解　説

オブジェクト指向言語を用いた画像処理への応用†

黒野剛弘†† 永田真啓††

1. はじめに

ソフトウェア開発作業は，利用者が望む「実世界」と計算機の中に作られるモデルである「抽象世界」との対応づけを行うことである[1]．したがって，開発作業の効率化のためには，この計算機の中に作られる抽象世界の記述性をなるべく実世界に近づけてやることが重要になる．このモデル記述性の向上をもっとも素直に取り込もうとしているパラダイムの一つがオブジェクト指向パラダイムであり，このオブジェクト指向パラダイム[2]を支援するように作られているプログラミング環境が Smalltalk-80 である．

ところで，Smalltalk-80 は，もっとも充実したオブジェクト指向パラダイムのプログラミング環境として，早くから高い評価を受けながらも，実際にアプリケーションとして利用された例は意外に少ない．これには，いくつかの理由が考えられるが，処理スピード及びそれに比較したマシンコストがもっとも大きな点だったといえよう．また Xerox 社の Smalltalk 販売ライセンスの供与対象がハードウェアメーカに限られていたために，Smalltalk とマシンを一体で供給することしかできず，Smalltalk-80 が単体のソフトウェアとして普及する道を妨げていた点も見のがせない．

しかし近年，ハードウェア技術の進歩，移植技術の向上がめざましく，また Xerox 社の Smalltalk ライセンスの考え方も柔軟な方向へと変化しつつあり，特に，Xerox 社から分化した Parc Place Systems の EWS（エンジニアリングワークステーション）への積極的な移植活動も手伝って，Smalltalk-80 のアプリケーションへの応用の道が新しい局面をむかえている．

こうした状況をふまえ，本稿では Smalltalk-80 で実用のアプリケーションを組む場合における必要条件や，その効果を，画像処理応用を例に述べるとともに，Smalltalk-80 の課題についても言及する．

2. Smalltalk-80 実用化の条件

2.1 実行速度

Smalltalk-80 の実用化の条件として，ここではまず実行速度の問題について取り上げてみる．Smalltalk-80 は，実際には図-1 のように三つの部分からなる．最上位は，VI（バーチャルイメージ）と呼ばれる部分で，Smalltalk-80 システム全体を構成するオブジェクトの集まりである．これは実際にはハードウェア構成に依存しないバイトコードといわれる中間言語で記述されている．その下が，VI を実行するための，仮想的なコンピュータで，VM（バーチャルマシン）と呼ばれる．これは，VI を直接実行するハードウェアであってもよいが，多くの場合ソフトウェアである．この VM はさらに，バイトコードを解釈実行するバイトコードインタプリタと，その実行のためのメモリ管理及びプリミティブメソッドの三つの部分からなる．そして最下位がハードウェアである．

Smalltalk-80 におけるプログラムの実行とは，メソッドの起動のことであるが，このメソッドは，基本的には，メッセージセンドを行う式の集まりである．メッセージセンドの結果起動されるメソッドもまた同様である．このようにメソッドの起動のしあい（メッセージパッシング）が，Smalltalk-80 の実行の基本原

