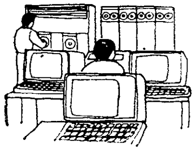
特別論説



情報処理最前線

オブジェクト指向プログラミングの利用価値†

久世和資††

1. はじめに

最近，ソフトウェアは，ますます，多種多様化および巨大化している．この一つの要因として，パーソナル・コンピュータやワークステーションの低価格化と高性能化による普及が考えられる．ソフトウェアの生産性向上のために，これまで各種の手法や技法が考察され実践されてきたが，生産性を飛躍的に向上するものはなかった[1]．そこで，ここ数年来，注目を集めているのがオブジェクト指向プログラミングである．果たしてオブジェクト指向技術が，ソフトウェアの開発に有効であるのかどうか，解説したい．

2. ソフトウェアの現状

まず，現在のソフトウェアの状況を整理してみる．ソフトウェアの分類方法はいろいろあるが，ソフトウェアを作成する人に要求されるスキルのレベルを基準に分類する．

例として DOS 上で動作するスプレッドシートの C++ による開発をとりあげる．ソフトウェアとして必要なものを，計算機よりのシステムプログラムから，エンドユーザの方向に考えてみる．まず，DOS を作成しなければならない．次に，その上で C++ を使ってプログラミングするには，C++コンパイラ，エディタ，デバッガ，リンカなどが必要である．C++ の場合，スプレッドシート用のクラスライブラリと，それを使用するプログラムに分類できる．

スプレッドシートは，エンドユーザよりのソフトウェアであるが，本当のエンドユーザには容易に使いこなせない．そこで市販のスプレッドシートには，マクロ機能が用意されている．マクロの記述により，よく使用する一連のキー入力列を，一つのキーに割り当てたり，カーソルの動きを制御したりできる．ユーザが，機能拡張のためにマクロでプログラミングすることもあるが，ソフトウェア製品として特定業務用のツールや家計簿プログラムが，マクロで開発されている(図-1)．

これらのソフトウェアの開発には，何種類かのプログラマが登場する．一人のプログラマが複数種類のプログラミングを同時にすることもあるが，一般には，それぞれが別のレベルである．このレベルは，プログラマのプログラミング技術や能力に対応している．オペレーティングシステムやコンパイラなどのシステムプログラムの開発は，高度なプログラミングの技術が要求される．これに対して，特定のアプリケーション用の C++ クラスライブラリは，ハードウェアやオペレーティングシステムに独立な分だけ開発が容易になっている．また，スプレッドシートのマクロ機能は，エンドユーザにとって使用するのは難しいが，C++ に比べれば，言語仕様も単純で，機能もスプレッドシートが前提のものに限られるので，より容易にプログラミングできる．コンパイラから，スプレッドシート，マクロによる特定アプリケーションと進むにしたがって，プログラミングの知識は，少くてすむ．しかし，その反面，アプリケーション特有の専門知識は，逆に要求され，アプリケーションの種類も増える．

ソフトウェアの生産性向上には，システムよりもアプリケーション自体のプログラミングをより単純にするほうが効果的である．これは，オブジェクト指向技術にもあてはまる．オブジェクト指向の導入によりアプリケーション開発が，単純になれば，プログラミングの知識は少ないが，アプリケーション分野の知識は豊富にある多人数の

---

† Advantages of Object-Oriented Programming by Kazushi KUSE (IBM Research, Tokyo Research Laboratory).

†† 日本アイ・ビー・エム東京基礎研究所

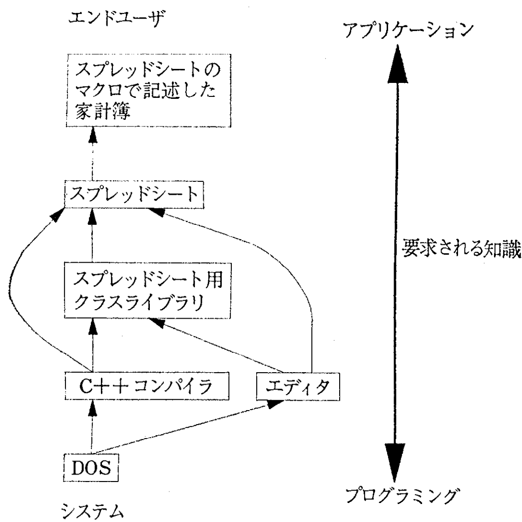


図-1　ソフトウェアの階層化

プログラマによって，幅広いアプリケーション開発が可能となる．ただ，実用のアプリケーションとなると，実行効率などの点から，オブジェクト指向をささえる基盤システム技術も重要になってくる．そこで，まず，オブジェクト指向の特徴を簡単に紹介した後，アプリケーション開発および，それを支える基盤システム技術の両面から，オブジェクト指向技術について説明する．

