モデル

複数個の自律的機能を持つオブジェクトがお互いにメッセージをやり取りしながら協調して問題解決にあたる。

② データ抽象化機能

個々のオブジェクトはデータと手続き（メソッド）を一体化（カプセル化）した自律的機能モジュールとして定義され，外部からのメッセージを受け取ると手続きが起動される。データに対しては外部から直接アクセスはできない。

③ クラスからのインスタンス生成機能

データの値が未定のオブジェクトを型（クラス）として定義し，機能が同じでデータの値だけが異なる複数個のオブジェクト（インスタンス）を効率良く生成できる。

④ クラスの階層化と継承機能

上位クラスの持つ手続きを下位クラスの手続きとして継承できるようなクラス階層構造により，少しづつ機能の異なるオブジェクトを効率良く作成できる。

以下では，これらの特徴について具体的に説明する。

(1) 分散協調型計算モデル

オブジェクト指向概念の最も本質的な特徴である分散協調型計算モデルを「複数個の自律的機能を持つオブジェクトがお互いにメッセージをやり取りしながら協調して問題解決にあたる」計算モデルとして定義した。これを例を用いて説明する。今，会社や大学などの組織内での会議開催の事務処理について考えてみる。図8.2に示すように，以下の4種類のオブジェクトを導入し，互い

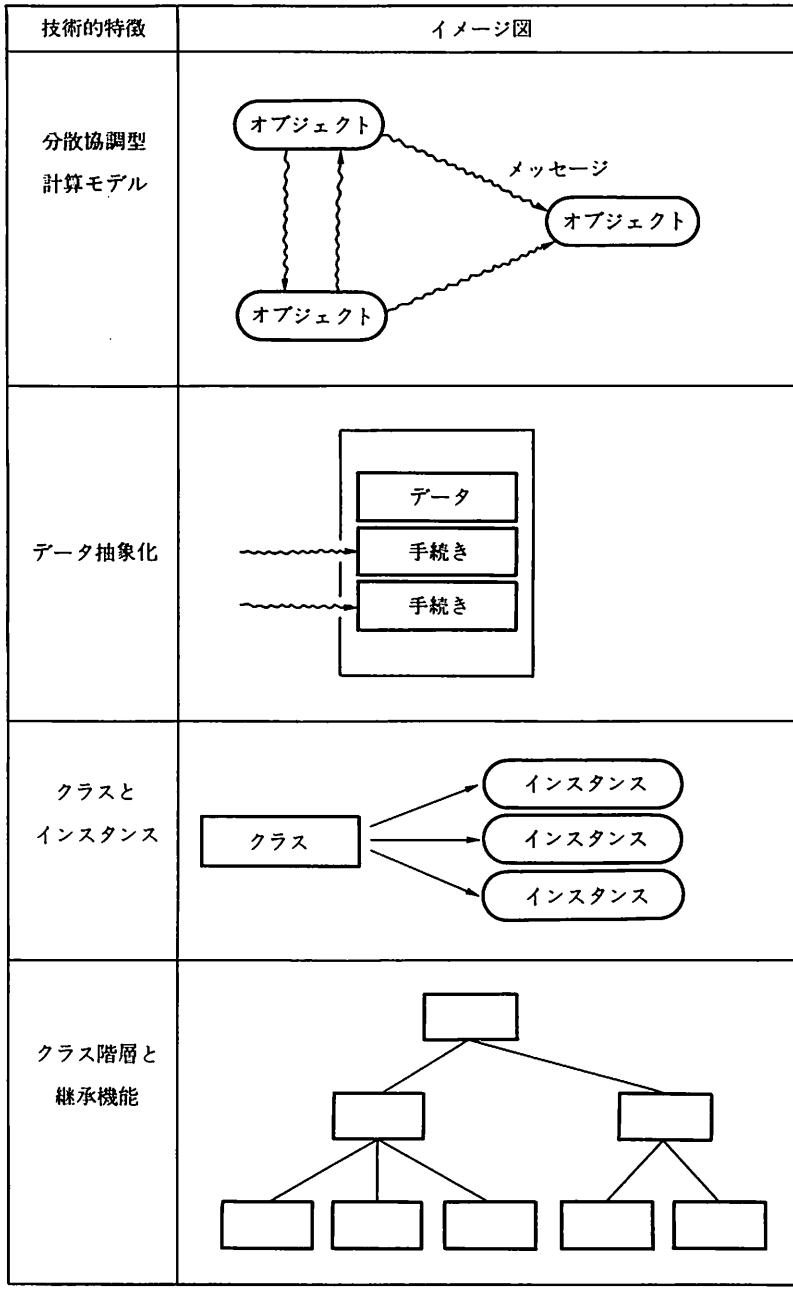


図8.1 オブジェクト指向の4つの基本的概念

にメッセージをやり取りしながら協調して問題解決にあたるようにモデル化できる。

① 事務局オブジェクト

会議開催の要求メッセージを受けると，会議室の予約，OHP の予約，その会議のメンバへの会議開催案内などのためのメッセージを各々の担当オブジェクトへ送り，その後，メンバからの出欠通知のメッセージを受け取り，管理する。

② 会議室予約管理オブジェクト

会議室予約の要求メッセージを受け取り，予約管理をする。

③ 備品予約管理オブジェクト

備品予約の要求メッセージを受け取り，予約管理をする。貸出期限をすぎ

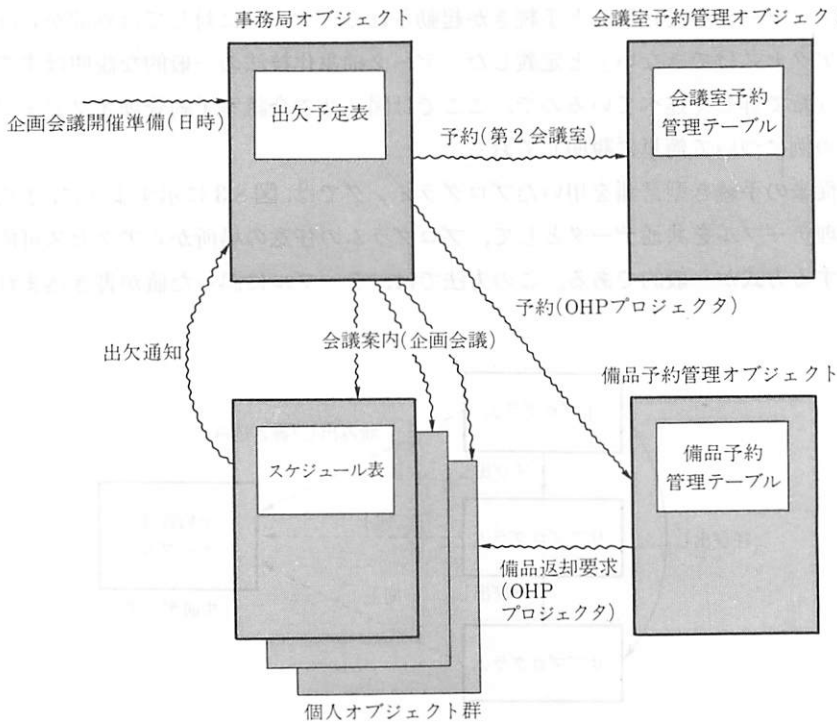


図8.2　分散協調型計算モデルの例（会議開催事務処理）

て未返却の備品への貸出要求があれば，貸出先へ返却要求メッセージを送る。

④ 個人オブジェクト

会議開催案内のメッセージを受け取り，出欠通知のメッセージを送る。

このようにオブジェクト指向プログラミングでは，実世界の問題解決方法に近い計算モデルを用いて自然な表現が可能である。特に，タスクやモジュールをプログラム構成要素としてモデル化をする従来方式のように，プログラム設計の最初の段階から処理制御の順序や共通データの導入を考える必要がないので，「どのように」(how) ではなく，「何を」(what) 主体の表現ができる。

(2) データ抽象化

データ抽象化機能を先に「個々のオブジェクトはデータと手続き (メソッド) を一体化 (カプセル化) した自律的機能モジュールとして定義され，外部からのメッセージを受け取ると手続きが起動される。データに対しては外部から直接アクセスはできない」と定義した。データ抽象化技法の一般的な説明はすでに5章で詳しく述べているので，ここでは図8.2の会議室予約管理オブジェクトの例について簡単に説明しておく。

従来の手続き型言語を用いたプログラミングでは，図8.3に示すように，予約管理テーブルを共通データとして，プログラムの任意の場所からアクセス可能とする方式が一般的である。この方法では，テーブルに誤った値が書き込まれ

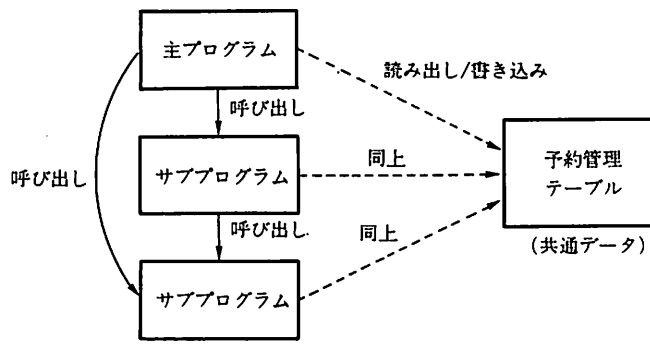


図8.3 共通データアクセス方式を用いた従来の手続き型処理モデル

る危険が高いとか，テーブルのデータ構造を変更したときにテーブルへアクセスしている部分をすべて修正しなければいけないなど，プログラムの信頼性と生産性の問題があった。