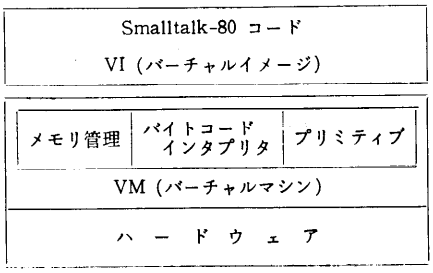


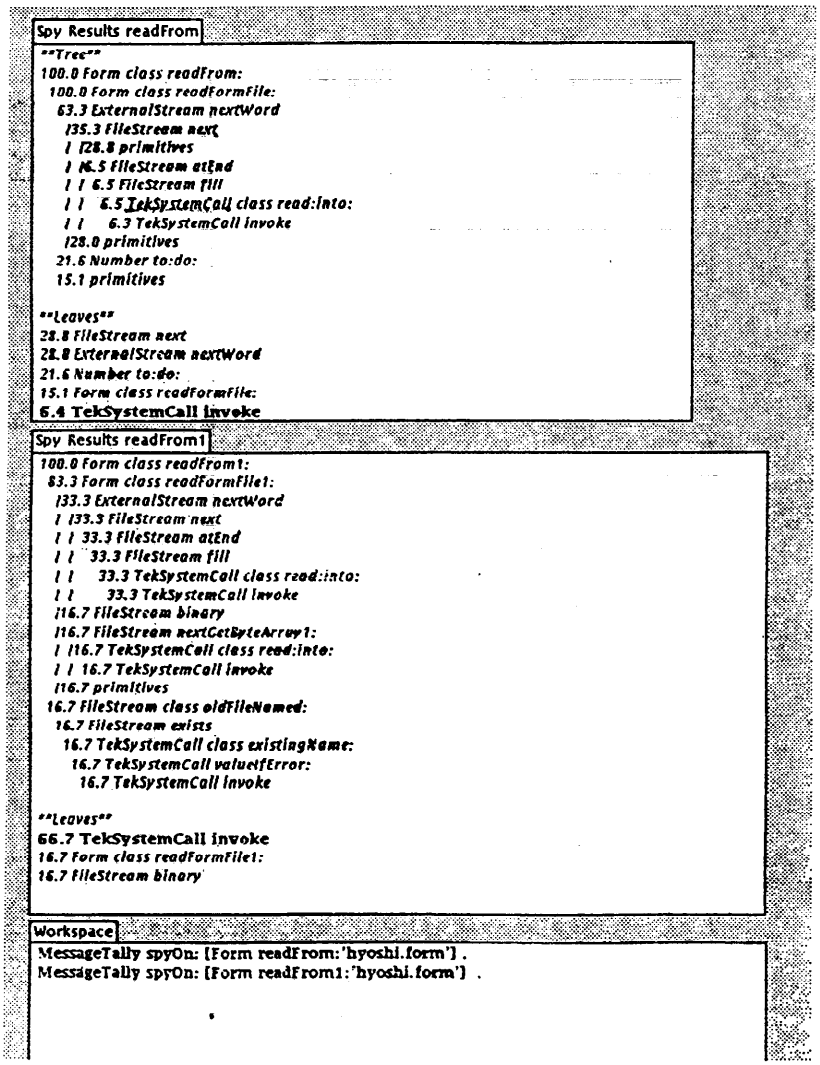
図-1　実行環境の概念図

† The Application for Image Processing by using Object Oriented Language by Takehiro KURONO and Masahiro NAGATA (Hamamatsu Photonics K.K. R&D division).

†† 浜松ホトニクス(株)技術部

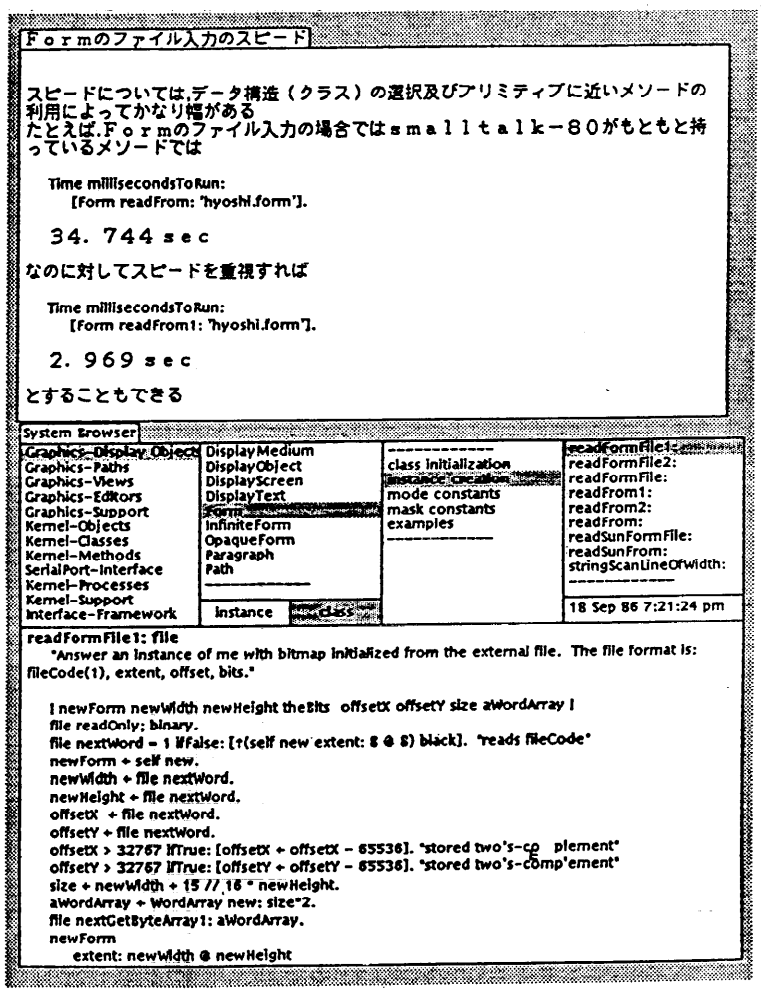
理になっている．このことから一般に Smalltalk-80 の高速化は，このメッセージパッシングの実行メカニズムの効率化が中心となる．（バイトコードインタプリタやメモリ管理の改良は，こうした目的である．）

