岐阜経済大学論集 30 巻 2 号（1996 年）

経営情報システム・モデルの開発環境

安 田 晶 彦

1. はじめに

1990 年代に入って，パーソナル・コンピュータに代表される小型のコンピュータの性能向上が特に著しかったため，経営情報システムの開発研究を行うための環境が，パーソナルなレベルでもかなり整ってきた。パーソナル・コンピュータによって，大規模な経営情報システムをそのまま再現することはできないものの，ダウンサイジングを想定したシステムのモデル研究を行うことは十分に可能になっている。

しかし，パーソナル・コンピュータの場合には，COBOL によるメインフレーム上でのシステム開発のような，いわば定番的なスタイルが明確ではなく，経営情報システムの開発環境を状況の変化に対応した形で見直す必要が出てきていることも事実である。

幸いなことに，パーソナル・コンピュータのプログラム開発環境は，安価なパッケージ・ソフトウェアとして豊富に存在しているため，ごく容易に開発環境を整えることができる。問題は，メインフレームにおける COBOL に

よる経営情報システム開発を，パーソナル・コンピュータで再現しても大した意味はなく，これからの経営情報システムを模索しながら開発に取り組まなければならない点であろう。

本稿ではこうした問題について検討を試みることにする[1)]。

パーソナル・コンピュータは，小型で安価なパーソナル情報処理機器として発達してきたため，経営情報システムにおける役割も限定されたものであった[2)]。1980年代においては，パーソナル・コンピュータをベースとする経営情報システムの開発は特に考えられていなかった。中規模以上の経営情報システムの場合，LANのサーバー・コンピュータがシステム開発の中心に置かれ，パーソナル・コンピュータは端末としての役割を担うのが一般的であった。

しかし，パーソナル・コンピュータは台数もユーザー数も多く，圧倒的に普及しているため，ソフトウェア開発の教育・研究環境として広範囲に利用可能であるというメリットがある。また，アプリケーション・ソフトウェアも，需要の大きさの恩恵を受けて安価で使いやすいものが販売されている。さらに，近年の飛躍的な性能向上により，パーソナル・コンピュータ自体が，小規模なLANのサーバー・コンピュータとしての能力を有するまでになっている。

従って，経営情報システムの教育・研究を具体的に進めていく上で，パーソナル・コンピュータのシステム開発環境は決して無視できない存在になってきていると言えよう。

経営情報システムのようなアプリケーション・ソフトウェアを取り上げる場合，実務的な立場と教育・研究の立場とでは，異なったスタンスをとる場合がある。前者においては，特定の開発環境を前提として，標準や移植性の問題を差し当たり考慮せずに，もっぱら開発効率とコスト・パフォーマンス

の面からシステム開発を追求するケースがあり得るが，後者では標準や移植性を軸にすえたアプローチをとることになる。この論文では，後者の立場から経営情報システムの開発環境を論じている。

しかし，後者の局面においても，経営情報システムのモデルを実際に構築する段になると，標準化された開発ツールではカバーしきれない面を，いわゆる業界標準（事実上の標準；de facto standard）によって補わざるを得ないのが実状である。コンピュータに関する標準は，コンピュータ技術の発達が成熟していない以上は流動的なものと考える必要があり，そうした状況では業界標準に依存せざるを得ないのである。本稿では，標準化されたツールで経営情報システムのパーソナルな開発環境をどこまで整備できるのかについて，現状を考察してみたいと考えている。

経営情報システムは企業や官庁のような大規模な組織において，大量のデータ集計を行う業務に計算機を導入したのがきっかけとなって発達してきた。これらのソフトウェアの開発は，ソフトウェア著作権やセキュリティ管理の面から非公開で行われることが多い。従って，プログラミング言語とシステム設計に関する一般的な教育・研究は可能なものの，経営情報システムの実態に関する具体的な教育・研究は基本的に困難なのである。

小型のコンピュータ，あるいはそれらによるネットワークによって実現される経営情報システムのモデルは，経営情報システムを小規模にエミュレート（emulate）するものであるが，実際の経営情報システムがいわば「関係者以外立入禁止」の区域にあって，一般には開放されていない以上，経営情報システムの具体的な研究のあり方としては，十分に意義があると考えられる。