これに対し，オブジェクト指向プログラミングでは，図8.4に示すように，データとその手続き群を一体化したオブジェクトとして定義する。これらの手続きはメソッドと呼ばれ，外部から見たオブジェクトの機能は，このメソッドの集合で表現される。データそのものは外部からは隠蔽され，直接アクセスすることはできない。図8.2の例では，会議室予約管理オブジェクトの会議室予約管理テーブルの他に，備品予約管理オブジェクトの備品予約管理テーブル，事務局オブジェクトの出欠予定表，個人オブジェクトのスケジュール表などが外部から隠蔽されている。このようなオブジェクトは独立性が強く，相互作用は相手オブジェクトのメソッド起動のためのメッセージを送信することにより行う。

このようなプログラミング方式では，オブジェクトを利用する側はオブジェクト内のデータの詳細な構造やデータアクセスの詳細な手順を知る必要がないので，プログラムの信頼性と保守性が向上する。また，オブジェクトを作成する側は，オブジェクト内のデータの変更やメソッドの処理手順の変更が生じても，メソッドの機能が不変ならば，まわりのオブジェクトは変更しなくてすむため，プログラムの保守性が高くなる。

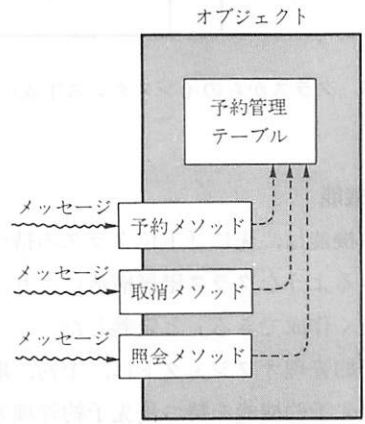


図8.4　データ抽象化機能の例（予約管理オブジェクト）

(3) インスタンス生成

クラスからのインスタンス生成機能は，先に「データの値が未定のオブジェクトを型（クラス）として定義し，機能が同じでデータの値だけが異なる複数個のオブジェクト（インスタンス）を効率良く生成できる」と定義した。

図8.2では，会議室予約管理オブジェクトと備品予約管理オブジェクトがこの例にあたる。両者は，予約管理テーブルの内容が異なる以外は図8.4のような同じ構造のオブジェクトと考えてよい。このように，メソッドが同じで，データの内容だけが異なるオブジェクトを効率良く作成するためには，それらを個々に作成する代わりに，その型となるオブジェクトを1つだけ作成し，それを複製することにより必要な個数分だけ生成するのがよい。この型となるものをクラスオブジェクト，生成されたものをインスタンスオブジェクトと呼ぶ。その様子を図8.5に示す。

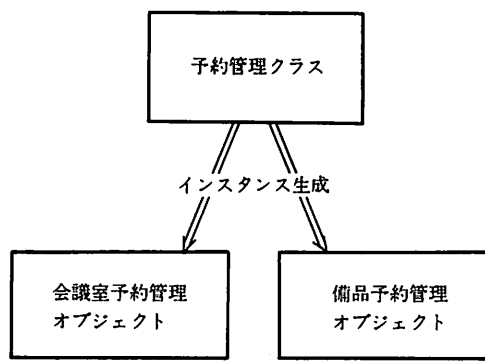


図8.5 クラスからのインスタンス生成の例

(4) クラス階層と継承機能

クラスの階層化と継承機能は，先に「上位クラスの持つ手続きを下位クラスの手続きとして継承できるようなクラス階層構造により，少しずつ機能の異なるオブジェクトを効率良く作成できる」と定義した。

たとえば，図8.4の予約管理オブジェクトは，予約，取消，照会のメソッドを持つが，今，新たに優先予約機能を持つ優先予約管理オブジェクトを導入する場合を考えよう。3つのメソッドを持つ予約管理オブジェクトとは別に，4

つのメソッドを持つ優先予約管理オブジェクトを定義してもよいが面倒である。そこで，その代わりに，図8.6に示すように，予約管理オブジェクトの下にその子供のクラスとして優先予約管理オブジェクトを位置付け，親クラスが持っていない優先予約メソッドだけを定義する。そして，他の3つの予約，取消，照会メソッドは親クラスから継承する。図8.6のもう1つの例の取消待ち予約管理オブジェクトについても同様である。予約管理オブジェクトの下にその子供のクラスとして取消待ち予約管理オブジェクトを位置付け，親クラスが持っていない取消待ち予約メソッドだけを定義すればよい。

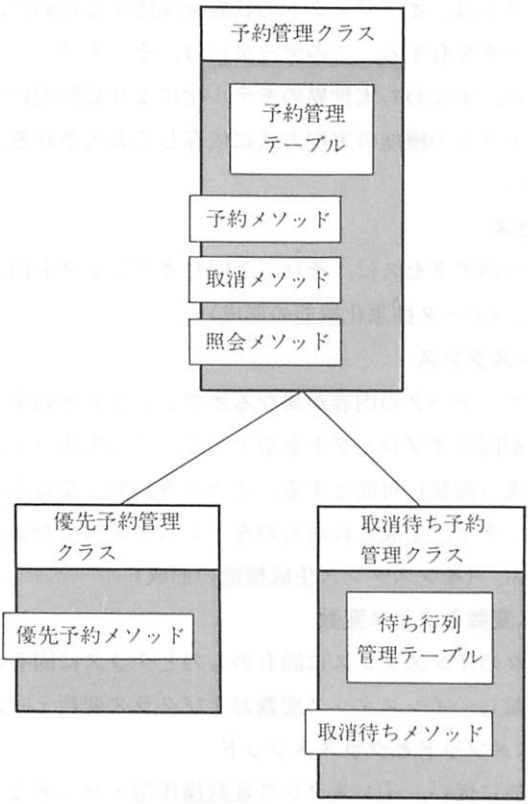


図8.6　継承機能を持つクラス階層の例

8.1.3 発生学的定義

オブジェクト指向概念を用いて実際にプログラムを作成するにはそれを表現する言語が必要である。そこで，オブジェクト指向概念を構成する基本要件として前項で述べた4つの機能をより厳密に規定するために，オブジェクト指向言語モデルの形態形成過程による発生学的定義を以下に示す。

(1) メソッドとメッセージ送信

個々のオブジェクトの機能（外部仕様）は，メソッドの集合とする。各メソッドはそれを指定したメッセージの受信によって起動され，所定の機能を果たす。その際，必要ならば入出力パラメータを有する。(分散協調型計算モデルの形成)

(2) データ

個々のオブジェクトは，オブジェクトの状態を保持するために，必要ならば変数宣言されたデータを有する。このデータには，そのオブジェクトの機能を象徴するようなもの，すなわち実世界のモデル化により必然的に導入されるものと，そのオブジェクトの機能の実現方式に依存して導入されるものがあるが，ここでは区別しない。

(3) データアクセス

これらのデータへのアクセスは，それらと同じオブジェクト内のメソッドからのみ可能とする。(データ抽象化機能の形成)

(4) クラスとインスタンス

メソッドが同じで，データの内容が異なるオブジェクトを効率よく作成するために，メソッドが同じオブジェクトを型として，その実体（インスタンス）を動的に複数個生成（複製）可能とする。このとき，型となるものをクラスまたはクラスオブジェクト，生成されたものをインスタンスまたはインスタンスオブジェクトと呼ぶ。(インスタンス生成機能の形成)

(5) インスタンス変数とクラス変数

データには，個々のインスタンスに固有のものとクラスに固有のものがあるので，それらを分離し，インスタンス変数およびクラス変数と呼ぶ。

(6) インスタンスメソッドとクラスメソッド

データ抽象化機能に従い，インスタンス変数操作用メソッドとクラス変数操作用メソッドを分離し，それぞれをインスタンスメソッドおよびクラスメソッ

ドと呼ぶ。

(7)　クラス階層と継承機能

最後に，メソッドの集合が少しずつ異なるオブジェクトを効率よく作成するために，クラス間に階層構造を導入する。そして，下位のクラスは上位のクラスの性質，すなわちメソッドと関連データの宣言を継承できるものとする。(クラス階層と継承機能の形成)

このような言語モデルの形成過程において，前のものほどオブジェクト指向概念として本質的な機能と考えられる。ただし，このような機能に基づくオブジェクト指向概念の多くの特長，利点はユーザ視点で決められるものである。したがって，ある特長に注目したときに，それを実現するために必要な機能が備わっていれば，すべての機能を適用していなくても「オブジェクト指向」という用語が用いられることがある。

8.2　ソフトウェア生産技術への適用

このようなオブジェクト指向概念の特徴は，ソフトウェア生産技術の観点から以下のような利点をもつ。

8.2.1　アプリケーションソフトウェアアーキテクチャ

昔，ハードウェアが非常に高価であった時代には，「1つのハードウェアの上で沢山のソフトウェアが動く」ことが重要だった。しかし，ハードウェアが急激に低廉化する一方で，アプリケーションソフトウェアの開発コストが膨大になっている現在では，「1つのアプリケーションソフトウェアが沢山のハードウェアの上で動く」ことが重要になってきた。そのために，最近では，アプリケーションソフトウェアのアーキテクチャをできるだけ標準的に作る努力がなされている。たとえば，3.4節 (図3.1) でも述べたように，アプリケーションソフトウェアの稼動環境を含めたソフトウェア構成を以下のようにすることが多くなっている。

①階層化：ハードウェア，基本ソフトウェア，ミドルソフトウェア，アプ

リケーションソフトウェアを完全な階層構造にする。

②部品化：各階層を部品の集合で構成し，統一的なインタフェースとする。

このようなソフトウェア構成に適した技術として，オブジェクト指向概念が注目され，実際に適用され始めている。具体例は8.4節で述べる。

8.2.2 開発者のための部品化，再利用

ソフトウェアの生産性向上の切り札として部品化，再利用技術は古くから研究されてきたが，これまで必ずしも実用技術として成功したとは言えない。その原因としては，従来のプログラミング方法に基づく手続き的モジュールやサブルーチンでは部品としての機能の独立性が弱く，汎用部品ライブラリの構築が容易ではなかったことが考えられる。