ところが，これに比べ，実際にパッシングを発生している側，つまり Smalltalk 自身で記述されている VI 部分の効率化の工夫が，意外に行われていない．たとえば，Form クラスの基本的なクラスメソッドである readFrom：を例にして考えてみる（図-2）．このメソッドは，Disk 内の Form データを読み出し，オブジェクトを作成する機能をもつ．Form のサイズが，1024×1024 の場合その実行時間は，34.74 秒（Tek 4405）である．これを Messagetally クラスの機能を使って解析すると，図-2（a）のようになる．この結果をもとに Form 内の readFormFile：の記述を改良し VI 内でメッセージパッシングの効率化の工夫をすると，その実行時間は 2.96 秒になり約 10 倍の速度を得ることができる．Smalltalk-80 では，メソッドの起動のしあいが，実行原理ではあるもののこれだけでは結局なにも起こらない．現実には，メソッドのあるものは，プリミティブメソッドとして VM により直接実行されることで実際の処理が行われている．した



（a） Messagetally クラスによるメソッドの評価

図-2



(b)

図-2

がって図-2 の例が象徴するように VI 内のメソッド記述が，プリミティブメソッドを効率よく呼び出しているかどうかが，Smalltalk-80 におけるスピードにきわめて大きな影響を与えている．

しかも現実のほとんどのアプリケーションプログラムが実際に使用するメソッドは，数千にもおよぶ Smalltalk-80 内の全メソッドのごく一部である．そこで不要なメソッドを取り除いてしまえば，メッセージパッシングの効率もほかの言語の機能呼び出しメカニズムに比べて，そんなに非効率なものとはいえない．このようにアプリケーションの実行速度で，重要なのは従来からいわれるようなメッセージパッシングそのものの効率よりは，起動されるメソッドが，プリミティブメソッドを効率良く呼び出しているかどうかの VI 内部の問題と，プリミティブメソッド自身のスピード，さらには，定義されているプリミティブメソッドが，目的のアプリケーション実行に適しているか，(いい換えるとハードウェアを有効に利用できるようにプリミティブが設計されているか) といった点と考えている．

このような観点から，われわれの VM には，Smalltalk 内部からの動的なプリミティブ追加機能を実現し，これを積極的に利用し，画像処理応用で十分利用できることを確認した．また追加プリミティブの呼び出し失敗時には，等価の機能を Smalltalk 自身の記述で補足する手法をまもれば，ポータビリティの問題も発生しない．

2.2 外部インタフェース

Smalltalk-80 を実用機として利用する場合，外部システムとの制御，通信，データ伝送を行う外部インタフェースが重要となる．市販の Smalltalk-80 では，シリアルインタフェースは使用できるものの，画像データのような大容量データの高速伝送や計測器のような外部装置の制御ができない閉ざされたシステム構成となっている．しかし，最近では，EWS の普及によって，ネットワークによる分散処理や VME bus の採用によるオープンアーキテクチャが一般化したため，これら EWS 上で可動する Smalltalk-80 であれば，そのまま VI，VM の工夫しだいによって，Smalltalk-80 上から高速データ伝送や外部装置の制御が可能となる．

われわれは，具体的に画像処理応用で利用するために，画像入力用各種検出器が接続できる画像処理装置 DIPS[3] (Digital Image Processing System) を画像処理サーバとして用いることによって，画像入力や高速画像処理が Smalltalk-80 から制御できることを可能にした．