3. オブジェクト指向技術の特徴

オブジェクト指向プログラミングは，これまでの主流である構造化プログラミング (structured programming)[2] に代わる技術として数々の新しい概念をソフトウェア開発に導入した．それらの概念は，抽象データ型 (abstract data typing)，カプセル化 (encapsulation)，メッセージ通信 (messaging) および継承 (inheritance) である．以下，これらの概念の説明も含めて，オブジェクト指向技術の特徴について述べる．

3.1 実世界のオブジェクトによる自然なモデル化

現実世界に存在するオブジェクトは，状態をもち，外からの刺激に反応し，種々のふるまい (behavior) をする．この実世界のオブジェクトが，そのまま，オブジェクト指向のオブジェクトに対応する．したがって，複数のオブジェクトの相互作用として，アプリケーションを自然にモデル化できる．

たとえば，銀行業務のアプリケーションの場合，普通預金，定期預金，当座預金といった現実に存在するオブジェクトが，そのまま，アプリケーションのモデル化に利用できる．また，実際のインプリメンテーションも，そのモデルにしたがって，素直にできる．

構造化プログラミングの場合，アプリケーションのモデル化は，データフロー図や制御フロー図などを用いて行い，それらのモデルから手続き型言語で，かなりモデルとは構造の異なるインプリメンテーションをする必要がある．これでは，アプリケーションの要求定義，設計のフェーズから，コーディングのフェーズへの連続的な移行は望めない．

これに対して，オブジェクト指向では，先にも述べたように，実世界を自然にオブジェクトでモデル化できるので，オブジェクトという首尾一貫した概念でソフトウェアの開発が実現できる．

3.2 ソフトウェアの開発効率の向上

オブジェクト指向技術に導入された各種の概念は，ソフトウェア開発効率の向上に役立っている．

3.2.1 カプセル化による高機能部品の実現

オブジェクトには，複数のデータと，それらに関連する手続きであるメソッド (method) 群がひとまとめにされる[3]．プログラマは，オブジェクトが提供するインタフェースだけ使用してプログラムを作成できる．オブジェクト中のデータやメソッドの詳細なインプリメンテーションを意識する必要はない．

社員情報のオブジェクトを例にして説明する．このオブジェクト中には，社員に関する情報として，名前，生年月日，給料のデータがある．また，これらのデータを社員情報として，取り出すメソッドと，年齢を計算するメソッドがあるとする．オブジェクトのインタフェースとして提供するのは，これらのメソッドの呼び出し方法だけでよい．オブジェクトの使用者には，各データの型や構造および，各メソッドの詳細を隠すことができる．また，それぞれのメソッドを実行するための補助的なデータや手続きも，オブジェクトの外に見せないようにできる (図-2)．

オブジェクトの機能を実現するためのデータとメソッドを，オブジェクトという部品の中に集約できることが重要である．オブジェクトを使用せ

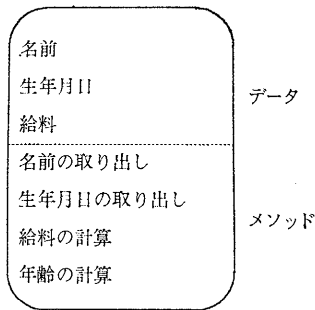


図-2 社員情報用オブジェクトの例

ずに，一般の関数群と大域データで，同等の機能をユーザに提供することもできる．しかし，これでは，ユーザが意識する必要のないデータや手続きまで同じプログラムのスコープ中に散在してしまうし，単一のインタフェースとして手続き群をまとめることもできない．

関連するデータとメソッドをオブジェクト単位にカプセル化することにより，ソフトウェアの作成や更新時に着目するデータや手続きが局所化され，作業効率が高くなる．

3.2.2 クラスと継承によるコードの再利用

同じふるまいをするオブジェクトのひな型として用いられるのがクラス (class) である．たとえば，先の社員情報のオブジェクトの場合，実際の社員に対応して，複数のオブジェクトが存在する．各オブジェクトのデータの値は，それぞれの社員に対応して異なるものとなっている．社員情報のクラスは，複数の社員情報オブジェクトのひな型として用いられ，オブジェクトと同一の構造をもつ．