これに対し，オブジェクト指向概念は次のような点で部品化，再利用に適している。

①データ抽象化機能により，独立性の強い，汎用的な機能部品を作りやすい。

②階層化機能と継承機能により，部品ライブラリを構築しやすい。

③インスタンス生成機能により，部品のカストマイズがしやすい。

④分散協調型モデルに基づいてプログラム設計を行うことにより，部品（オブジェクト）の抽出，利用が容易である。

典型的な例としては，Smalltalk-80やObjective-Cが提供するクラスライブラリがある。Smalltalk-80では約5000のメソッドを含む250のクラス，Objective-Cでは約300のメソッドを含む20のクラスが定義されている。特にObjective-Cでは，これらのクラスをハードウェアにおけるIC部品のように使うという意味で"ソフトウェアIC"と呼ぶ。

8.2.3 保守者のための拡張性と移行性

情報化社会のインフラストラクチャとして，SISやCIMなどのアプリケーションソフトウェアが複雑化，大規模化しているが，このアプリケーションソフトウェアに関しては，ソフトウェア生産性の低さによる開発コストの増大，開

発要員不足によるバックログの増大などの問題が発生している。そのため，このような高価なアプリケーションソフトウェアをできるだけ長い期間使用するための拡張性とできるだけ広範囲で使用するための移行性が重要な課題である。

オブジェクト指向概念は，以下の特徴により，拡張性と移行性の確保に適している。

①階層化機能と継承機能により，機能追加が容易である。

②データ抽象化機能により，オブジェクトの機能独立性が強く，外部インタフェースと内部構造が完全に分離しているため，機能変更や他機種への移行が容易である。

8.2.4　利用者のための操作性と統一性

パーソナルコンピュータやワークステーションの高機能化と低廉化に伴い，オフィス業務やエンジニアリング業務へのコンピュータ利用が急速に進んでおり，利用者の大半は，従来のような情報処理の専門家ではなくなってきている。このような意味でのコンピュータのパーソナル化に対応して，パーソナルコンピュータやワークステーションのソフトウェアは，誰でも簡単に使えるユーザインタフェースを提供することが必須になっている。

オブジェクト指向概念は，以下のような特徴により，ユーザインタフェースの操作性の容易化と統一性の確保に適している。

①データ抽象化機能とメッセージ駆動型制御方式により，画面上に見えている各々のオブジェクトが持っている幾つかの機能の１つを選ぶという操作，すなわち，オブジェクトにメッセージを送るという単一的操作で簡単にコンピュータと対話が可能である。

②部品化，再利用機能を用いて，種々の応用ソフトウェアのユーザインタフェースを共通部品オブジェクトで構築することにより，異なる応用ソフトウェア間でユーザインタフェースを統一できる。

たとえば，図8.4の予約管理オブジェクトを画面上で選択すると，予約，取消，照会の選択メニューが表示され，その中から照会を選ぶと予約状況が表示される，というようなシステムを簡単に作れる。

8.3 C++

C++は，1983年頃にAT&TのB.Stroustrupによって開発された。当初は，型の機能の強化やデータ抽象化機能の導入などにより，“よりよいC言語”を作ることを目的としていた。数回の言語仕様の改定を経て，現在，C言語ベースのオブジェクト指向言語としてANSI標準化の努力がなされている。

ここでは，8.1.2項でオブジェクト指向概念の説明に用いた図8.2の会議開催事務処理を例題として，C++のプログラム例を示す。このプログラムの名前をOOO（Object-Oriented Office）としておく。

8.3.1 オブジェクトの設計

ここで例として取り上げた会議開催事務処理業務が，これまで実世界では人手作業で行われていたとする。これをコンピュータ化するためには，そのアプリケーションソフトウェアを開発する必要がある。その開発にオブジェクト指向概念を適用するとき，実世界とOOOシステムの対応をとるために，最初に何をオブジェクトとするかが問題となる。その決定手順の例を以下に示す。

(1) 事務局オブジェクトの導入

まず最初に，事務局オブジェクトを導入する。実世界では，事務処理を会議の主催者が自ら行う場合，その担当者を決めて頼む場合，その担当部署が存在し，そこに頼む場合などが考えられる。この事務局オブジェクトを特定の会議専用とするか，複数の会議を扱うかはシステム要求によって決める。実世界において，ある部署が複数の会議開催の事務処理を扱う場合も，事務局オブジェクトを個別に作ることは可能である。

OOOでは，事務局オブジェクトは，企画会議(KK)，予算会議(YK)，品質管理会議（QC）の3つの会議の開催事務を行い，以下の機能（メソッド）をもつとした。

①会議開催準備（会議室予約，備品予約，開催案内通知）
②会議開催取消（会議室予約取消，備品予約取消，開催取消通知）
③出欠管理（出欠変更受付）
④照会（出欠予定状況表示）

(2)　会議室予約管理オブジェクトの導入

次に会議室予約に関して，実世界では，各会議室に予約簿を備えておき，利用者が記入するか，特定の場所に複数の会議室の予約簿をまとめておき，利用者が記入するか，あるいは会議室予約管理者がいて，予約を依頼する等の方法が考えられる。

会議室予約に関し，もし実世界で利用者が勝手に予約簿に記入する方式をとっていた場合，図8.3で述べた従来の手続き型処理モデルに近い。しかし，オブジェクト指向設計やデータ抽象化技法では，この方式は許さない。予約簿はオブジェクト化されることになる。その代わり，メソッドの外部仕様さえ変えなければ，予約簿のデータ構造やメソッドの処理手順は自由に決めたり，変更したりできる。システム化にあたっては，各会議室毎に対応するオブジェクトを作る方法と複数の会議室をまとめて管理するオブジェクトを導入する方法がある。

OOOでは，各会議室毎に対応する管理オブジェクトを作り，以下の機能(メソッド）を有することとした。

①予約
②予約取消
③予約状況の照会

(3)　備品予約管理オブジェクトの導入

次に備品の予約管理に関しても，会議室予約と同様の設計方針とし，各備品毎に対応する管理オブジェクトを作り，以下の機能（メソッド）を有することとした。会議室予約と機能が同じなので，図8.5に示したように両者は同一のクラスから生成されるものとする。

①予約
②予約取消
③予約状況の照会

なお，図8.2の備品返却要求の機能は，管理テーブルの構造と処理が複雑になるので，今回の例題では省略した。

(4) 個人オブジェクトの導入

会議への参加者のオブジェクト化を検討する。一般にはシステム化の対象外とし，システムとの間のインタフェースだけを決めておけばよい。しかし，このような分散協調型モデルは，1つのコンピュータの上で実現することもできるが，分散システムの上で個々のオブジェクトが異なる端末上にあってもよい。たとえば，分散コンピューティングの環境下でグループウェアが用いられ，各人のスケジュール表が外部からアクセス可能な場合は，スケジュール表をオブジェクト化することが考えられる。

OOOでは，このようなグループウェアを想定し，個人毎に対応するスケジュール表管理オブジェクトを作り，以下の機能（メソッド）を有することとした。

①予約（会議開催通知の受付と出欠の返事）

②予約取消（会議取消通知の受付）

③予約状況の照会（スケジュール表の表示）

④優先予約（優先すべき会議の開催通知受付と先約の会議の欠席通知）

(5) データ関連オブジェクト

上記の(1)～(4)は，分散協調モデルの中で自律的に機能するオブジェクトであった。図8.2にも示されるような，これらの自律的オブジェクトが内部に持っているデータも，抽象データ型の観点でオブジェクトとして設計しておくことができる。

OOOでは，会議室予約管理テーブル，備品予約管理テーブル，スケジュール表を同一のデータ構造で実現することにし，それらに共通のクラスオブジェクトを導入した。出欠予定表については，会議の種類毎に別々の表で管理することにし，それらに共通のクラスオブジェクトを導入した。

これらの抽象データ型としてのクラスオブジェクトに関する，実世界との対応については，個々の会議室毎に管理簿を作成したり，さらに各月毎に作成する場合を考えればよい。実世界では，それらの用紙は定形フォーマットを決めておき，必要に応じてそれをコピーして利用する。システム内では，この定形フォーマットがクラスオブジェクトであり，コピーされたものがインスタンスオブジェクトである。

(6)　インスタンス名の管理オブジェクト

実世界上の概念とは対応しないが，システム構築に際して，インスタンスオブジェクトの名前を管理する必要がある。たとえば，同一のクラスから生成された複数のインスタンスのいずれかにデータの表示メッセージを送ったとき，そのデータの値の表示と共にインスタンスの名前を表示しないと，どのインスタンスのデータかがわからない場合がある。一方，その表示用のメソッドはインスタンス間で共通なので，インスタンスの名前はインスタンス対応に管理する必要がある。

OOOでは，インスタンス名の管理のためのクラスオブジェクトを導入し，インスタンスの名前を管理する必要のあるクラスは，このクラスの子クラス（派生クラス）とするようにした。

8.3.2　オブジェクトの実現

以下に実際のプログラム例を示すが，オブジェクト指向プログラミングの説明をわかりやすくするために，処理を簡略化している。また，自明の初期処理などについては，説明を省く。

(1)　日程表クラス（Date）

日程表のクラス定義を図8.7に示す。これは，会議室予約管理テーブル，備品予約管理テーブル，スケジュール表に用いる。クラスDateは，1年分のカレンダに相当するもので，クラスMonthのインスタンスを12個（実際は13個）作成，保持する。クラスMonthは，クラスDayのインスタンスを31個(実際は32個)作成，保持する。クラスDayは，1時間単位に24時間分の文字列記入欄を持つ。なお，配列変数の添字の値と実際の月日の数字を対応させるため，定数宣言のMONTHとDAYは1だけ多くなっている。