図-3 に示すように画像処理サーバの構成は，TV

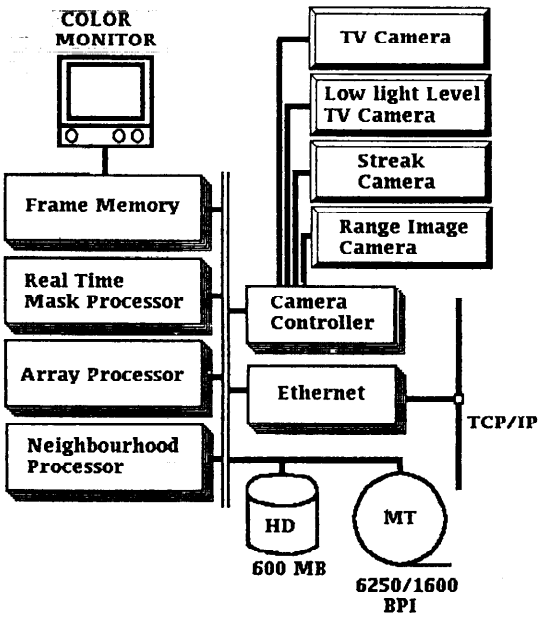


図-3 画像処理サーバ構成図

カメラ，微弱光計測カメラ[4]，超高速時間分解能光計測カメラ（フェムト秒光オシロ），距離画像計測カメラ[5]などの計測カメラの接続が可能で，リアルタイムマスク処理プロセッサ，ランダムアクセス型アーキテクチャのアレイプロセッサ，近傍演算プロセッサ[6]などの高速処理用プロセッサが内蔵されている．また，300 MB～1.2 GB の SMD ハードディスクや MT (1600/6250 BPI) などが接続されており，GPIB や DRIIW などの高速データ伝送用インタフェースも利用可能となっている．

従来，Smalltalk-80 上では，スキャナ入力画像（写真や紙絵）を除き，画像入力方法がなかったために，画像データの入手や画像処理応用が困難であった．しかし，この画像処理サーバの利用によって，顕微鏡画像，天文画像，部品検査画像，医用画像など多種の画像入力が可能となり，現在，画像のデータベース化を進めている．Smalltalk-80 から画像データライブラリを画像データベース SIDBA（情報処理学会）のフォーマットや UNIX の tar フォーマットで，MT やカートリッジテープに記録することができる．

3. 画像処理への応用

3.1 オブジェクト指向の必要性

Smalltalk-80 は，すべてがオブジェクトであり，そのオブジェクト自身がその振舞い方を知っている．したがって，メソッドの内容やツール群の詳細は知らなくても，簡単な手続の方法と構造が分かっていれば，オブジェクトにメッセージを問いかけることとツール群の利用によって，しだいに理解してゆくことができる．**図-4** を用いてメッセージパッシングの簡単な例について説明する．たとえば，数字の 1 は，そのほかのプログラミング言語ではただのデータだが，Smalltalk-80 では，これもオブジェクトである．つまり，数字の 1 が自分自身の振舞い方を知っている．数字 1 に対して class というメッセージを送れば，Small Integer クラスであることを答えてくれる．続けて allselectors というメッセージを送ると，数字 1 が理解しているすべてのインスタンスメソッドを答えてくれる．次に，インスタンスメソッドの内容を具体的に知りたい場合には，そのメソッドを Browser 内にコピーし，explain することによって，実行文と説明文が得られる．これを doit すれば，そのメソッド名で使われているすべてのソースリストをみることができる．そして，そのメソッドを数字 1 に送り実行してみれば，その動きをテストすることができる．このようにすべてがオブジェクトで記述されている一貫したコンセプトをもつ Smalltalk-80 は，こうしたツール群を検索しながら，部品として積極的に使ってゆくプログラム開発スタイルが可能であるといえる．

こうした開発環境を画像処理に応用した場合，特にメッセージパッシングは，画像の処理手順を記述するのに向いており，この様子を**図-5** で説明する．まず濃淡画像クラスとして FrameBuffer というクラスを新たに作成したが，2 値画像に関しては，Form クラスに加える形で実現した．FrameBuffer に fileIn: を送ることによって，Disk から画像を読み出し an-Image というインスタンスが作成される．この an-Image に imageSize, max, min などのメッセージを送り print it すれば，それぞれの答えが得られる．次に profileX: 100 は，x=100 の水平プロファイルを求めるメッセージで結果は，一次元データ（Word Array クラス）として aProfileX に書き込まれる．この aProfileX もまたオブジェクトであるため，一次元データ処理に関するメソッドが利用できる．画像処理では，処理の結果，データ型がいろいろと変化することが多い．したがって，そのほかの言語では処理結果のデータ型に合わせて，プログラミングに気を配る必要がある．たとえば，average という同じ名前の