経営情報システム・モデルの開発環境としては，標準化された高水準言語とその開発環境を考察の中心的な対象とし，必要に応じて標準的な基本ソフ

トウェアについても検討を試みることにする。

基本ソフトウェアについて検討しなければならないのは，高水準言語では標準化されていないユーザー・インターフェイスの問題があるからである。また，データベースとネットワークへの対応も，基本ソフトウェア依存の面があるため，現状を把握しなければならない。データベースに関しては，汎用のプログラミング言語だけではなく，データベース言語，つまりDBMS (Database Management System) に付属のアプリケーション開発用言語についても考察する必要があろう。

2. プログラミング言語の変遷

経営情報システムの開発において，伝統的に採用されてきた言語は言うまでもなくCOBOLである。COBOLはFORTRANに次いで古い高水準言語であり，標準化も早くから進められていたので[3]，ソフトウェアの異機種への移植が容易な事務計算用プログラミング言語として広く普及した。

COBOLは汎用のプログラミング言語としてではなく，事務計算専用のプログラミング言語として分類されることもある。その分だけ，経営情報システムの開発言語として最適な言語と考えられてきた。従って，企業などの組織における事務処理を目的としたソフトウェアは，もっぱらCOBOLを用いて開発されてきたと言っても過言ではない。経営情報システム関連の教育・研究機関においても，標準化された言語としてのCOBOLの利用率は極めて高かったと言える。

しかし，近年ではCOBOLの利用率は徐々に低下しており，代わりにCやデータベース言語などの他のプログラミング言語が，事務計算用のプログラム開発において注目されるようになってきている。

COBOLの利点は，① 事務計算用に特化したプログラミング言語なので，

事務計算のソフトウェア開発が容易であり，プログラマー養成も比較的容易であること，② 10進演算を行うので，他のプログラミング言語の場合と異なり，2進演算の誤差対策が必要ないこと，③ 前述のように標準化が進んでいるので，COBOLの開発環境を持つ異機種間での移植が容易なことがあげられる。

COBOLの欠点としてあげられるのは，① プログラムを事務文書に似せるため，表現が冗長であり，他の汎用高水準言語と比較して仕様が異質なこと，② 古いプログラミング言語であるため，パンチカードに合わせた書法が残存していることなどである。

また，ANSI '85より前のCOBOLでは算術演算機能が貧弱だったため，意思決定支援システムのような設計思想を実現するのにCOBOLは向いていなかったと言える。ANSI '85で各種の数学・統計関数が使えるようになり，意思決定支援システムもようやく容易に構築できるようになった[4]。

COBOLはメインフレーム上の開発環境として長らく使用されてきたが，パーソナル・コンピュータに代表される小型のコンピュータの普及によって，状況は変化してきている。パーソナルな開発環境においては，COBOLはかなりマイナーな存在であり，それほど普及しているとは言えない。このことを反映して，ダウンサイジングの過程で，COBOL一辺倒だった事務計算の開発環境にも変化が訪れつつある。

ミニコンピュータ，そしてワークステーションではC，初期のパーソナル・コンピュータではBASICというように，比較的小型のコンピュータでは，メインフレームと異なりCOBOLは主たる開発言語たり得なかったのである。ミニコンピュータやワークステーションの場合は，OSとしてUNIXが普及したため，UNIXの開発言語となったCが主力言語として定着した経緯があり，パーソナル・コンピュータにおいては，初期のマイクロ・コンピュータのごく限られた能力の範囲内でプログラムを開発・実行可能な開発

環境としてBASICインタプリタが採用されたという経緯がある。

プログラミング言語を比較検討する場合，① 標準化が進んでいるかどうか，② どのような用途のシステムを開発するのに適しているか，③ 必要なプログラミング・パラダイムをサポートしているかどうか，④ プログラマーの人数が多いかどうかといった様々な基準が考えられる。

プログラマー数に関して言えば，メインフレーム時代に経営情報システムの開発言語としてCOBOLが使用されてきたのは，メインフレームにおいてCOBOLの利用実績が累積して無視できない状況に至ったからであると言えるし，経営情報システムの開発に携わるプログラマーの多くがCOBOLプログラマーであったからだとも言える。

プログラミング言語の主流が代わっていく原因は，実のところ，多くのユーザーが利用するコンピュータ，特にその基本ソフトウェアが変わっていくからなのである。

UNIXを基本ソフトウェアとするミニコンピュータが普及していった際には，UNIXにCが標準的に備わっていたことから，ミニコンピュータに携わるプログラマーは必然的にCプログラマーとなっていった。