Dateは，抽象データ型として定義され，ある特定の月，日，時間の文字列記入欄に対する比較（compare），書き込み（copy），読み出し（read）のデータ操作関数を有する。

```
#define HOUR      24
#define DAY       32
#define MONTH     13

class Day  {
  char id_title [HOUR*10] ;
public:
  char* h [HOUR] ;
  Day();
} ;

class Month  {
public:
  Day d [DAY] ;
} ;

class Date  {
  Month m [MONTH] ;
public:
  int compare(int mm, int dd, int hh, char* title) {
    return strcmp(m [mm] .d [dd] .h [hh] , title);
  }
  void copy(int mm, int dd, int hh, char* title) {
    strcpy(m [mm] .d [dd] .h [hh] , title);
  }
  char* read(int mm, int dd, int hh) {
    return m [mm] .d [dd] .h [hh] ;
  }
} ;
```

図8.7 日程表 (Date) のクラス定義

(2) 会議管理クラス (Meeting)

先に述べたインスタンスの名前を管理するクラス Labeled_object と会議の種類ごとに出欠予定などの管理をするクラス Meeting の定義を 図8.8 に示す。Labeled_object は，インスタンス名の記憶場所を持つ。

Meeting は，3人の個人オブジェクトの出欠管理と会議室や備品の予約の要否を記憶しているが，詳細は後述する。

ここで，C++の基本的な文法規則を説明しておく。例からわかるように，C++では，クラスは以下の構文で定義される。

```
class Labeled_object {
  char idname [20] ;
public:
  char* name;
  Labeled_object() {name = &idname [0] ;}
} ;

class Meeting :public Labeled_object {
  char n [30] ;
public:
  int s [3], e1, e2;
  char* attendant [3] ;
  Meeting(char*, int, int, int, int, int);
  void display();
} ;
```

図8.8　会議管理クラス（Meeting）の定義

```
class クラス名：親クラス名 {
  非公開部
public：
  公開部
} ;
```

クラスの階層は，図8.8のLabeled_objectとMeetingのように，子クラスの定義のときにクラス名の後に親クラス名を書くことによって表す。C++では，子クラスを派生クラス，親クラスを基本クラスという。クラスの中で定義されている変数や関数はクラスのメンバと呼ばれる。クラスの定義本体はpublicというキーワードを境にして非公開部と公開部に分かれる。非公開部で定義されたメンバはそのクラスの中でのみ利用できる。公開部で定義されたメンバは，このクラスの子クラス定義での親クラス名の指定時にpublicと宣言しておけば，子クラスでも利用できる。

関数の定義は，クラスの中ではプロトタイプ宣言だけしておき，本体の定義を後でることもできる。Meetingの中のMeetingとdisplayがその例である。クラスと同じ名前の関数は，コンストラクタと呼ばれ，クラスからインスタンスが生成されるときに実行される初期化関数である。Day，Labeled_

object, Meeting がその例である。

OOO で用いるクラスのうち，Labeled_object を基本クラスとするクラス階層を 図8.9 に示す。この中で，Reservation, Staff, Office が OOO の分散協調システムの中で自律的に動くオブジェクトのクラスになる。それらのプログラムを 図8.10 に示す。

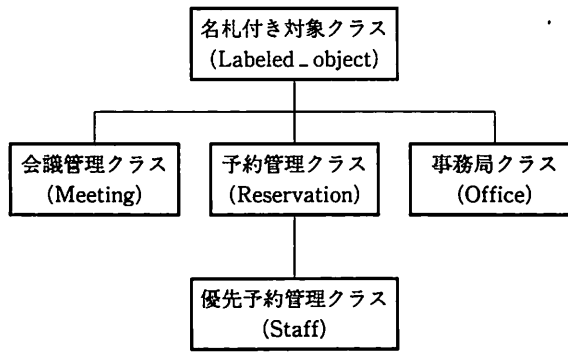


図8.9 クラス階層図

(3) 予約管理クラス (Reservation)

クラス Reservation は，そのインスタンスが会議室予約管理オブジェクトと備品予約管理オブジェクトになるものである。予約管理テーブルは，変数 schedule を Date のインスタンスであると宣言して用いる。図 8.4 で示した例と同じように，メンバ関数として，予約(reserve)，取消(cancel)，照会(display)を用意する。

(4) 優先予約管理クラス (Staff)

クラス Staff は，そのインスタンスが個人オブジェクトになるものである。これは，基本機能はクラス Reservation と同じであるが，さらに優先予約機能が必要なので，Reservation の派生クラスとし，優先予約 (priority) 関数を定義する。Reservation と Staff の関係は図 8.6 に対応する。

優先予約の場合，もし，すでに同じ時間に他の会議の予定が入っていれば，それを取り消す必要がある。その欠席通知メッセージを事務局オブジェクトに

```
class Reservation :public Labeled_object {
 int i;
public:
  Date schedule;
  Reservation(char* id ="no_name")  {strcpy(name, id);}
  int reserve(char*, int, int, int, int);
  void cancel(char*, int, int, int, int);
  void display(int, int);
} ;

class Office;

class Staff : public Reservation {
  Office* officer;
  char* meeting;
  char n [10] ;
  int i, j, flag;
public:
  Staff(char* id ="no_name")  {strcpy(name, id); meeting = &n [0] ;}
  void initialize(Office* ptr) {officer = ptr;}
  void priority(char*, int, int, int, int);
} ;

class Office :public Labeled_object  {
  char list [20] ;
  Reservation* room;
  Reservation* ohp;
  Staff*       xxx [3] ;
  Meeting*     mmm [3] ;
public:
  Office (char* id ="no_name")  {strcpy(name, id);}
  void initialize(Reservation*, Reservation*, Staff*, Staff*, Staff*,
                  Meeting*, Meeting*, Meeting*);
  void arrange(Meeting*, int, int, int, int);
  void cancel(Meeting*, int, int, int, int);
  void attend(char*, int, char*);
  void display( );
} ;
```

図8.10　主要クラス（Reservation, Staff, Office）の定義

送るためには事務局オブジェクトのアドレスを知る必要がある。そこで，Staffの非公開部でこの事務局オブジェクトのアドレスを記憶するために，Officeのインスタンスへのポインタ変数officerを宣言している。そして，その実際のアドレスを教えてもらうために，公開部に初期処理関数initializeを用意している。

(5) 事務局クラス (Office)

クラスOfficeは，そのインスタンスが事務局オブジェクトとなるものである。オブジェクト設計で設定した4つの機能を実現するために，会議開催準備(arrange)，会議開催取消 (cancel)，出欠管理 (attend)，照会 (display) というメンバ関数を用意した。

事務局オブジェクトは会議室予約管理オブジェクト，備品（ここではOHP）管理オブジェクトおよび3人の個人（ここでは安部氏，馬場氏，千葉氏）オブジェクトにメッセージを送る必要があるため，それぞれのクラスのインスタンスへのポインタ変数として，room, ohp, xxx[3]を宣言している。さらに，3種類の会議（企画会議 (KK)，予算会議 (YK)，品質管理会議 (QC)）の管理テーブルが必要なので，Meetingのインスタンスへのポインタ変数として，mmm[3]を宣言している。これらの変数に実際のインスタンスのアドレスを設定するために，初期処理関数initializeを用意している。

以上のクラス定義の中でプロトタイプ宣言だけしているメンバ関数の定義本体については，次の実行過程の説明の中で必要に応じて説明する。

8.3.3 オブジェクトの実行

これまで定義してきたクラスを用いた分散協調システムOOOの実行過程を見てみよう。まず，主プログラムの一例を図8.11に示す。

(1) インスタンス生成

最初の9行の宣言はすべてクラスからのインスタンスの生成である。クラスOfficeから事務局オブジェクトooo，クラスReservationから会議室予約管理オブジェクトroom_mgrとOHP予約管理オブジェクトohp_mgr，クラス

```
enum months {JAN=1,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC} ;
enum compare {SUCCESS,FAIL} ;
enum membership {N,Y,P} ;

main() {
  Office*      ooo       =new Office("ooo");
  Reservation* room_mgr  =new Reservation("room_mgr");
  Reservation* ohp_mgr   =new Reservation("ohp_mgr");
  Staff*       abe       =new Staff("abe");
  Staff*       baba      =new Staff("baba");
  Staff*       chiba     =new Staff("chiba");
  Meeting*     kk        =new Meeting("KK",Y,Y,Y,Y,Y);
  Meeting*     yk        =new Meeting("YK",P,Y,N,Y,N);
  Meeting*     qc        =new Meeting("QC",N,Y,P,N,N);

    ooo->initialize(room_mgr,ohp_mgr,abe,baba,chiba,kk,yk,qc);
    abe->initialize(ooo);
   baba->initialize(ooo);
  chiba->initialize(ooo);

  ooo-> arrange(kk,MAY, 11, 14, 15);
  ooo-> arrange(yk,MAY, 11, 14, 16);
  ooo-> arrange(yk,MAY, 11, 15, 17);
  ooo-> arrange(qc,MAY, 11, 13, 15);
  ooo->  cancel(kk,MAY, 11, 14, 15);
  ooo-> arrange(kk,MAY, 11, 13, 15);

        ooo->display();
  room_mgr->display(MAY, 11);
   ohp_mgr->display(MAY, 11);
        abe->display(MAY, 11);
       baba->display(MAY, 11);
      chiba->display(MAY, 11);
}
```

図8.11　OOOシステムの主プログラム

Staffから3人の個人オブジェクトabe, baba, chiba, クラスMeetingから3種類の会議（KK, YK, QC）の管理オブジェクトkk, yk, qcが生成される。ここでの変数の値は，実際には，生成されたオブジェクトを指すポインタ値である。