図-4 オブジェクトへのメッセージ問い合わせ

```
"[1] The Example of HalfTone Image Processing by Smalltalk-80"

"1"  anImage ← FrameBuffer fileIn : 'temp256.img'.
"2"  anImage imagSize. 256@256
"3"  anImage max. 173
"4"  anImage min. 0
"5"  anImage average asFloat. 121.145
"6"  aProfileX ← anImage profilex: 100.
"7"  aProfileX average asFloat. 100.46
"8"  aHistogram ← anImage histogram.
"9"  anImage2 ← anImage medianFilter.
"10" anImage2 displayAt: 200@100.
"11" aBinary ← anImage toForm: 154.
"12" FrameBufferView ditherOpenOn: anImage2.
```

図-5 画像処理手順の記述

メッセージでも anImage や aProfileX のようにクラスが違えばそれぞれ異なったメソッドが実行されるというポリモルフィズム (polymorphism) の性質が利用でき，画像などのデータ型を意識する煩わしさから解放されるメリットは大きい．

3.2 MMI (マン・マシン・インタフェース)

画像処理応用では，入力した画像や画質のバラツキなどによって処理手順を変更することが多く，アルゴリズムを決定するまでに何度も試行錯誤を繰り返す．これは，まさに，画像処理用プロトタイピングとしての開発環境そのものである．

画像処理の操作形態は，利用者の要求レベルに応じて三つに大別することができる．第1は，ボタンを押すことによって操作する単純な操作形態である．多くの場合コマンド形式やメニュー形式などが利用され，最近では，処理手順の流れを視覚化するために，ポップアップメニュー，アイコンやマルチウィンドウ機能も利用されている．第2は，コマンド列などの操作手順を組み合わせることによって，独

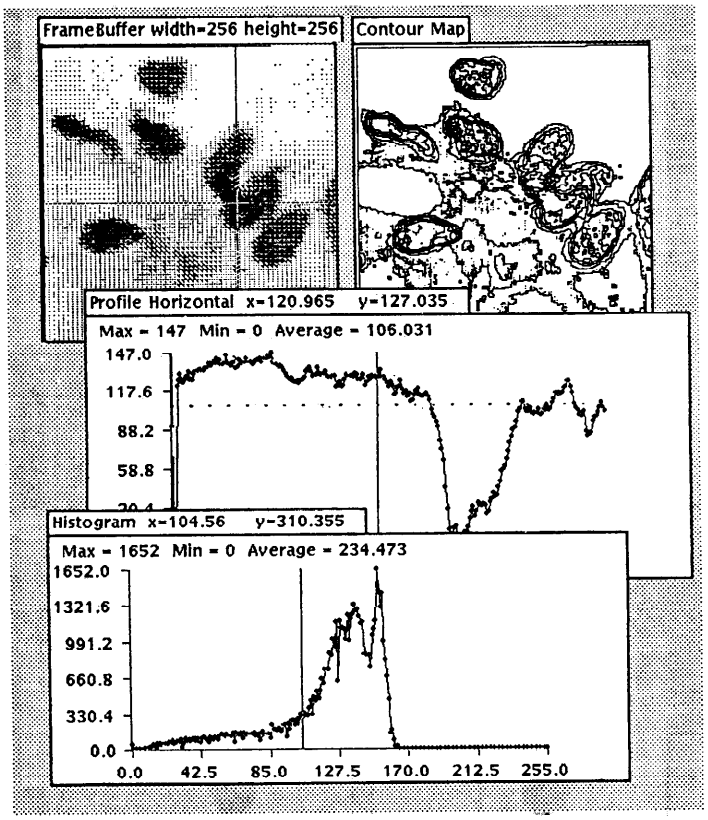


図-6 FrameBufferView の処理例

自の処理機能を追加したり，変更したりできる試行錯誤的プログラマブル操作形態である．これは，いわゆる画像処理言語のような機能として必要となってきている．第3は，ライブラリを利用し，利用者がプログラム開発できる操作形態であり，プログラミング作業そのものである．