すでに存在するクラスから，少し機能の異なるクラスを作成するのに利用するのが継承である．継承する元のクラスを，スーパクラス (super class)，継承によりできた新たなクラスをサブクラス (subclass) と呼ぶ．たとえば，社員クラスと少し機能の異なる管理職社員のクラスは，社員クラスから継承を用いて簡単に作成できる．この場合，スーパクラスは，社員クラス，サブクラスは，管理職社員クラスとなる (**図-3**)．

サブクラスには，スーパクラスを親として継承することと，スーパクラスとの差異だけを記述すればよい．管理職社員クラスの場合，管理職手当用のデータが新たに加わり，給料の計算用メソッドも変更される．このとき，これらの変更部分だけサブクラスに記述すれば，スーパクラスの残りの機能は自動的に引き継がれる．

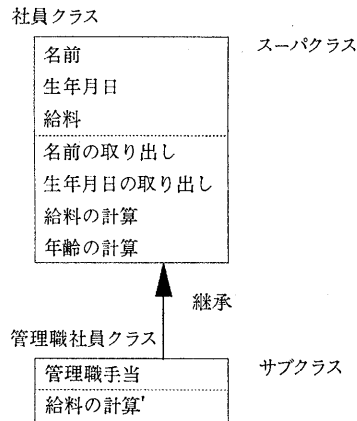


図-3 継承による管理職社員クラスの作成例

継承を利用することにより，重複する機能をスーパクラスにまとめることができ，コード量を少なくできる．前述の銀行業務の場合，普通預金，定期預金，当座預金のスーパクラスとして預金クラスを作れば，それらに共通する機能を，預金クラスにまとめることができ，コード量が減る．さらに，同一コードが，分散することがなく，ソフトウェアの保守や更新時の機能拡張が容易になる．

3.2.3 クラスライブラリの利用

ソフトウェア部品として利用できるクラスの集合をクラスライブラリと呼ぶ[4),5)]．クラスライブラリは，一般にクラスの階層構造をもっている．Cなどの関数ライブラリに比べ，もう1レベル，クラスという単位のスコープが増えるので，部品としての抽象度が高くなっている．また，クラスライブラリの階層構造を見ると，ライブラリ全体の構成が理解でき，部品として適当なクラスを見つけやすくなっている．さらに，前述の継承を利用して，クラスライブラリ中のクラスを修正して利用することもできる．

Xウィンド用の C++ クラスライブラリとして，よく利用されているのが，InterViews である[16)]．この中には，約150のクラスが含まれている．その中の図形の描画用のクラス群は，図形の性質に基づき自然に継承で表現されている．つまり，抽象度の高い図形のクラスを継承して，より

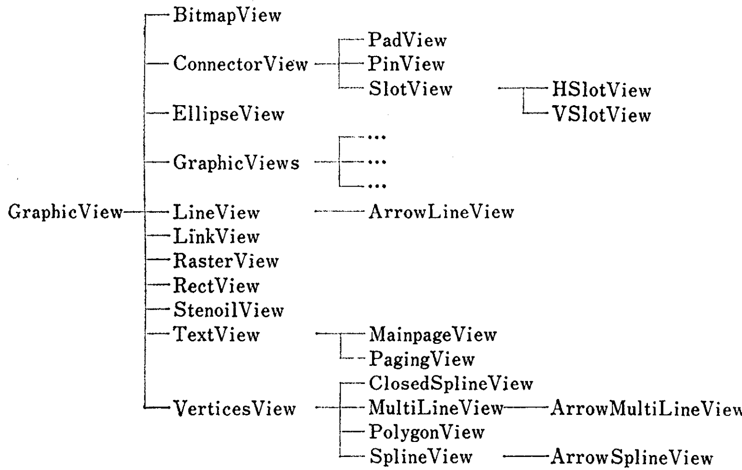


図-4　InterViews クラスライブラリ（一部）

具体的な図形クラスが定義されている（**図-4**）．

InterViews を利用して作成されたアプリケーションは数多くある．その中には，描画用ツールの Idraw や，C++ のクラスブラウザの Iclass などが含まれる．Iclass は，クラス名リスト，クラスインタフェースのテキスト，および，階層構造のグラフが連動するブラウザである．これは，C++ で約 1000 行というサイズで実現されているが，InterViews を使用しなければ同等のものを記述するのに，数倍のコード量が必要であろう．

クラスライブラリの形にすれば，どんなものでも部品として再利用性が向上するというものではない．やはり，使い勝手のよいクラスライブラリを作るには，そのデザインが最も重要である[11),34),39)]．

4. アプリケーション開発における オブジェクト指向技術

ソフトウェア開発におけるアプリケーションよりのプログラマにとってメリットのあるオブジェクト指向技術について説明する．ここでは，フレームワークと，アプリケーション・ビルダについて解説する．