各行の右端の引数リストは，そのインスタンス生成時に，対応するクラスのコンストラクタへ渡される。文字定数は各々のインスタンスの名前として登録するもので，各クラスの基本クラス Labeled_object の中で宣言されている文字型配列変数 idname に記憶される。Meeting の第2引数から第6引数までは，3種類の会議の各々に対して，安部氏，馬場氏，千葉氏の出席の要否，および会議室と OHP の予約の要否の情報を渡している。Y が要，N が否を示し，P は優先的に出席すべきことを示している。これらの情報は，図 8.8 の Meeting のクラス定義の公開部の変数 s[3]，e1，e2 に記憶される。

(2) 初期処理（オブジェクトのアドレスの配布）

主プログラムの第2集団の4行は，初期処理関数 initialize の実行メッセージを事務局オブジェクトと3人の個人オブジェクトに送っている。その実引数で，分散協調型の計算を行うために必要となるメッセージ送信先のオブジェクトのアドレスを知らせている。

(3) 主業務遂行（会議開催準備指示）

主プログラムの第3集団は，実際の会議開催の準備やその取消を事務局オブジェクト ooo に指示している。たとえば，最初の指示は，「企画会議（KK）を5月11日の14時から15時まで開催する」というものである。このメッセージにより 図 8.12 に示す ooo の arrange 関数が実行される。

この関数の定義側において，第1仮引数の k には，KK の会議管理オブジェクト kk のアドレスが代入される。これを用いて会議室予約の要否を調べると，e1＝＝Y が成立するので予約要であることがわかる。そこで，次に会議室予約管理オブジェクト room_mgr のアドレスを保持している変数 room を用いて，room_mgr に reserve メッセージを送り，指定された時間の予約を試みる。成功すれば，次に abe，baba，chiba の個人オブジェクトに開催通知を送る。この時，KK 会議管理オブジェクトを調べ，出席要(Y)の人には，reserve メッセージを送る。優先出席要(P)の人には，priority メッセージを送る。いずれの場合も，SUCCESS の値が返されると，スケジュール表への予約が成功したこと，すなわち出席を意味するので，KK 会議管理オブジェクトの出欠表 attendant に出席者名を書き込む。最後に，OHP の要否を調べ，e2＝＝Y が成立するの

```
void Office::arrange(Meeting* k, int mm, int dd, int hh1, int hh2) {
  if(k->e1 == Y) {
    if(room->reserve(k->name, mm, dd, hh1, hh2) == FAIL) {
      cout << "Room-reservation failed" << "¥n";
      return;
    }
  }
  for (int i=0; i<3; i++) {
    strcpy(k->attendant [i],"");
    switch(k->s [i] ) {
      case Y: if(xxx [i] ->reserve(k->name, mm, dd, hh1, hh2)==SUCCESS)
              strcpy(k->attendant [i],xxx [i] ->name);
              break;
      case P: xxx [i] ->priority(k->name, mm, dd, hh1, hh2);
              strcpy(k->attendant [i],xxx [i] ->name);
              break;
    }
  }
  if(k->e2 == Y) {
    if(ohp->reserve(k->name, mm, dd, hh1, hh2) == FAIL)
      cout << "OHP-reservation failed" << "¥n";
  }
}
```

図8.12　クラスOfficeのメンバ関数arrangeの定義

で，OHP管理オブジェクトohp_mgrにreserveメッセージを送る。

(4)　結果の確認（予約状況表示）

主プログラムの最後の集団は，結果を確認するために，各オブジェクトにdisplayメッセージを送っている。oooのdisplayでは，3種類の会議管理オブジェクトのattendantを見て，各会議の出席予定者を表示する。そのほかのオブジェクトは，自分のscheduleを見て，5月11日の予約状況または会議出席予定を表示する。

8.3.4 C++の特徴

分散協調システムOOOの例題を通じて,8.1節で述べたオブジェクト指向プログラミングの4つの主要概念(分散協調型計算モデル,データ抽象化,インスタンス生成,クラス階層と継承機能)がC++を用いて実現できることを示した。

しかしながら,この例題ではC++の特徴の一部を紹介したに過ぎない。この他にも,次のようなこの例題の拡張機能を検討することにより,ポリモフィズムや関数と演算子の多義化(overloading)を利用できる

① ポリモフィズム

たとえば,上記の例では,displayという関数がMeeting, Reservation, Officeの3つのクラスで定義されていた。この関数を実行するために主プログラムの第4集団では,個々のインスタンスオブジェクトにそのメッセージを送っている。ここで,新たにこれら3種類のクラスに共通の基本クラスであるLabeled_objectに,displayを仮想関数として定義しておけば,Labeled_object型のポインタ変数に上記3種類のいずれのインスタンスオブジェクトのアドレスを設定してdisplay関数を呼び出しても,実行時に3つのdisplay関数の中から適切なものを選んで実行してくれる。会話型システムの構築の時などに便利な機能である。

② 関数の多義化

たとえば,上記の例では,Officeで定義されたdisplay関数は,3種類の会議管理オブジェクトのattendantを見て,各会議の出席予定者を表示するという機能に限定されていた。しかし,実際には,特定の会議だけの出席予定者を表示するとか,特定の個人オブジェクトの名前を指定したらその人の予定を表示するとか,月日のうち,月だけ指定したらその月の一ヶ月分の予定を表示し,日も指定されたらその日だけの分を表示するなど,いろいろな照会のバリエーションがある。これらの少しずつ異なる関数をその都度名前を変えて定義していてはプログラムが複雑になってしまう。このようなときには,C++の関数の多義化機能を用いて,すべて同じdisplayという名前で定義できる。実際に関数呼び出しがあったときには,引数の型が合致するものを選んで実行してくれる。

8.4　オブジェクト指向パラダイムの応用分野

8.4.1　概　要

オブジェクト指向概念は，種々の技術分野に応用されている。その各分野に比較的共通と思われる，ソフトウェア生産技術としての一般的特徴については8.2節で述べた。ここでは，オブジェクト指向概念の応用技術分野の例について，下記の技術分野別に述べる。

- ソフトウェア基礎論（計算モデル）
- プログラミング言語
- 仕様記述
- 部品化・再利用
- 知識工学（知識表現）
- ユーザインタフェースおよび構築ツール
- オブジェクト管理システム
- データベース
- オペレーティングシステム
- ネットワーク
- シミュレーション

8.4.2　ソフトウェア基礎論（計算モデル）

現在，広く利用されている大半のコンピュータは，フォン・ノイマン型といわれる「汎用」アーキテクチャを採用している。この「汎用」コンピュータは，利用目的に応じたソフトウェアプログラムを付加することによって「専用」コンピュータになる。コンピュータの動作手順を決めるプログラムの記述は，当初は機械語の列で表現していたが，現在では，COBOL，FORTRAN，Cなどの高級言語が用いられている。これらの高級言語は，コンピュータのアーキテクチャに合わせてプログラムを手続き的に記述しなくてはいけないという意味においては機械語と同じである。

ところが，実際にコンピュータにやらせたい仕事は，このような手続き的な表現に適さないことも多い。この問題は，セマンティックギャップ（semantic gap）と呼ばれ，できるだけ対象とする問題に適した方法でプログラムを記述す

るプログラミングパラダイムの研究がなされてきた。この研究は，ある計算モデルを提案し，それをプログラミングパラダイムとする言語を開発するという方法を採ることが多い。

計算モデルとしては，これまで種々のモデルが提案されてきた。オブジェクト指向概念の計算モデルとしては，C. Hewitt の Actor モデルがある。このモデルは，人工知能研究のための問題解決用言語 PLANNER の意味論の研究の中で生まれたものであるが，次のような特徴から，オブジェクト指向概念の計算モデルとみなすことができる。

- ●計算の単位となる actor は，手続きやデータの概念を含み，外部からのメッセージによって起動される自律的なモジュールであり，オブジェクトの概念と同じである。
- ●全体の計算は，これらの actor 間のメッセージのやり取りによって実行される。

このモデルは，従来の手続き型計算の世界の種々の概念，すなわち，データ抽象化，手続き，関数，プロセス，並列計算などの概念を包含した非常に汎用性の高いモデルである。

Actor モデルを用いて folk 機能と join 機能による並行処理を記述した例を図 8.13 に示す。この例における処理の概要を以下に説明する。

① F に対し，「$f(x)=h(g_1(x), g_2(x))$ を計算して，結果を R へ送れ」というメッセージが発行された。

② F から G_1 に対し，「$g_1(x)$ を計算して，結果を H へ送れ」というメッセージを発行した。

③ F から G_2 に対し，「$g_2(x)$ を計算して，結果を H へ送れ」というメッセージを，②と並行して発行した。(folk 機能)

④ G_1 から H に対し，「$g_1(x)$ の計算結果を送る」というメッセージを発行した。

⑤ G_2 から H に対し，「$g_2(x)$ の計算結果を送る」というメッセージを発行した。

⑥ H は，④と⑤の両方のメッセージを受信した後，R に対し，「$h(g_1(x), g_2(x))$ の計算結果を送る」というメッセージを発行した。(join 機能)

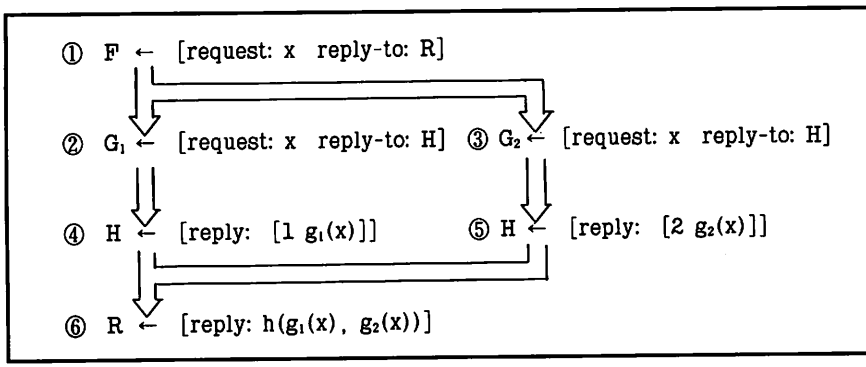


図8.13　Actorモデルによる並列処理の記述例(文献(52)より引用)