このように利用者の目的とするシステム規模，用途や要求レベルに応じて操作形態を合わせてやる必要がある．画像処理システムにおけるソフトウェア開発の本来の目的は，収集されたデータを処理するプログラム開発であって，画像処理ソフトウェアだけの開発コストはそれほど高くかからない．ところが，上述のような操作性に関するソフトウェアの開発費用は，システム全体費用のかなりの割合を占めているのが現状である．

こうした利用者の要求レベルにすべて答えることはなかなか難しいが，われわれは，Smalltalk-80 がもつ豊富なツールに着目し，画像処理応用に必要な各種操作機能の実現[7] を試みている．

Smalltalk-80 は，部分実行，メッセージの追加や除去などが自由に行えるため，プログラム途中の画像やデータの内容を表示することによって容易に確めることができる．たとえば，図-4 に示すような簡単なプログラムで，アルゴリズム開発の試行錯誤ツールとしても利用できる．さらに，濃淡画像の表示に関するクラス FrameBufferView を利用して画像のプロファイルやヒストグラムなどの表示が行える．**図-6** は，ガン細胞画像の等輝線表示やヒストグラム表示の操作途中の画面を示している．このほか，アイコンやポップアップメニューなどの利用も容易で，画像処理実行環境の実現が容易である．**図-7** は，画像処理関係のツールやツリー構造表示の Browser などを表示したオリジナル版 VI の画面を示しており，現在，各種応用ツールの追加作業を進めている．

3.3 濃淡，カラー画像表示機能

画像処理を行う上で，濃淡8ビットあるいは，RGB カラー表示 (2^{24} 中 256 色表示) はどうしても必要である．現状の Smalltalk-80 マシンは，ほとんどがモノクロ表示であり，ディザ表示などの密度変調をせざるをえない．最近，濃淡 16 階調やカラー 16 色が表示可能な機種も利用できるようになったが，表示階調が少ないため画像処理応用には不十分である．

そこで，濃淡8ビット，カラー表示可能な EWS に Smalltalk-80 を移植し，濃淡，カラー画像表示を試み

図-7 オリジナル版 VI

ている．基本的には，画像の濃度値＝ディスプレイの表示値となっており，表示値 ϕ が黒レベルでしだいに増加し，最後に表示値 255 が白レベルとなる．しかし，一般に EWS は，モノクロ用と互換性をもたせてあるため，濃度値 ϕ が表示値 255（白レベル），濃度値 255 が表示値 ϕ（黒レベル）となっており，この二つの濃度値だけが白黒レベル反対となっている．これは，ルックアップテーブルで簡単に補正できるが，マルチウィンドウ表示されていると，指示したウィンドウのルックアップテーブルが有効となり，このウィンドウ以外の色表示に悪影響を及ぼす．これは，画像をマルチウィンドウで利用する場合の大きな問題となり，ハードウェアの改良が強く望まれる．

Smalltalk-80 は，VI や VM の高速化も重要であるが，BitBlt の表示時間の感覚がマシン全体の処理速度の使用感に大きく影響を与えている．つまり，カラーシステムが一般におそく感じるのはモノクロ表示に比べ表示速度が著しく低下しているためである．今後，ソフトウェア開発マシンとしてだけでなく，画像処理などの応用分野でも実用化するためには，濃淡，カラー画像表示用の BitBlt の高速化（512×512×8 bit の表示時間 1/30〜1/60 秒以内）が強く望まれる．

3.4 応 用 例

画像処理サーバの利用によって，画像データが容易に取り込めるため，入力した画像を用いて工業用部品の寸法計測，形状認識[8]，物体認識，顕微鏡画像の細胞判別や医用応用など各種実験の試みを行っている．図-8 は，ガン細胞の細胞の判別を行っている画面で，各細胞の形状計測を行うために，重なった細胞を分離している．2値画像処理は，BitBlt 機能を積極的に利用することによって，十分高速処理が可能である．図-8 の左上の画像は，2値化した画像で，これに収縮と拡張処理を行いノイズ除去した画像がその右の画像である．これからセグメント処理し，重なった細胞を抽出したのが Segmented Image で，各細胞ごとに分離した結果が Separated Result の画像である．このようにして，一つ一つ分離された細胞は，面積，周長などの物理計測後，特徴空間のクラスタリングによって判別分類される．