マイクロ・コンピュータが登場した時点では，そのハードウェアの能力的制約から言語仕様の小規模なBASICインタプリタが基本ソフトウェアとして定着したため，マイクロ・コンピュータのユーザーは必然的にBASICプログラマーとなっていった。

経営情報システムの開発言語として，COBOLに代わって台頭しつつある言語はやはりCであろう。

Cの開発環境は，ワークステーションのみならず，パーソナル・コンピュータにおいても急速に普及するに至った。

パーソナル・コンピュータにおいても，IBM/PCとその互換機用のPC-

DOS ないし MS-DOS が主流になると，プログラミング言語の主流は BASIC から C へと移行していった。BASIC は基本的にはプログラミング入門用言語として位置づけられており[5]，ポインタのような低水準の機能がなく，システム・プログラムの開発には不向きな言語仕様となっていた。C の場合は，UNIX のシステム記述に用いられたように低水準のプログラミングが柔軟に行えるため，MS-DOS によってディスク・ファイルの管理が可能になった時点で，BASIC に代わってパーソナル・コンピュータの主力言語となっていったのである。また，パーソナル・コンピュータ用の C 開発環境が安価に提供されたため[6]，ユーザーも増加の一途をたどったのであった。

パーソナル・コンピュータにおいては BASIC や C が普及したため，経営情報システム・モデルの開発環境として COBOL にこだわるのは無理がある。現時点では，C は COBOL に比べて，柔軟性に優れており，システム・プログラミングからアプリケーションまで幅広い分野で利用できる言語だと言えるからである[7]。

C のユーザー数の増加を背景に，新しいプログラミング・パラダイムに対応した言語が C をベースに開発され，注目されるに至った。C++ がそれである。

3. プログラミング・パラダイム

プログラミング言語の仕様は，改訂によって他のプログラミング言語の優れた部分を採り入れるようになっているので，プログラミング言語間の一時的な優劣をもとに，経営情報システム・モデルの開発に適するプログラミング言語を論じるだけでは限界がある。プログラミング言語を考察する上での重要な指標として，プログラミング・パラダイムがある。

1970 年代の前半に提唱され，プログラミング言語 Pascal によって実践さ

れた構造化プログラミング（structured programming）は，ひとつの新たなプログラミング・パラダイムとして，プログラミング・スタイルに大きな影響を与えた。C も構造化プログラミングをサポートしており，BASIC や COBOL，そして FORTRAN も，改訂によって構造化プログラミングをサポートするようになった。

現段階では構造化プログラミングをサポートしていないプログラミング言語は実用性を失ってしまったと言っても過言ではない。ANSI '74 COBOL や初期のパーソナル・コンピュータ用の BASIC などはその例である。

構造化プログラミングは，システムの保守という点から，経営情報システムの開発においても重要なインパクトを与え，構造化設計を定着させるに至った。

さて，1990 年代に入ってから急速に関心を持たれるようになったオブジェクト指向プログラミング（Object Oriented Programming）は，それまでのプログラミング・スタイルに対してどのようなインパクトを与えたのであろうか。

オブジェクト指向プログラミングも，構造化プログラミングと同じように源流は古い。しかし，Smalltalk-80 によってオブジェクト指向プログラミングは実用化の域に達したものの，1980 年代においてはグラフィカル・ユーザー・インターフェイス（Graphical User Interface）を持つ基本ソフトウェアの革新に及ぼした影響を除き，オブジェクト指向自体が大きく取り上げられることはあまりなかった。Apple Macintosh のグラフィカル・ユーザー・インターフェイスに，オブジェクト指向プログラミングの影響があったことは特筆すべきであるが，経営情報システムの分野に関して言えば，1980 年代は構造化プログラミングを定着させる段階にとどまっていたと言わざるを得ない。

オブジェクト指向プログラミングによってもたらされる効果については，

様々な観点からの指摘がなされているが，最も頻繁に取り上げられるのはやはりグラフィカル・ユーザー・インターフェイス開発との関連であった。グラフィカル・ユーザー・インターフェイスはイベント駆動型（event driven）ないしメッセージ駆動型と呼ばれるプログラミング技法と深い関連を有している。ユーザーがグラフィック・オブジェクトを操作することによってプログラムが呼び出され，必要な処理が行われるのであるが，その際のグラフィックスは単なるデータではなく，一連の処理と結びつけられているのである。多様なグラフィック・オブジェクトは根底において系統性を有しているので，それらを分類する，つまりクラスを構築することによって，プログラミングの効率性が高まることになる。