8.4.3 プログラミング言語

オブジェクト指向言語の代表例として Smalltalk がある。1980年に実用化された Smalltalk-80 は，オブジェクト指向概念を徹底して追究したため，言語としては美しいが，細かい処理の記述に関して従来のプログラミング言語からみて不自然に見える部分もある。しかしながら，実用化された先駆的な言語として，その後のオブジェクト指向言語の研究に与えた影響は大きい。

詳細はすでに8.1.1項で述べたが，Smalltalk-80 が発表された後，種々の言語が開発された。そのほとんどは，既存の言語をベースにオブジェクト指向機能を導入したハイブリッド型あるいはマルチパラダイム型と呼ばれる言語である。筆者らも，対象世界のマクロな計算モデルの自然な記述にオブジェクト指向を用い，オブジェクト内のミクロな計算には既存の言語を用いる2階層モデル方式を開発し，その実現例として，後者に Prolog を用いるプログラミング言語 S-LONLI を開発した経験を有するが，その内容は第10章で述べる。

8.4.4 仕様記述

ソフトウェアの仕様には，要求仕様，設計仕様，モジュール仕様など種々のレベルのものがあるが，いずれもなんらかの計算モデルあるいはプログラミングパラダイムをベースに記述される。要求分析および要求仕様定義の従来技法としては，機能階層モデル，ER モデル (entity relationship model)，状態遷移モデル，プロセスモデルに基づく方法などがある。代表的なものとして，機

能階層モデルをベースに各階層の機能をデータフロー図（DFD：data flow diagram）を用いて記述する構造化分析技法（SA：structured analysis）や，プロセスモデルをベースに実世界の実体とその動作に注目した仕様化手順を与えるジャクソン法(JSD：Jackson development method)などがある。設計技法としては，具体的なモジュール分割法を与える構造化設計技法（SD：structured design）や複合設計法がある。

しかしながら，従来技法は，要求分析から設計へ，設計からプログラミングへのつなぎ目のところに大きなギャップがあった。これは，上流工程で作成する仕様の記述に用いる計算モデルとそのすぐ下流工程で作成する仕様の記述に用いる計算モデルとが異なり，相互の変換が必要なためである。人間にわかりやすい実世界のモデルとコンピュータが理解可能なプログラムの間に存在するこのようなセマンティックギャップをどのように克服するかはソフトウェア工学上の重要課題である。

オブジェクト指向概念はその1つの有力な候補である。すなわち，以下の3つの技法は同じ計算モデルに基づいているため，要求定義，設計，プログラミングの各工程の間のギャップが小さくてすむという期待がある。

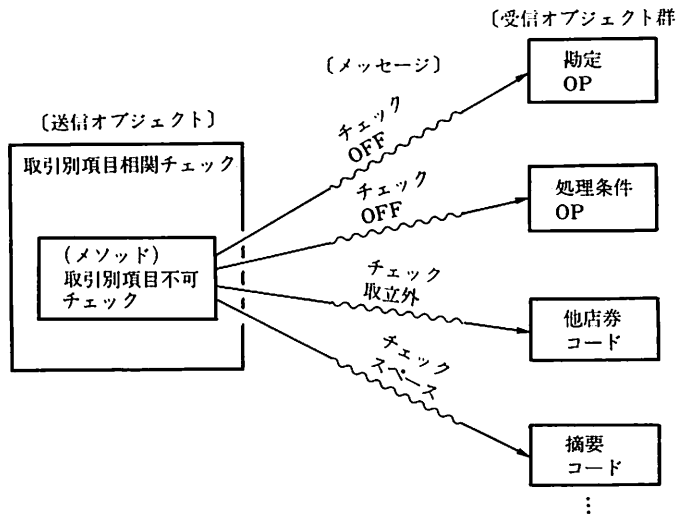
- オブジェクト指向分析技法（OOA：object-oriented analysis)
- オブジェクト指向設計技法（OOD：object-oriented design)
- オブジェクト指向プログラミング技法
  (OOP：object-oriented programming)

まだ十分な実績は積んでいないが，いくつかの具体的な技法が提案され，実用化されつつある。

筆者らも，プログラムの機能仕様を形式的に記述し，それからプログラムを自動生成するシステム SPECTRUM を開発した経験を有する。このシステムの仕様記述言語として，オブジェクト指向概念におけるデータと手続きのカプセル化機能とメッセージパッシング機能を表現可能な表形式言語を開発した。その記述例を 図 8.14 に示す。これは，銀行オンラインシステムの業務プログラムの仕様記述の一部である。表の左端の欄の上から下に「取引別項目相関チェック」オブジェクトの「取引別項目不可チェック」メソッドの処理が記述されている。上端の欄には本オブジェクトからメッセージ送信をするオブジェクト

取引別項目相関チェック				勘定 OP	処理条件 OP	他店券 コード	摘要 コード	流動 暗証
method 取引別項目不可チェックis								
僚店　?								
YES	"僚店処理"					チェック 取立外		
NO								
不渡り　?								
YES	"不渡り処理"			チェック OFF	チェック OFF	チェック コード	チェック スペース	チェック スペース
NO	入金　?							
	YES	取消　?						
		YES	"支払取消"	チェック OFF		チェック コード		チェック スペー
		NO	"通常入金"	チェック OFF	チェック OFF			チェック スペー
	NO	取消　?						
		YES	"入金取消"	チェック		チェック		チェック

(a)　表形式による形式的仕様記述例



(b)　上記の例に対応する概念図

図8.14　オブジェクト指向ベースの仕様記述言語 SPECTRUM の記述例

が記載され，それへのメッセージがその下に書かれている。具体的なメッセージ送信の概念図を示したものが図の(b)である。なお，本システムの詳細は後の日本語プログラミングのところで述べる。

8.4.5 部品化・再利用

部品化，再利用については，8.2.2項ですでに概略を述べた。その具体例については，本節の各項で随時言及する。

ここでは，ビジネス分野の業務プログラムへの応用例として，既存のソフトウェアの仕様書を再利用するシステムREUSEを紹介しておく。REUSEでは，要求文に表れない隠れた意味を推論するために，一定の規範に基づいて構築した知識体系を用いる。そのためにオブジェクト指向モデルを適用し，図8.15に示すように，業務で実際に使われている対象物と業務処理の組を知識単位（部品）とすると共に，対象物あるいは概念を分割または抽象化することにより，知識（部品）を構造化している。

8.4.6 知識表現

現在，実用化が進んでいるエキスパートシステムなどの知識ベースシステムでは，知識をどのような形で記憶し，利用するかが重要な課題である。そのための知識表現言語としては，基本言語としての関数型言語Lispや論理型言語・Prologの他に，プロダクションシステムのルール，フレーム，意味ネットワークなどが代表的である。

この中でフレームは，M. Minskyが人間の記憶構造や推論過程の説明のために提案した概念である。このフレームは，知識の構造化や知識間の関係の記述が容易なため，多くのエキスパートシステムで利用されている。これらのフレームの特徴は，9.3節で述べるが，先に述べたオブジェクト指向概念との類似性が強いため，両者はあまり厳密な区別はせずに使われることが多い。実際にオブジェクト指向型知識表現を含むエキスパートシステム構築ツールが数多く開発されている。

筆者らが開発したエキスパートシステム構築ツールES/X90においても，その知識処理言語にはマクロな記述にオブジェクト指向パラダイムを用いている。詳細は第10章で述べる。

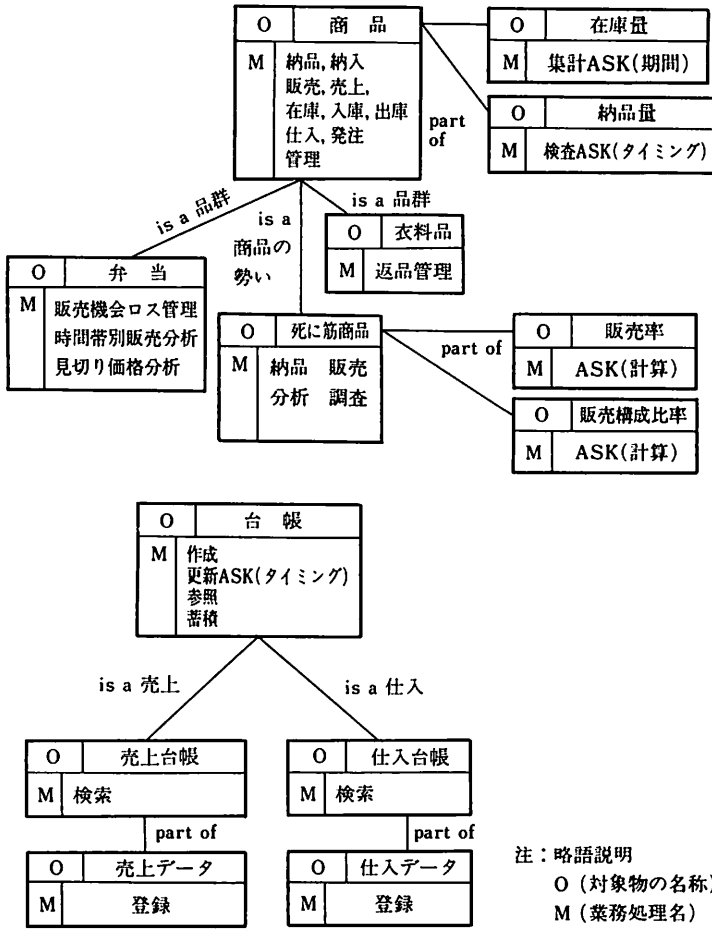


図8.15　部品化再利用システム REUSE の部品構成(文献(60)より引用)