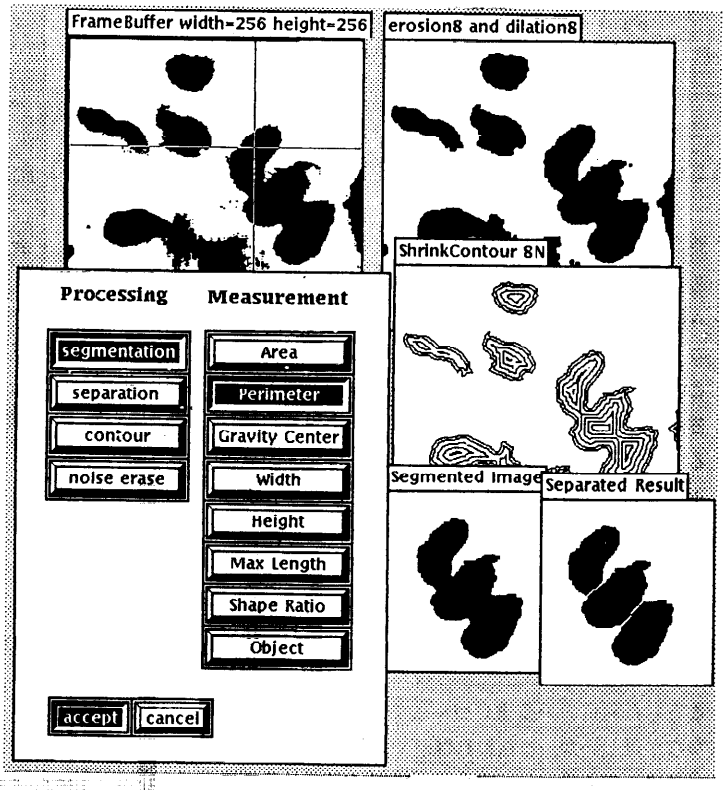


図-8 ガン細胞の細胞分離処理例

4. Smalltalk-80 の問題と課題

4.1 発展型プロトタイプ手法の必要性

Smalltalk-80 では，すべてがオブジェクトであり，そのオブジェクトの振舞いを記述するということがプログラミングになる．こうしたコンセプトはモデルの記述性を確かに向上させるものの，システム全体に対する透過性は著しく低下する．たとえば，あるオブジェクトが存在する場合，そのオブジェクトと，その時点で関連しているすべてのオブジェクトの位置づけを明確に把握することは不可能である．しかし，一つのオブジェクトをある目的に対して部品として使ってゆく場合には使いやすくなっている．つまり，部品を検索しながら積極的に使ってゆくプロトタイピングが Smalltalk-80 を有効に生かせる開発手法となる．

Smalltalk-80 は，特にラピットプロトタイプに向

いている．ところが，全体的な実行速度がおそいという理由から最終的には全く別の手続き型言語に書き換えられてしまい，使い捨てプロトタイプに終わることが多い．こうした使用法では，従来の開発手法がもつ仕様記述と実行プログラムの二重構造の問題をもち続けてしまう．そこで，ラピットプロトタイプから最終的なターゲットプログラムが完成するような発展型プロトタイプ[9]が Smalltalk-80 上で実現される必要がある．

具体的な実現方法としては，Smalltalk-80 内の不必要な開発支援ツールを削り取って，ターゲットプログラム部のみを残し，コンパクトなターゲット VI を作成する方法である．そして，今までも述べてきたように部分的にボトルネックとなっている部分を他の言語で補い，これをユーザ定義プリミティブとして扱うことで Smalltalk-80 世界に矛盾を起こすことなくコンパクト化，高速化が可能となる．こうした方法が発展型プロトタイプを実践できる最低条件で，われわれはこの方法を現在進めている．

もう一つの方法は，Smalltalk-80 で記述したターゲットプログラムを Smalltalk-80 内のツールによって，実行効率の高い他言語たとえば Objective-C に変換してしまう方法である．こうしたアプローチの確立が望まれる．