従来のプログラミングがプログラマー主導型のプログラミングだったのに対し，イベント駆動型プログラミングはユーザー主導型のプログラミングである。プログラマー主導型のプログラムではデータ入力から処理を経て情報出力に至るメインの流れが存在するのであるが，ユーザー主導型のプログラムはユーザーのマウスやキーボード操作による要求に応じた短い処理の集合体とならざるを得ない。

さて，ユーザー・インターフェイス以外の面で経営情報システムにとってオブジェクト指向プログラミングが重要になるのはどのような側面であろうか。

一般的に指摘されているのは，カプセル化（encapsulation）による，従来のライブラリの欠点の補正である。そこから，オブジェクト指向プログラミングを採り入れる際には，クラス・ライブラリの構築が非常に重要なキーポイントになると言うことができる。

さらに，クラス・ライブラリを利用する際には，従来のライブラリとは異なり，クラスの継承（inheritance）を活用することになる。このような形で，プログラムの再利用を促進し，システム開発の効率を高めるのがオブジェク

ト指向プログラミングの狙いとされているのである。

経営情報システムの開発に際しては，ビジネス・アプリケーションのフレームワークを再検討した上で，クラス・ライブラリを構築し，プログラムの再利用を図っていくことになる。

しかし，実際のところは，クラスからクラスを導出する作業を繰り返すプログラミングは，従来のプログラミング・スタイルとはかけ離れているため，従来のプログラミング・スタイルに慣れたプログラマーにとっては必ずしも習得が容易でなく，プログラミング教育についても見直さなければならない時期に来ているように思われる。

オブジェクト指向プログラミングをサポートしている言語[8]の中で標準化が進んでいるのはC++であるため，C++が経営情報システム・モデルの開発用のプログラミング言語として，現時点では最良であると言える。オブジェクト指向をサポートしていても標準化されていないプログラミング言語は処理系依存にならざるを得ず，移植性を重視したシステム開発の場合は問題が残る。標準化された他の言語でオブジェクト指向への対応が進むようであれば，C++以外のプログラミング言語も有力視されるが，現状ではC++が一歩リードしていることは否定できない。

この理由は，C++がCを仕様に含んでいることであり，膨大な数のCプログラマーがC++プログラマーになる可能性を含んでいるからである。また，インターネットとともに重視されつつあるSun MicrosystemsのJAVAもC++の流れを汲んでいるため，C++からJAVAへの移行は比較的容易であり，現時点でのC++の重要性は極めて高いと言うことができる。

4. 基本ソフトウェア依存の領域

プログラミング言語におけるユーザー・インターフェイスのサポートは，

標準的なコンソール入出力では現状にそぐわないため，業界標準を検討せざるを得ない。

BASIC インタプリタの場合は，パーソナル・コンピュータのハードウェアに合わせて，画面制御文が言語仕様に組み込まれていた。この仕様を受け継いだ MS-DOS の BASIC 開発環境も画面制御文を組み込んでおり，ユーザー・インターフェイスの構築が可能であった。日本ではパーソナル・コンピュータ用の BASIC として，日本電気の N_{88}-BASIC（それに次ぐものとして富士通の F-BASIC）が普及していたため，N_{88}-BASIC の画面制御文が 1980 年代におけるユーザー・インターフェイス開発の業界標準として定着した[9]。

そのため，C の開発環境においても，BASIC の画面制御に近いライブラリが提供され，用いられた。パーソナル・コンピュータ用の C では（当然のことながら）標準入出力が MS-DOS の入出力であるため，MS-DOS のコマンドライン用ユーティリティも作成できる利点を持っており，インライン・アセンブラを組み込むことによって，BASIC よりも柔軟性の高いプログラミングを可能にした。

他方，COBOL においては，例えば Micro Focus COBOL のように，パーソナル・コンピュータ用の開発環境によっては，処理系依存ではあるがキーボード，ディスプレイ入出力の拡張が行われており，システム開発上かなり有用であった。

このように，基本ソフトウェアのユーザー・インターフェイスがコマンドライン入出力のみであった頃は，ユーザー・インターフェイスを処理系依存で提供していたのであった。

経営情報システムの基本となる帳票処理のユーザー・インターフェイスにおいては，伝票形式のトランザクション処理を固定画面による穴埋め方式で行うのが一般的であり，テキスト画面を制御すれば足りるので，上記のライブラリであれば，それらの入出力画面を構築することができた。