8.4.7　ユーザインタフェースと構築ツール

コンピュータのパーソナル化に対応して，だれでも簡単に使えるソフトウェアを実現するために，ユーザインタフェースは非常に重要になっている。オブジェクト指向概念がこのユーザインタフェースの操作性の容易化と統一性の確

保に適していることはすでに8.2節で述べた。ここでは，ユーザインタフェースそのものとその構築ツールの例を示す。

(1) ユーザインタフェース

オブジェクト指向概念は，使いやすいユーザインタフェースのための3つの特徴を実現する。

① メタファ（暗喩）

オフィスシステムにおける書類，フォルダ，キャビネ，あるいは制御システムにおけるパラメータ設定盤などに対応するアイコンをビジュアルオブジェクトとして画面に表示することにより，操作の対象がわかりやすくなる。

② 直接操作

このビジュアルオブジェクトにデータやメソッド（操作手続き）を持たせておくことにより，画面上で「操作対象のビジュアルオブジェクトを指定して，必要なメソッドを選ぶ」という直感的で判り易い直接操作が可能となる。

③ 統一性

画面上の表示物をすべてオブジェクトとして作成することにより，操作手順をオブジェクトにメッセージを送るという上記の直接操作で統一でき，単一操作で簡単にコンピュータと対話ができる。

このようなグラフィカルユーザインタフェースはすでに一般化しているが，先駆的システムとして，Apple社のハイパメディアシステムHyperCardがある。**図8.16**にHyperCard(日本語版)の初期画面を示す。画面上のアイコンの1つを選択すると，それに対応するスタック（カードの集まり）と呼ばれるドキュメントが使用可能になる。カードの上にボタンやテキスト，ピクチャなどを並べて他のカードとリンクをはることができる。少し複雑な処理はスクリプト言語で記述する。

(2) ユーザインタフェース構築ツール

個々の応用ソフトウェアのユーザインタフェースは，各々の中で一貫性があるだけでなく，異なる応用ソフトウェアの間でも統一がとれている方が利用者の使い勝手がよい。そのためには，開発者にとってもユーザインタフェースが作りやすくなっていることが重要であり，すでに多くのユーザインタフェース

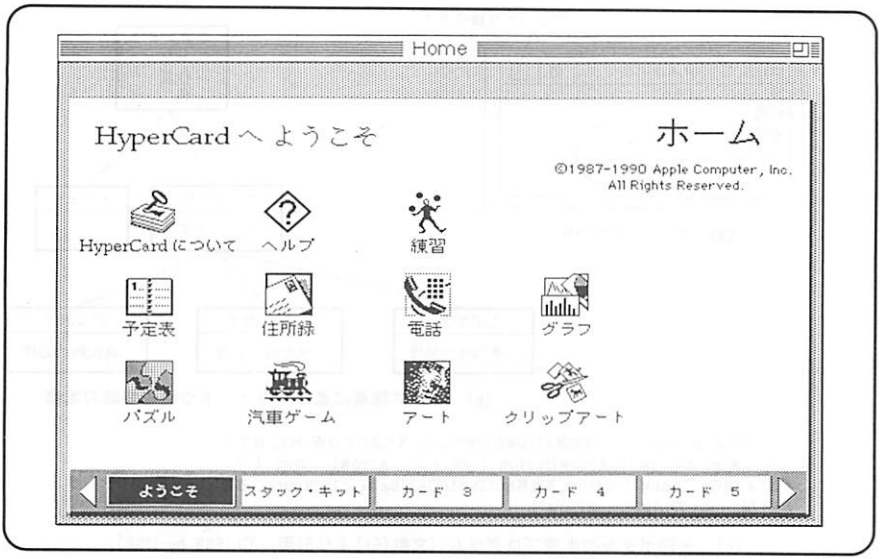
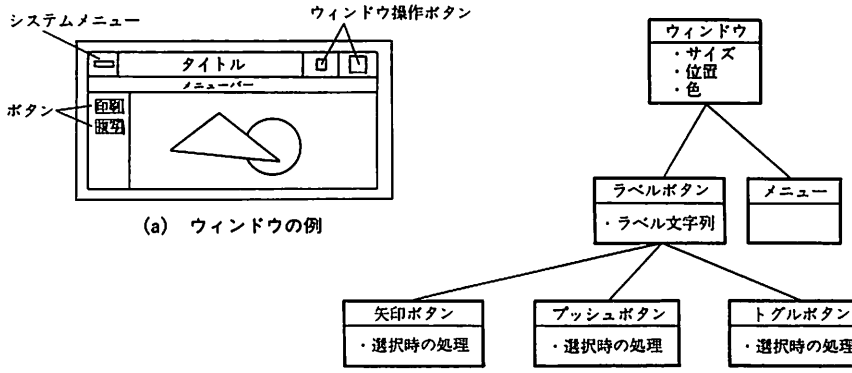


図 8.16　HyperCard（日本語版）の初期画面
（アップルコンピュータジャパン提供，© 1987-1990）

構築ツールが商品化されている。

たとえば，OSF/Motif では，スクロールバーやボタンなどのウィンドウの構成部品を widget と呼ぶオブジェクトとして定義した部品ライブラリを提供している。開発者は，この widget に個々の属性に対応したパラメータ付きのメッセージを送ることにより，ユーザインタフェースを構築できる。その一例を図 8.17 に示す。図の(a)は，Motif のウィンドウの例である。図の(b)は，矢印ボタン，プッシュボタン，トグルボタンの3種類のボタンのクラス定義がウィンドウおよびその構成要素に関するクラス階層の中で定義されていることを示す。ウィンドウクラスのインスタンスは，位置，サイズ，色を変更する手続きと値を格納するデータ領域を持つ。ウィンドウクラスのサブクラスはすべて，サイズ，位置，色を変更するための手続きを継承するため，親クラスが持っていない手続きのみを作ればよい。そして，たとえば，「印刷」を指示するプッシュボタンを作る場合，プッシュボタンクラスのインスタンスを生成し，以下のデータに値を設定するだけでよい。




```
XtSetArg (args [0], XmNarrowDirection, XmARROW-RIGHT );
arrow = XmCreateArrowButton ( parent, "Arrow1", args, 1 );
XtAddCallback ( arrow, XmNactivateCallback, click_proc, NULL );
XtManageWidget ( arrow );
```

(c) 矢印ボタンの生成プログラム (文献(61)より引用, © 1989 by OSF)

図8.17 OSF/Motifによるユーザインタフェース記述例

インスタンス名：“印刷ボタン”
サイズ　　：100 * 50
位置　　：20, 60
色　　：青
ラベル文字列　：“印刷”
選択時の処理　：プリント

図の(c)は矢印ボタンのクラスからインスタンスを生成するプログラムの例である。1番目の文は，矢印ボタンのインスタンス生成のための引数設定である。第一引数は引数の設定エリア，第二引数は矢印の向きを指定すること，第三引数は矢印の向きが右であることを指定している。2番目の文は，矢印ボタンのインスタンス生成関数の呼び出しである。第一引数は親クラス，第二引数はインスタンス名，第三引数は引数の設定エリア，第四引数は引数の数を指定している。3番目の文は，このインスタンスがマウスでクリックされた時にコールバックされる手続きがclick_procであることを指定している。4番目の文は，このインスタンスの表示を依頼している。

以上の例はテキストベースで記述するものであったが，グラフィカルユーザインタフェースを画面上で直接作成するツールもある。NextStep (Next 社) のインタフェースビルダは，画面上のビジュアルオブジェクトのセットの中から必要なものをコピーして，位置や大きさを自由に変えたり，関係するものを線で結ぶという方法で，ユーザに見せる画面を直接作成できる。複雑な計算処理が必要になったときは，部分的に Objective-C を用いてプログラムを記述する。

このようなユーザインタフェースの標準化を促すために，IEEE の OS の技術委員会では，ユーザインタフェースのレファレンスモデルを定める活動が行われている。さらに，ユーザインタフェース構築ツールの発展として，応用プログラムとユーザインタフェースを完全に分離するためにユーザインタフェース管理システム (UIMS：User Interface Management System) を導入する試みが始まっている。このような方式を用いれば，応用プログラムの移行性がいっそう高まる。

8.4.8 オブジェクト管理システム

オブジェクト指向ベースのアプリケーションソフトウェアを次々と開発していくとき，アプリケーション間に共通のオブジェクト管理の処理をアプリケーションから分離，独立して共有できれば，個々のアプリケーションの開発効率や保守，拡張性が良くなるし，異機種間の移行性も向上する。このような目的を持つオブジェクト管理システムは，オブジェクト指向アプリケーションアーキテクチャのインフラストラクチャとして重要である。

このようなシステム構成について，HP 社の NewWave の例で説明する。このシステムでは，テキスト，グラフ，スプレッドシート，ボイスオブジェクト等を統合して，マルチメディアのドキュメントを簡単に作ることができる。図 8.18 に複合オブジェクトの例を示す。図中の線はオブジェクト間の関係を表す。シンプルリンクは，単純な親子関係を表す。ビジュアルリンクは，子オブジェクトが親オブジェクトと表示や印刷の内容を共通化することを表す。この例では，チャートオブジェクトのグラフがドキュメントオブジェクトの構成要素として取り込まれており，双方のオブジェクトのグラフ表示は常に同じになる。データパッシングリンクは，子オブジェクトから親オブジェクトへデータ