4.1 フレームワーク

前述したように，オブジェクト指向でアプリケーションを設計するときは，実世界をオブジェクトでモデル化する．たとえば，ユーザインタフェースの場合，メニュ，ウィンド，スクロールバーといった実際に目に見えるものをオブジェクトとみなしモデル化する．同種類のオブジェクトをまとめてクラスとする．クラスの中でも，類似性のあるものは，継承を使ってさらに抽象化する．ユーザインタフェースの例では，テキストウィンド，リストウィンド，および，グラフィックウィンドから，より一般的なウィンドクラスを定義することに対応する．

実際のアプリケーションの構築では，具体的なクラスのオブジェクトを必要数だけ生成して使用する．さらに継承を利用して，より具体的なクラスとそのオブジェクトを生成する場合もある．アプリケーションでは，各種のクラスから生成された複数のオブジェクトが，メッセージで交信しながら全体として動作する．

ここで問題になるのが，実際にオブジェクト同士を，いかに関連付ければアプリケーションができあがるのかが不明瞭なことである．つまり，どのような種類のオブジェクトを，どのタイミングでどれだけ生成して，それらの間のメッセージを，いかに結ぶかといったことである．クラスライブラリ中のクラスの種類を，いかに豊富にそろえても，それらの使い方が明らかでないと効率的にはアプリケーションは作成できない．

そこで登場したのが，フレームワークである．フレームワークは，再利用可能なアプリケーションのひな型である[19)]．これにより，ソフトウェア部品であるクラスやクラスライブラリの再利用だけではなく，それらの使い方およびアプリケーションのデザインを再利用することができる．具体的なアプリケーションの構築の際は，標準のひな型であるフレームワークからの差異だけを意識すればよく，その部分を作成すれば，同種類のアプリケーションが簡単に構築できる．

フレームワークの具体例としては，MVC[17)]，MacApp[18)]，Unidraw[36)] などがある．ここでは，MacApp を簡単に紹介する．

MacApp は，Apple の Macintosh 上のアプリケーションプログラムの一貫性と生産性の向上のために利用されているフレームワークである．第1版が 1986 年にリリースされて以来，第2版，

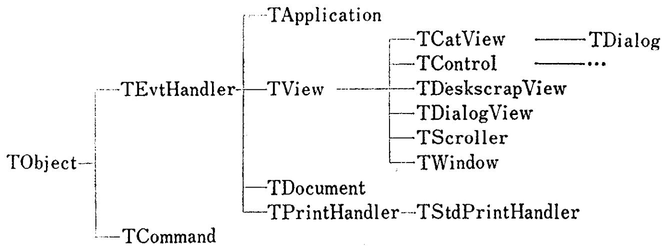


図-5 MacApp 第2版のクラス階層（一部）

第3版とそれぞれ，1988年，1992年にリリースされ，使用されている．

MacApp の主な抽象クラス（abstract class）には，TApplication, TView, TDocument, TPrint Handler の4つがある（**図-5**）．これらのクラスのオブジェクトは，直接，生成せずに，サブクラスを作ってそのオブジェクトを利用することになっている．MacApp のアプリケーションは，TApplication のサブクラスを一つもつ．アプリケーションのウィンドドの表示形式ごとに TView のサブクラスを作る．TDocument は，外部データファイルに対応するもので，その種類だけのサブクラスが必要となる．TCommand は，アプリケーションに対するユーザのアクションに対応し，その数だけサブクラスを用意する．

MacApp を利用することにより，プログラマは，アプリケーションを統一的に作成することができる．それと同時にフレームワークに組み込み部品を系統的に効率よく利用できるので，開発作業も大幅に軽減できる．

4.2 アプリケーション・ビルダ

アプリケーション・ビルダは，高度なプログラミング技術をもたないユーザでも，特定分野の専門知識を利用して，容易にアプリケーションが構築できることを目的とするツールである．つまり，従来のプログラム言語によるコーディング作業をできるだけ省略することにより，アプリケーション作成者が，プログラミングよりも，アプリケーション自体により集中できることを目ざしている．

アプリケーション・ビルダが提供する機能は，新たな部品の作成支援と，作成した部品および既存の部品の組み合わせ（composition）によるアプリケーション構築である．この中で，アプリケーションの部品化と再利用化の機能を提供するのに，オブジェクト指向は不可欠な技術といえる．