しかし，経営情報の技術を積極的に事業展開の手段として利用しようとする場合には，ユーザー・インターフェイスの再検討が特に重要となる。すなわち，従来型インターフェイスが社内の専門オペレーターによる操作を想定したものであったのに比べて，消費者一般を想定した，よりユーザー指向のインターフェイスが求められるようになるからである。

販売管理システムにおける商品情報なども，視覚的に画像データを扱わざるを得なくなる。情報技術の戦略的利用を考える場合には，従来型の入力効率重視型のユーザー・インターフェイスでは対応しきれなくなっているわけである。マルチメディアに対応したユーザー・インターフェイスを持つ経営情報システムを開発する場合には，従来型のプログラミング・スタイルではやはり限界がある。

1995年に安価なネットワーク・コンピュータが，いわゆる500ドル・コンピュータとして話題になったが，そうした軽装備のコンピュータを活用して，コンピュータのハードウェア資源を小規模に押さえた経営情報システムを構築する試みが始まっており，その動向が注目されている。そして，WWWブラウザ対応のプログラミング言語として，現状において最有力の言語は前述のJAVAであり，イントラネット（intranet）の基本的な開発環境として期待されている。

しかし，現時点ではJAVAは開発の途上にあり，JAVAを基本ツールとするネットワーク・コンピュータのプラットフォームも整備されていない。現段階ではパーソナル・コンピュータの基本ソフトウェアであるMicrosoft Windowsを，プラットフォームの業界標準と考えざるを得ない。JAVA対応のWWWブラウザはWindows用にも開発されているので，当面はWindowsをパーソナル・コンピュータの業界標準的な基本ソフトウェアと考えて差し支えないだろう。

Windowsを業界標準とした場合，Windows対応のグラフィカル・ユー

ザー・インターフェイスを開発する環境を問題にしなければならない。グラフィカル・ユーザー・インターフェイス開発環境そのものはコマンドライン・コンパイラ（この場合はCコンパイラ）の形で提供されることが多い。Windowsのプログラム開発においても，かつてはコマンドライン・コンパイラしか，開発環境が提供されていない時期があった。

パーソナル・コンピュータのプログラム開発環境は，もともとBASICインタプリタのような統合的な環境であったが，コンパイラに関しても統合開発環境（Integrated Development Environment）が開発され，多くのユーザーを獲得していた。そして，グラフィカル・ユーザー・インターフェイスにおいてはビジュアルな開発環境が普及するに至っている。マウス操作によってユーザー・インターフェイスを設計するシステムや，ダイアログボックスによる対話形式でプログラムの基本仕様を決定するとソース・コードが自動生成されるツール[10]を有する開発環境を，ビジュアルな開発環境ということができる。

グラフィカル・ユーザー・インターフェイスを持つ基本ソフトウェアに関して共通に言えることだが，標準化されたプログラミング言語ではユーザー・インターフェイスを開発できないため，大幅な言語拡張やライブラリの提供が必要となる。CでWindowsのプログラムを開発する場合も，Windowsのライブラリを組み込んでプログラミングを行う。

WindowsのAPI（Application Programming Interface）関数はWindows自体の基本的な機能を含んでおり，関数の数はマルチメディアやネットワーク関連を含めると3000以上ある。当然ながらプログラマーの負担は増大し，プログラム開発の効率は悪化せざるを得ない。

また，Windowsプログラムには，標準化されたプログラミング言語にはないプログラム・リソースが必要となる。主なリソースには，アクセラレータテーブル，ビットマップ，カーソル，ダイアログボックス，アイコン，メ

ニュー，ストリングテーブル，ツールバー・リソース，バージョン情報などがある。リソースに関しては，別のエディタで作成し，開発プロジェクトに追加する。

C++ による Windows のクラス・ライブラリの利用がこうした API 関数の増大に対処する方法として提唱されている。他方で，Microsoft Visual Basic や Borland Delphi のような Windows API の使用を前提とはしないプログラム開発環境もある[11]。