4.2 VI 間通信

Smalltalk-80 は，利用できる多くのツールが準備されているが，その完成度は一歩手前のものが多く，利用者がその不備な部分を自分なりにアレンジして利用しているのが現状である．特に他の VI のプログラムを利用する場合，不用意にプログラムを取り込もうとすると，自分のメソッドが書きかえられてしまうことがあり，ときにはシステムダウンなどの危険をともなうことがある．つまり，Smalltalk-80 は個人向け統合化プログラミング環境であり，他の VI やプログラムの共有化をすることはうまくいかないという問題がある．そこで，現状では，自分自身の VI に合うようにプログラムごとのアレンジとそれらの VI への取り込みが必要となり，その結果，VI は発展しながらしだいに膨張してゆき，VI の部分プログラムのポータビリティはどんどん失われてゆくこととなる．これは，Smalltalk-80 がすべてのクラスを静的に解釈することができないということに原因している．この解決には，実行中の 2 組の VI が互いに通信して，ほかのオブジェクトにメッセージパッシングして動作するメカニズム，つまり VI 間通信機能が必要である．われわれは，2 台の VI 間で，イーサーネットを介して，メッセージ通信で上記機能の実験を進めている．

4.3 オブジェクト指向プログラムガイドの必要性

Smalltalk-80 上で発展型プロトタイプが行えるようになると部品化再利用の技術はかなり高いレベルに達する．このソフトウェア開発の効率を良くするためには，信頼性の高い評価された部品を確立された手法で組み合わせてゆくことで対象とするモデルの記述が完成してゆくことが重要である．ところがオブジェクト指向パラダイム上での確立された組み合わせの手法が全くないのが現状である．どういう目的の場合に新たなクラスを定義し，そのクラスの階層構造や性質をどのように決定するか，また追加，変更していったクラスをいつどのような目安で整理し，再編集するのかなどといったガイドラインの確立が望まれる．

5. おわりに

オブジェクト指向言語 Smalltalk-80 を画像処理に応用した場合について検討し，その結果，目的のアプリケーションに対応したユーザ定義プリミティブを用いることによって，当面大きな問題であった処理速度の向上，Smalltalk-80 上からの外部装置の制御及び濃淡・カラー画像表示などの解決の糸口が得られた．このことは，Smalltalk-80 に発展型プロトタイプを実現するための最低条件が整ったと考えて良い．

優れたプログラミング開発環境をもたらした Smalltalk-80 は，あくまで個人向けシステムであって，他システムの VI との通信機能のメカニズムがないため，必要なメソッドをすべて自分の中に取り込んで，自分自身の VI を膨張するしかない．今後，Smalltalk-80 が実戦的応用で使われるためには，こうした VI 間の通信機能が重要となる．

最近のハードウェア進歩は目ざましく，Smalltalk-80 の実用化はますます近づいてくる．われわれは，画像計測分野を中心とした Smalltalk-80 の VI，VM の機能強化を行いながら，アプリケーションツールの開発を進めてゆきたいと考えている．

参考文献

1) 大野 豊，阿草清滋：ソフトウェア工学からみた自動プログラミング，情報処理，Vol. 28, No. 10, pp. 1262-1269 (1987).
2) 上谷晃弘（編著）：統合化プログラミング環境，丸善 (1987).

3) 細田 誠, 島 晴久, 加藤 誠：高速汎用画像処理システム C 2000-02, 03, Vol. 19, No. 11, INDUSTRIAL, 映像情報, pp. 47-55 (1987).
4) Tsuchiya, Y., Inuzuka, E., Kurono, T. and Hosoda, M.: Photon Counting Image Acquisition System and its Applications, Journal of Imaging Technology, Vol. 11, No. 9, Oct, pp. 215-220 (1985).
5) 花嶋正昭, 黒野剛弘：距離画像計測とその応用, テレビジョン学会技術報告, IPA-86-3, pp. 41-46 (1986).
6) 細田 誠, 黒野剛弘, 土屋 裕：近傍画像処理プロセッサ, AMOEBA, 第16回画像工学コンファレンス, 3-2, pp. 45-48 (1985).
7) 黒野剛弘, 永田真啓：Smalltalk-80 による画像処理ワークベンチ, Computer Today, 9月号, No. 21, pp. 34-38 (1987).
8) 黒野剛弘：自動化に使われる画像処理技術, 自動化推進, Vol. 15, No. 3, pp. 9-13 (1986).
9) 佐伯元司：実行可能な仕様記述, 情報処理, Vol. 28, No. 10, pp. 1346-1358 (1987).

(昭和63年1月26日受付)