また，前述のフレームワークとアプリケーション・ビルダが，うまく融合すれば，より強力なプログラミング支援環境となる[14]．

アプリケーション・ビルダは，GUI (Graphical User Interface) の分野では，GUI ビルダまたは WYSIWYG エディタとして実用のものがいくつかある．NeXTStep の GUI ビルダや InterViews 用の Ibuild[37] は，その例である．

オブジェクト指向ベースのアプリケーション・ビルダとしては，Smalltalk/V を使った DIGITALK の PARTS (Parts Assembly and Reuse Tool Set) がある．Smalltalk や C++ では，クラスを中心にプログラムを作成する．クラス中にメソッドを記述すると，そのクラスのすべてのオブジェクトのふるまいが，それによって指定できる．これに対して，PARTS は，オブジェクト中心のプログラミングを支援する．つまり，あるオブジェクトのふるまいを記述しても，その内容は，そのオブジェクトのみに有効である．

PARTS では，プログラミングのスキルの少ないプログラマでも，ソフトウェア部品であるオブジェクトを，ビジュアル・インタフェースを通して単純に組み合わせて，より高度な部品やアプリケーションを構築できるようになっている．具体的には，オブジェクトを図で表示し，各データに値をセットしたり，異なるオブジェクトのメソッド間を線で結ぶことにより，メッセージ通信が指定できる（**図-6**）．

PARTS では，確かに単純なアプリケーションは，短時間で作成できそうであるが，より複雑なものまでうまく対応できるかは，検討の余地がありそうである．大きなアプリケーションに対応するには，アプリケーション特有の部品をうまく用意することが重要である．また，PARTS は，アプリケーション分野を規定しない汎用のビルダであるが，少しアプリケーション分野を限定することも有効であると思われる．たとえば，Borland の ObjectVision は，スプレッドシートと GUI が関係するアプリケーションに限定されるが，アプリケーションの作成は，PARTS に比べれば，かなり単純である．この種のツールでは，汎用性と

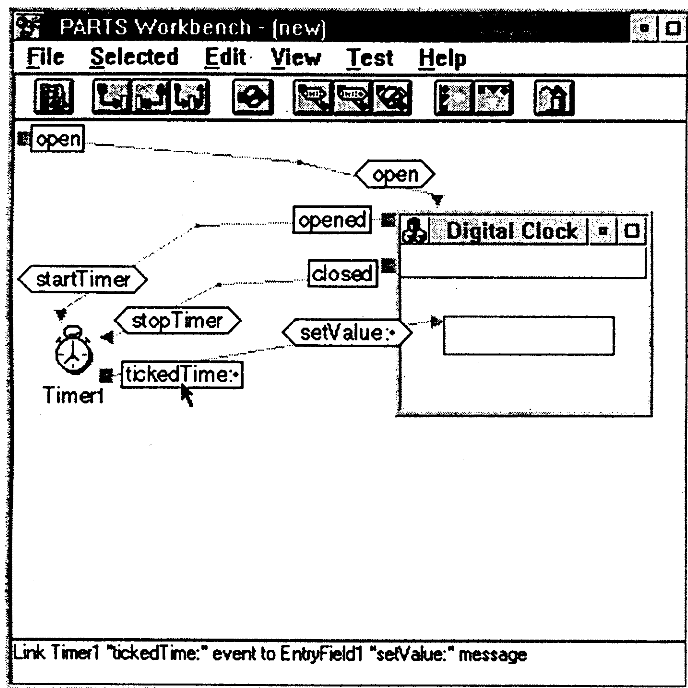


図-6 PARTS による部品オブジェクトの結合例

操作性は，相反する要素なので，適当な切り口を見つけるのが重要である．

5. システム基盤におけるオブジェクト指向技術

ここでは，アプリケーション開発を，下から支えるオブジェクト指向技術として，分散オブジェクト，永続オブジェクト，および，コンパイラとプログラミング環境について述べる．これらの技術は，アプリケーションに直接，依存せず，汎用であることからシステムよりに位置付けた．しかし，これらの必要性は，アプリケーション開発からきたものも少なくないし，その恩恵は，アプリケーション開発におよぶものである．

5.1 分散オブジェクト

はじめに述べたように，ワークステーションの低価格化と高性能化により，現実の業務やシステムに，ネットワークで結ばれた複数台の異種マシンが，急速に導入されてきている．このような環境で動作するアプリケーションを，手軽に構築する技術も重要になってくる．アプリケーション開発のレベルでは，分散の概念を，ユーザに陽に意識させるかどうかを含めてフレームワークやアプリケーション・ビルダに，いかに取り込むかが課題である．それには，まず，システムレベルで，分散を支えるしくみが必要となる．