機能が巨大化した基本ソフトウェアにおいては，こうした方策によってプログラミングの容易化が図られている。

グラフィカル・ユーザー・インターフェイスを持つ基本ソフトウェアの普及により，ユーザー・インターフェイスの開発は基本ソフトウェアに依存せざるを得ない状況となったが，基本ソフトウェアに依存するもうひとつの重要な課題として，データベースの開発があげられる。

データベースの開発手段には，汎用プログラミング言語の開発環境と，DBMS の二通りがある。両者の間に連携システムがあれば，汎用プログラミング言語によって開発されたシステムから，DBMS のデータベースを利用することが可能になる。しかし，パーソナル・コンピュータの場合は，両者の連携が十分であるとは必ずしも言える状況ではなかった。パーソナル・コンピュータをデータベース・サーバーとして利用する試みは，パーソナル・コンピュータの近年の著しい性能向上に伴ってようやく可能になったものであり，1980 年代まではパーソナルなレベルでの汎用プログラミング言語によるシステム開発と，DBMS によるデータベース開発とは切り離して考えるのが一般的であったと言える。

データベースに関する両者の連携は，パーソナル・コンピュータ相互の

ネットワーク化の動向に大きく依存しており，パーソナル・コンピュータ用のネットワークOSの普及が，重要な鍵を握っていた。例えば，Windowsの場合は，Windows NT対応のSQL ServerをLANで利用することにより，パーソナル・コンピュータによるクライアント/サーバー型システムの開発が可能となった。

パーソナル・コンピュータによるLANの普及に伴って，SQLを利用できるプログラム開発環境が一般的になった。Windowsの場合，Microsoft ODBC（Open Database Connectivity）やBorland DBE（Database Engine）によって，外部データベースへのアクセスが容易になっている点も重要である。

1980年代までは，汎用プログラミング言語でパーソナル・コンピュータのアプリケーション・ソフトウェアを開発する際には，データベースを個別に構築するのが一般的であったが，1990年代においては，SQLなどによる外部データベースの利用によって，システム開発を効率化するとともに，データベースの有効利用を促進する方向性が明確になったと言うことができる。

このような動向の中で，パーソナル・コンピュータ用のDBMSはどのように位置づけられるのであろうか。

1980年代のパーソナル・コンピュータ用リレーショナルDBMSはアプリケーション開発環境として位置づけられており，製品の価格も高かったため，専ら業務用として位置づけられていた。代わりにリレーショナル機能を持たないカード型データベース（管理システム）というソフトウェアが低価格で販売され，パーソナルなデータベース構築に活用されていた。

しかし，カード型データベースはデータベース作成機能を持つスプレッドシートと，低価格化したリレーショナルDBMSによって駆逐され，現在ではほとんど姿を消している。

低価格化したリレーショナルDBMSによって，カード型データベースの場合のようなパーソナルなデータベースを構築することも可能であるが，そ

の程度の利用であればスプレッドシートの方が手軽であり，やはり付属の開発用言語を用いてアプリケーション・ソフトウェアを開発するのがリレーショナルDBMSの本格的な活用法であると言える。

しかし，汎用プログラミング言語で外部データベースに容易にアクセスすることができるようになると，DBMSとの機能面での重複を生じ，開発環境を厳選する必要性が生じてくる。汎用プログラミング言語によるデータベース開発はDBMSによるそれと比べて一般的に効率が低いとされているので，スタンド・アローンのデータベース開発に関しては，DBMSが適していると言えるが，LAN環境でサーバーのデータベースにアクセスするアプリケーション・フトウェアの開発に関しては，汎用プログラミング言語の開発環境が適していると考えられるのである。少なくとも標準化された言語を用いて開発を行える点で，教育・研究機関における経営情報システム・モデルの開発に際しては，汎用プログラミング言語からSQLを用いる方が望ましいと指摘できよう。

5. 結　び

経営情報システムのモデルを開発するための，パーソナルな開発環境の現状について結論をまとめると，以下のような点を指摘できる。

まず，パーソナル・コンピュータ上で，経営情報システムのモデルを汎用プログラミング言語を用いて開発する場合には，Cをベースとしてオブジェクト指向プログラミングをサポートしているC++が現段階では最有力であると言える。メインフレームの事務計算では主流のCOBOLは，今のところオブジェクト指向プログラミングをサポートしていない上，パーソナル・コンピュータでの普及度が高くないため，C++に比べて優先度が低くならざるを得ない[12]。