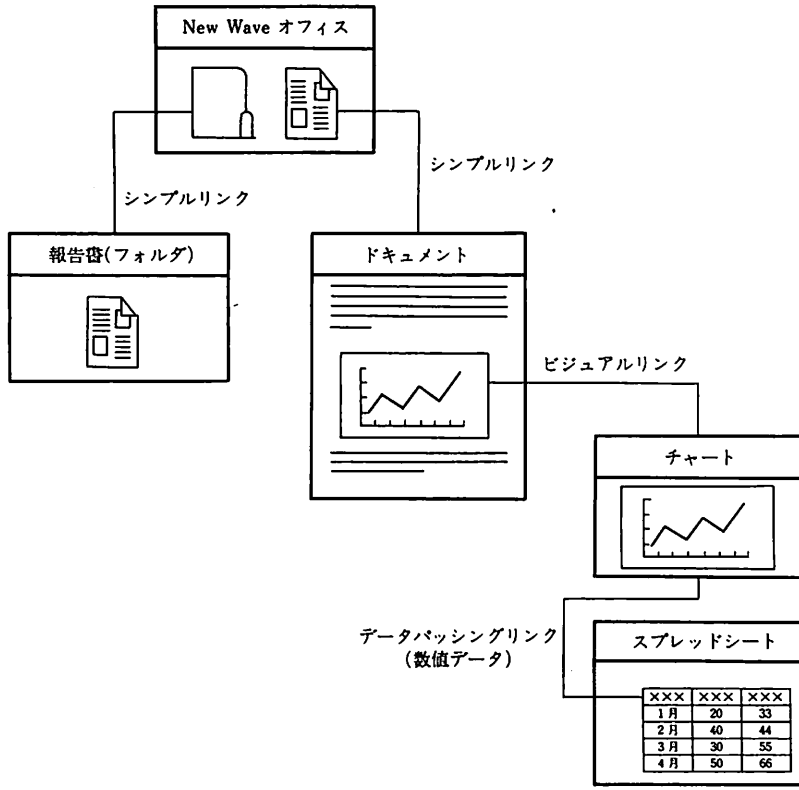


図8.18 NewWaveにおける文書データのオブジェクト構成例
(文献(62)より引用, © 1989 by Hewlett-Packard)

値が渡されることを表す。この例では，スプレッドシートのオブジェクトから渡された数値データに基づいて，チャートオブジェクトの中に対応するグラフが作成される。スプレッドシートの値が変わればグラフも自動的に変更される。

このような機能を簡単に実現するために，文書，図，表，音声データなどの基本的なユーザオブジェクトの管理や，グラフや図を含んだ文書のように異種のユーザオブジェクトを含んだ複合オブジェクトの管理，あるいはオブジェクト間にリンクをはるハイパーメディア機能などを含めて，オブジェクト管理のプラットフォームを標準化しようとする活動が活発に行われている。米国では，パソコンや大型機を含む異機種間ネットワーク上で稼動する異なるアプリケーションを統合するような分散コンピューティング向きプラットフォームを作ろ

うとしている。ヨーロッパでは，CASEツールの移行性を目的としたプラットフォームが開発されている。

8.4.9 データベース

データ管理技術の観点でこれまでのデータベースの発展過程を見ると，1960年頃のファイルシステム以降，1963年にネットワーク型データベース，1968年に階層型データベース，1970年にリレーショナルデータベースが登場してきた。特に，最後のリレーショナルデータベースは，集合論と関係論理という数学的基礎を持ち，表形式のデータモデルの簡潔さ，データ独立性，厳密性などの特長のために広く普及している。

しかしながら，これらのデータベースがこれまで主に数値や長さの限られた文字列を扱ってきたのに対し，最近の情報化の進展に伴い，文章（テキスト）データ，図形データ，イメージデータなど，データ構造が複雑なものを扱う必要が生じてきた。たとえば，テキストと表，図，グラフなどが混在する文書を扱うオフィスシステム，さらに画像，作図なども含む設計図や設計仕様書を扱うCAD／CAMシステムやソフトウェア開発支援環境のCASEシステムをはじめとして，コンピュータ化の進展に伴い，印刷分野，医療分野，教育分野など，多くの分野でそのニーズが高まっている。

このようなマルチメディアデータを扱うデータベースのための基本技術（データモデル）としてオブジェクト指向概念が利用されている。従来のリレーショナルデータベースなどと異なるオブジェクト指向データベースの特徴は以下のようなものである。

①実世界の物や概念をオブジェクトとして統一的に表現できる。情報としてのデータの形態（メディア）が異なっていてもよい。またデータとしては全く同じものでも実体が複数あれば複数のオブジェクトを作成できる。

②複数のオブジェクトからなる複合オブジェクトも1つのオブジェクトとして統一的に操作できる。(part-of関係の表現)

③データと手続きのカプセル化機能により，データの構造に依存した操作手続きをデータに埋め込むことにより，アプリケーションソフトウェア

を簡単な構造にでき，拡張，保守が容易になる。

④クラスの階層化と継承機能により，知識表現に用いた場合と同じように，データベースの知的検索が可能となる。

さらに，従来のリレーショナルデータベースでは，データベース操作言語SQLが表を処理単位とするのに対し，アプリケーションソフトウェアを記述するプログラミング言語がデータレコードを処理単位とすることから生じるいわゆるインピーダンスミスマッチ(impedance mismatch)の問題があったが，オブジェクト指向データベースでは，オブジェクト指向プログラミング言語と組み合わせることにより，この問題が解消する利点もある。

すでにオブジェクト指向言語のSmalltalkやC++をベースにしたもの，PrologやSQLをベースにしたものなどが商品化されている。

8.4.10 オペレーティングシステム

かつて大型計算機が中心の時代には，オペレーティングシステム (OS) は高価なハードウェア (CPUとメモリ) をいかに有効に使うかというリソース管理が主な役割だった。しかし，「1つのアプリケーションソフトウェアが沢山のハードウェアの上で動く」ことが重要になってきた現在では，OSには，

①OSとアプリケーションソフトウェアとのインタフェースができるだけ抽象的なレベルで標準化されること，

②OS自身の拡張性や移植性が高いこと，

③分散処理や並列処理の機能を持ちながら，安全性や信頼性も確保できること，

などの要求がある。

オブジェクト指向概念はこのような要求に対応できる有力な技術としてOSの開発に利用され始めている。各種のリソースをオブジェクトという概念で統一的にとらえ，リソースへの処理手続きを付加するという，データと手続きのカプセル化により，リソース固有の処理をアプリケーションソフトウェアから見えなくすることができ，上記①を実現できる。さらに，このような相互に独立性の高いオブジェクトを組み合わせてOSを構築することにより，上記②を実現できる。また，従来のプロセスやタスクをオブジェクトとしてとらえ，オ

ブジェクト指向概念のメッセージ駆動型の分散協調モデルを用いて，柔軟な分散，並列システムを構築することができる。この場合，オブジェクト単位にアクセス権を設定することにより安全性と信頼性を確保できる。すでにこのような設計思想に基づいたOSが開発されている。

8.4.11 ネットワーク

異機種コンピュータを接続する通信プロトコルの国際標準であるOSIのネットワーク管理に関する部分には，規格自身がオブジェクト指向の概念で記述されているものがあり，オブジェクト指向に基づく設計がしやすくなっている。OSIのシステム管理では，管理する側と管理される対象が明確に分離され，相互にメッセージのやりとりをするモデルが採用されている。図8.19に示すように，管理対象のネットワーク機器や回線はそれぞれの属性を持ったオブジェクトとして扱われる。これらの管理対象オブジェクト群はエージェントが監視している。管理する側はマネージャと呼ばれ，管理対象オブジェクトへの管理操作の指示や管理対象オブジェクトで発生した事象の報告は，このエージェントとのメッセージ交換により行う。

実際のネットワーク管理システムの開発においては，プロトコルや管理機能，管理対象の拡張，変更容易化のために，このOSI管理に基づくオブジェクトクラスやクラス定義テンプレートを導入したオブジェクト指向設計が試みられている。

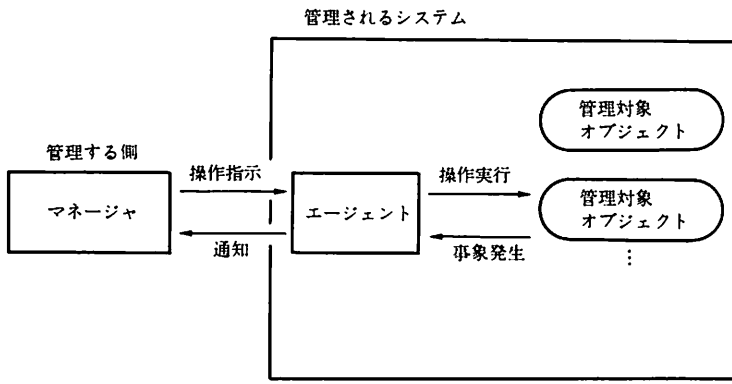


図8.19　ネットワークの管理モデル

8.4.12 シミュレーション

オブジェクト指向概念がシミュレーション技術に適していることは，先駆的オブジェクト指向言語 Simula が 1966 年にシミュレーション言語として開発されたことからも明らかである。その当時，シミュレーションモデルは，再定義と修正の繰り返しによって作られ，モデルは，データ構造とその演算から構成されていた。そこで，対象毎にシミュレーション言語を簡単に作るための言語として Simula が開発された。この言語は，Algol60 をベースに設計されたが，この時，オブジェクト指向概念の重要な特徴であるデータ抽象化，クラスからのインスタンス生成，クラスの階層と継承などの機能が付加された。

最近では，コンピュータグラフィックスの分野で，シナリオ記述言語などにオブジェクト指向言語が使用されている。