分散オブジェクトに関する種々のアプローチがあるが，ここでは，OMG (Object Management Group) の CORBA (Common Object Request Broker Architecture) を紹介する[35]．OMG は，異なる複数のベンダ間でのオブジェクト指向技術に関する標準化を目的としている．この中で CORBA は，異種マシンおよび異種 OS 間で，オブジェクトを分散する標準アーキテクチャである．オブジェクトの部品としての再利用性を高めるには，分散の機能とインタオペラビリティが重要となってくる．

CORBA では，クライアントから呼び出しとして，静的呼び出し (static invocation) と動的呼び出し (dynamic invocation) の 2 種類がある．静的呼び出しは，あらかじめ存在するクライアントのスタブとサーバのスタブが対になって実行される．動的呼び出しでは，スタブの生成が動的に起こる．

CORBA の課題は，インタオペラビィリティの実現と，実行効率の改善である．まだ，異なるベンダ間のプロトコルは設計段階であり，具体的な成果は出ていない．CORBA は，もともと粒度の大きなオブジェクト間の通信を仮定していると思われる．このように大きな粒度のオブジェクト間の通信でも，その効率がボトルネックになる可能性はあるので，その改善は必要であろう．

5.2 永続オブジェクトとオブジェクト指向データベース

オブジェクト指向プログラムは，通常，アプリケーションの開始後に，オブジェクトを生成し，アプリケーションの終了時には，全てのオブジェクトが消滅する．しかし，アプリケーションの実行中に生成したオブジェクトのうち，次回のアプリケーションの実行や異なるアプリケーションでも利用したいものは，オブジェクトの形で格納しておきたいという要求がある．アプリケーション

の実行セッションを越えて，生き残るオブジェクトを永続オブジェクト (persistent objects) と呼ぶ.

永続オブジェクト機能をもったオブジェクト指向言語もいくつか開発されているが，通常はこの目的でオブジェクト指向データベースが用いられる．オブジェクト指向データベースは，永続性のほかに，リカバリ，問合わせ言語，複合オブジェクトなどもサポートする[38]．主要なオブジェクト指向データベースは，ONTOS，ObjectStore，Versant などである[12),13]．

オブジェクト指向データベースでは，永続オブジェクトと一般オブジェクト間での一貫したデータモデル，既存のオブジェクト指向言語との親和性，メモリとディスクなどのストーレジ間でのオブジェクトの出入れの効率化，キャッシングなどがポイントとなっている．

まだ，大規模なアプリケーションで，関係データベースをオブジェクト指向データベースで，置きかえてうまくいった例は少ない．今後，前述の使い勝手と実行効率の向上とともに，アプリケーションのフレームワークやビルダに，いかに適合させるかが重要な課題の一つである．ちなみに，Borland の ObjectVision は，外部の関係データベースとアプリケーションを容易に連動させることができる．これは，対象としているスプレッドシートが，関係データベースの表になじみやすいからであろう．

5.3 コンパイラとプログラミング環境

アプリケーション開発は，フレームワークやビルダにより生産性の向上が期待できるが，これと同時にできあがったアプリケーションの実行効率の向上を考える必要がある．これは，部品としての再利用性や汎用性を上げると，それらの部品からできあがったアプリケーションの効率は，逆に悪くなることが予想されるからである．

まず，オブジェクト指向プログラムは，手続き型プログラムに比べて，メソッドとメッセージの動的結合 (dynamic binding) やオブジェクトの生成と消滅にともなうオーバヘッドが問題になっている．動的結合に関しては，コンパイラの最適化をより強力にして，動的結合で記述されたものも静的結合 (static binding) に変換できるものは，変換して実行する必要がある[23)~25]．オブジェクトの生成と消滅に関しては，自動メモリ管理を充実することが重要である．たとえば，C++ の場合，不要になったオブジェクトのネットワークを回収するには，それをたどるようなプログラムを消滅子 (destructor) を使ってプログラマが書く必要がある．これでは，とても再利用性の高い部品の作成は難しい．やはり，メモリ管理は，できるだけプログラマから解放し，システムが行うべきである[29]．

また，実行効率の改善には，アプリケーション開発レベルでの効率のチューニングがより効果があると思われる．アプリケーション開発者のほうが，アプリケーションの特性を生かした最適化が可能であるからだ．ただ，最適化の手段は簡単化する必要がある．何種類かのメモリ管理方式の中から一つ選ぶとか，静的結合に変換できることのヒントを与えるようなものである．ソフトウェアの部品がオブジェクトの形で提供されるときは，コンパイルしなおさずに，上記のようにオブジェクトの動作を変える必要がある．このために，リフレクションを利用することも考えられる[26)~28]．