次に，グラフィカル・ユーザー・インターフェイスに関しては業界標準に依存せざるを得ない点が，教育・研究の観点からはややネックになっている。しかし，ユーザー・インターフェイスの不十分なアプリケーション・ソフトウェアは明らかに現状にそぐわないだけに，経営情報システムのオペレーションに関してはグラフィカル・ユーザー・インターフェイスを採用すべきであろう。その場合には，API を利用するか，クラス・ライブラリを利用するかが問題となるが，基本的にはクラス・ライブラリの方が有力であろう。

そして，パーソナル・コンピュータの LAN 環境が普及してきたことによって，サーバー・コンピュータのデータベースに容易にアクセスすることができるようになり，SQL などによる外部データベースの操作をサポートするプログラム開発環境が一般的になった。このことにより，ネットワークを利用したクライアント / サーバー型経営情報システムのモデル開発がより容易になったと言えよう。また，パーソナルなレベルで開発したシステムの中規模の経営情報システムへのアップサイジングも可能になったため，パーソナル・コンピュータの経営情報システムにおいて重要な役割を果たすようになりつつある。

経営情報システム・モデルの開発に際してオブジェクト指向を採り入れた場合は，これまでの COBOL によるメインフレーム上での伝統的なシステム開発とはかなり異なった開発スタイルとなる。ビジネス・アプリケーションのフレームワークを確立するためにシステム分析のあり方も変わり，再利用可能なライブラリの構築に重点が置かれ，実際の情報システム開発では，ライブラリをカスタマイズする作業が中心となるであろう。

〔注〕

1） なお，本稿に記載の社名および製品名は各社の商標または登録商標である。

2） 経営情報システムをコンピュータの規模と利用レベルの視点から体系化し，個人情報システムの特徴を論じた文献として，以下のものがある。

島田達巳，高原康彦編著『経営情報システム』日科技連，1993 年，第 2 章およ

び第4章。

3) 1960年にCODASYLで仕様がまとめられた。

4) ANSI '85より前のCOBOLでは，その他に構造化プログラミングが行えないといった欠点が指摘されていたが，ANSI '85（ならびにJIS '92）COBOLにおける改訂で，こうした問題は解決している。

5) パーソナル・コンピュータ用のBASICはMicrosoft系のBASICが主流となったが，BASICの開発者自身によるTrue BASICという開発環境もあり，プログラミングの入門に適する言語であることが強調されている。

John G. Kemeny & Thomas E. Kurtz, *Back to BASIC,* Addison-Wesley Publishing, 1985.（松田健生訳，市川新解説『バック・トゥ・BASIC』啓学出版，1990年。）

6) 1980年代後半におけるBorland Turbo Cが，その代表例と言える。

7) なお，Cを経営情報システム構築のメインの言語として採用した場合には，BCD演算を考慮に入れておく必要がある。COBOLの場合は10進演算が基本となっているため，利息計算などのケースで2進10進変換に伴う誤差を生じる可能性を考えなくても済むが，Cではこの点に留意しなければならない。

8) オブジェクト指向プログラミングをサポートする言語の比較に関しては，例えば以下の文献が参考になる。

Timothy A. Budd, *An Introduction to Object-Oriented Programming,* Addison-Wesley Publishing, 1991.（羽部正義訳『オブジェクト指向プログラミング入門』トッパン，1992年。）

9) ただし，BASICインタプリタは処理速度の面で実務的には不評だった。

10) Microsoftはウィザード（Wizard），Borlandはエキスパート（Expert）などと称していて，決まった用語はない。両者とも自社のC++クラス・ライブラリに基づくソース・コードを生成する。

11) 両者ともフォームやコントロールをマウス操作で設計していくビジュアルな開発環境である。Visual Basicはフォームやコントロールのプロパティをビジュアルな方法で設定した上で，ソース・コードを記述するスタイルを取っている。それに対して，Delphiの場合はPascalをC++風に拡張したObject PascalにWindows用のクラス・ライブラリを組み込んだ形になっており，フォームの設計結果をソース・コードとして生成する。

12) なお，BCD演算を使用する必要がある場合は，C++のクラス・ライブラリを構築するか，市販品を利用することになろう。また，Visual BasicとDelphiはともにcurrency型と称する金額計算用の固定小数点型をサポートしているため，必要に応じて使用することができる。currency型は，Visual BasicではWindows 3.0対応の最初のバージョンからサポートされ続けている。DelphiではWindows 95

対応のバージョン 2.0 からサポートされた。