プログラミング環境としては，大きなクラスライブラリの中から使用したクラスをすばやく見つけ出したり，内部のメソッドの呼び出し関係をたどって，そのクラスのふるまいが容易に理解できるようなクラスブラウザが重要である[21),22]．また，クラス構造や階層構造のために C++ などのコンパイル型のオブジェクト指向言語では，再コンパイル時間の増大が問題になっている[40]．この解決には，インクリメンタルコンパイラやインタプリタなどのツールのサポートが不可欠である[41]．

6. おわりに

本稿では，アプリケーション開発と，システム基盤技術の二つの立場から，オブジェクト指向技術について解説した．その中でも，前者は，ソフトウェア全般的な生産性向上に有効であることを述べた．しかし，現実のコンピュータ環境の急速な変化や，アプリケーションの実行効率を考えると後者のシステム基盤技術もおろそかにできない．これまで，オブジェクト指向言語[6~10]とオブジェクト指向設計法[30)~33]の議論は盛んであったが，この間にはギャップがあったように思う．このギャップを埋める技術として，本稿のフレーム

ワークやアプリケーション・ビルダなどは，有効である．結局，アプリケーション開発やソフトウェアの生産性向上に有効なことは，ソフトウェアの再利用性の促進である．この再利用を支える技術として，オブジェクト指向は，最も有力な技術であるが，まだ不十分である．オブジェクト指向技術の利用価値を高めるためには，ソフトウェアの再利用性向上を目標に，本稿で述べたようなオブジェクト指向をとりまく技術の強化が重要な課題であろう．

謝辞　本稿に対して数多くの貴重なコメントをいただいた日本アイ・ビー・エム東京基礎研究所ワークステーション・システムズ担当の上村務氏に感謝いたします．

参考文献

1) Brooks, F. P. Jr.: No Silver Bullet-Essence and Accidents of Software Engineering, *IEEE Computer*, Vol. 20, No. 4 (Apr. 1987).
2) Yourdon, E. and Constantine, L.: *Structured Design*, Prentice Hall (1979).
3) Snyder, A.: Encapsulation and Inheritance in Object-Oriented Programming Languages, *Proc. of ACM Conf. on Object-Oriented Programming Systems, Languages and Applications*, pp. 38-45. (1986)
4) Palay, A. et al.: The Andrew Toolkit: An Overview, *Proceedings of the 1988 Winter USENIX Technical Conference*, Dallas, Texas pp. 9-21 (Feb. 1988).
5) Gamma, E. and Weinand, A.: ET++—A Portable C++ Class Library for a UNIX Environment, *OOPSLA '90 Tutorial*, ACM (Oct. 1990).
6) Goldberg, A. and Robson, D.: *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, (1983).
7) Stroustrup, B.: *The C++ Programming Language, Second Edition*, Addison-Wesley, (1991).
8) Meyer, B.: Eiffel: The Language, Prentice Hall (1992).
9) Bobrow, D. C., DeMichel, L. G., Gabriel, R. P., Keene, S. E., Kiczales, G. and Moon, D. A.: Common Lisp Object System specification X3 J13, *SIGPLAN Notices*, 23 (Sep. 1988).
10) Shaffert, C., Cooper, T., Bullis, B., Kilian, M. and Wilpot, C.: An Introduction to Trellis/Owl, *Object-Oriented Programming Systems, Languages and Application*, ACM (1986).
11) Coplien, J. O.: *Advanced C++ Programming Styles and Idioms*, Addison-Wesley, (1992).
12) Bretl, R., Maier, D., Otis, A., Penney, J., Schuchardt, B., Stein, J., Williams, E. H. and Williams, M.: The GemStone Data Management System, *Object-Oriented Concepts, Databases, and Applications*, Kim, W. and Lochovsky, F. eds., ACM Press (1989).
13) *Ontos Object Database version 2.0 Developer's Guide*, Ontologic Inc., Burlington (Mass.) (Feb. 1991).
14) Helm, R. and Holland, J. M. and Gangopadhyay, D.: Contracts: Specifying Behavioral Compositions in Object-Oriented Systems, OOPSLA '90, pp. 169-180 (Oct. 1990).
15) Chambers, C., Ungar, D. and Lee, E: An Efficient Implementation of SELF, a Dynamically-Typed Object-Oriented Language Based on Prototypes, *SIGPLAN Notices 24* 10 (Oct. 1989).
16) Linton, M. A, Vlissides J. M, and Calder, P. R.: Composing User Interfaces with InterViews, *IEEE Computer 22*, 2, pp. 8-22 (Feb. 1989).
17) Krasner, C. E. and Pope, S. T.: A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System, *Journal of Object Oriented Programming I*, 3, pp. 26-49 (Aug. 1988).
18) Apple Computer, *MacApp II Programmer's Manual*, Apple Computer, Inc. Cupertino, CA (1989).
19) Deutsch, L. P.: Design Reuse and Frameworks in the Smalltalk-80 Programming System, pp. 55-71, *Software Reusability, Vol II*, ed. Biggerstaff, T. J. and Perlis, A. J., ACM Press (1989).
20) Young, D. A.: *Object-Oriented Programming with C++ and OSF/Motif*, Prentice Hall, Englewood Cliffs, NJ (1992).
21) Goldberg, A.: *Smalltalk-80: The Interactive Programming Environment*, Addison-Wesley, Chapters 4 and 23 (1984).
22) User's Guide to Objectworks Smalltalk Release 4, ParcPlace Systems (1990).
23) Suzuki, N. and Terada, M.: *Creating Efficient Systems for Object-Oriented Languages*, ACM Proceedings of POPL 1983, pp. 200-296 (1983).
24) Deutsch, P.: *Efficient Implementation of the Smalltalk-80 System*, ACM Proceedings of POPL 1983, pp. 297-302 (1983).
25) Johnson, R., Graver, J. and Zurawski, L.: TS: An Optimizing Compiler for Smalltalk, *Object-Oriented Programming Systems, Languages and Applications* (1988).
26) Smith, B. C.: Reflection and Semantics in Lisp, *ACM Symposium on Principles of Programming Languages*, pp. 23-35, ACM Press (1984).
27) Masuhara, H., Matsuoka, S., Watanabe, T. and Yonezawa, A.: Object-Oriented Concurrent Reflective Languages can be Implemented Efficiently, *Object-Oriented Programming Systems, Languages and Applications in 1992*, (Oct. 1992).
28) Yokote, Y.: The Muse Reflective Operating

System: The Concept and Its Implementation, *Object-Oriented Programming Systems, Languages, and Applications* (*OOPSLA*) (1992) To appear.

29) Appel, A.: Simple Generational Garbage Collection and Fast Allocation, *Software: Practice and Experience*, 19(2): pp. 171-183 (Feb. 1989).

30) Booch, G.: *Object-Oriented Design with Applications*, The Benjamin/Cummings Publishing Company, Reading, Mass. (1991).

31) Coad. P. and Yourdon, E.: *Object-Oriented Design*, Yourdon Press Computing Series (1991).

32) Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W.: *Object-Oriented Modeling and Design*, Prentice Hall (1991).

33) Wirfs-Brock, R., Wilkerson, B. and Wiener, L.: *Designing Object-Oriented Software*, Prentice Hall (1990).

34) Johnson, R. and Foote, B.: Designing Reusable Classes, *Journal of Object-Oriented Programming*, I(2): pp. 22-25 (1988).

35) Object Management Group: *Common Object Request Broker Architecture and Specification*, Document 91.12.1, Object Management Group (1991).

36) Vlissides, J. and Linton, M.: Unidraw: A Framework for Building Domain-Specific Graphical Editors, User Interface Software and Technoloies, Conference (1989).

37) Vlissides, J. and Tang, S.: A Unidraw-Based User Interface Builder, User Interface Software and Technologies Conference (1991).

38) 宮崎，川越他：オブジェクト指向データベースシステム特集号，情報処理，Vol. 32, No. 5 (May 1991).

39) 北山：プログラムコードの静的解析によるインヘリタンス使用法の分離とソフトウェア開発の応用，情報処理学会論文誌，Vol. 35, No. 9 (1992).

40) Onodera, T. Kuse, K. and Kamimura, T.: Increasing Safety and Modularity of C Based Objects, TOOLS 3 (1990).

41) Onodera, T.: Reducing Compilation Time by a Compilation Server, Software Practice and Experience (1993) to appear.

(平成4年11月16日受付)

久世　和資（正会員）

1959年生．1982年筑波大学第三学群情報学類卒業．1987年同大学院博士課程工学研究科（電子・情報工学専攻）修了．同年日本アイ・ビー・エム(株)に入社．東京基礎研究所に勤務．現在，同研究所プログラム言語グループ担当．工学博士．プログラム言語，プログラミング環境，オブジェクト指向プログラミングに興味を持つ．1989年本学会研究賞，1990年記念論文賞受賞．日本ソフトウェア科学会，ACM, IEEE各会員．