報　告

パネル討論会

視覚的プログラミング環境

昭和61年後期第33回全国大会†報告

パネリスト

紫合　治[1]，　鷹尾　洋一[2]，　平川　正人[3]

細谷　僚一[4]，

司会　市川　忠男[5]

総合司会　それでは，定刻になりましたので，これからパネル討論会を「視覚的プログラミング環境」という題で始めさせていただきたいと思います．

私は，現在情報処理学会の事業担当の理事をしております，東大の石田と申します．慣例によりまして，私のほうから司会者の広島大学の市川忠男先生をご紹介したいと思います．それでは，市川先生よろしくお願いいたします．

司会　ただいまご紹介いただきました広島大学の市川です．私ごとが先に立って恐縮ですけれども，私と情報処理学会とのかかわりと申しますと，それほど密なものはなかったような気がいたします．たとえば，いま思い出すことと申しますと情報処理学会が発足いたしまして五，六年ほどたったころに，情報処理学会誌のバックナンバーを揃えて神田へ出かけまして，それを確か2万円に替えまして，（笑い）その2万円がたちまち飲み代に変わったというようなことであります．（笑い）かといって，とりたてて飲んべいだということではありません．ただ，学会誌は残さないようにしようと考えたわけなんです．これはなかなか難しい問題で，すぐざ折してしまいました．その次が，学会誌は読まないようにしようというふうに考えたわけなんです．これはかなり実行可能な命題でございます．（笑い）

そんな程度の人間ですから，ここへ立つのはどうかなという気もいたしまして，表に出ないほうがいいのではないかと言ったんですけれども，なにか郷土芸能をご披露するというような意味合いもあるようでございまして，お引き受けしたわけでございます．東京からおいでになりましたパネリストの方々には，よろしくお付き合いのほど，お願いいたします．

† 日時　昭和61年10月3日(金)12：30～14：45
場所　広島工業大学
1）日電，2）日本IBM，3）広島大，4）NTT，5）広島大

プログラミング環境とその視覚化

ところで，近ごろ“しつけ”が大事というような声が，かなり粗野な響きをもって鳴り響くようになってまいりました．この“しつけ”と申しますのは，プログラミング，つまりソフトウェアづくりの分野では1970年代の初頭から言われておりまして，これがいわゆるソフトウェア工学というわけでございます．

随分とずっこけた話になりますけれども，皆さんご経験になっていると思いますが，海外などへ行かれますと，飛行機から降りて，まず入国管理手続きを受けます．その係員のいるガラス張りのボックスの二，三歩手前の床に黄色い線が引いてありまして，その線の後ろに1列に並んで順番を待ちます．こうして必要な手続きがスムーズに流れます．「なるほど大したものだ．これがしつけか，マナーか…」と感ずるわけでございます．

しばらくして，日本へ帰って来ます．大阪空港，あるいは成田空港になるわけですが，今度は例の黄色い線がありません．後ろからどんどん押されます．せめて自分の前だけでも空間を確保したいと足を踏ん張ったりしますと，突き飛ばされる．これはちょっと話がおおげさなんですが，（笑い）たまたまポケットからパスポートが出なかったりしてもたもたしておりますと，後ろからぐっと手が伸びてくる．で，後ろを振り返りますと，手を伸ばすところまではいかないんですが，マナーのいいはずの外国人までがぐんぐんと背中を押してくるのを見かけます．「これは“しつけ”じゃない．環境だ！」という気がいたします．つまり，環境があって初めての“しつけ”であり，プログラミングの分野でも，「環境が満たされて初めてソフトウ

ェア工学の基本理念が生きてくるんだ」というふうに感ずるわけでございます．

そこで，環境というものをもう少し具体的に考えてみたいと思います．今度は臭い話で恐縮なんですけれど，日本のトイレを覗いてみます．

ご利用になる方が多くて容器が全部ふさがっているとします．さて，どこへ行くか．なるべく列の短いところを選ぶ．ところが往々にしてお年寄りなどが混じっておりますと，その順番が狂うということがございまして，非常に具合が悪いわけです．（笑い）ところがアメリカでは，皆さん便所の入口のところでずっと1列に並んでいらっしゃる．空いたところに列の最初の人間が行く．非常に具合がいいわけで，これこそまさに“しつけ”の問題だというふうに考えるわけです．

ところが，日本では入り口はちょっと狭くって，体を置く場所を探すとなりますと，それはお仕事をなさっている方の背後にしかないというのが現実です．だから，やはり環境なんです．こういうわけで背後に立つと全体のようすが見えなくって，人間関係もぎくしゃくしてくる．そこで「まず環境をビジブルにしよう．そこから Friendliness が生まれてくる」ということを期待して，随分とこじつけがひどいんですけれど，こうして今日のパネルのテーマ「視覚的プログラミング環境」ができ上がるわけでございます．

さて，それで視覚情報というものを見てみますと，まず直感性に優れている．したがって，理解の助けになるという特色があります．すでに今日のプロフェッショナルな十分に信頼できるソフトウェア生産工程のうえで，非常に有効に利用されています．これは Program Visualization というふうに捉えられるんじゃないかと思います．この土俵の中で，まず日電のソフトウェア生産技術研究所にいらっしゃる紫合治さんに，特にソフトウェア設計の立場からお話を伺いたいと思います．その次に NTT のソフトウェア生産技術研究所の細谷僚一さんにご登場いただいて，デバッガ・システムについてのお話をしていただきます．

さて，視覚情報の特色といたしまして，言語独立の共通概念を表しているということがあります．これは，言葉は変わっても絵の意味するところは同じだということ．そのうえ子供は作文を覚える前にまずお絵かきから始めるわけですから，このほうが親しみやすい．そこで，絵を組み合わせてプログラムを作っていこうじゃないかという考えも出てくるわけで，先ほどの Program Visualization に対して，これを Visual Programming といいます．この分野については，IBM の鷹尾洋一さんにお話いただきます．「エンドユーザのための」というまくらことばを設定されておりますけれども，かなりグローバルな立場でいろいろなお話を伺えるんじゃないかと思っています．

それから最後に，広島大学の平川正人さんには，アイコニック・プログラミングとかなりテーマを特定いたしまして，ごく新しいお遊びのお話を伺いたいと思っています．皆さんにはその間に十分頭を熱くしていただきまして，そのあとディスカッションに移ります．そしてもし時間がありましたら，最後に視覚的プログラミングやその環境といったものはソフトウェア工学の基本理念にどのようにかかわるのか，あるいはまた，ソフトウェア工学の基本理念に沿ってみるとき，それらにこの先どのような貢献が期待できるのだろうかというようなことを意識いたしまして，パネルをまとめさせていただこうと思っておりますが，そこへ至る過程で，どんどん皆さんから活発なご質問をいただき，またディスカッションにも参加していただきたいと思っております．よろしくお願いします．

それでは，まず最初が日本電気の研究所の紫合さんでいらっしゃいます．紫合さんは，ソフトウェア生産技術研究所の技術課長として，プログラミング言語，ソフトウェア開発方法論，開発環境などの研究に従事していらっしゃいます．お願いします．

視覚的ソフトウェア設計

紫合 日本電気の紫合でございます．視覚的ソフトウェアの設計についてお話したいと思います．後で鷹尾さんから事務システムの場合についてお話があると思いますので，ここでは通信や制御向けシステムについて考えたいと思います．

まず最初に，視覚的ソフトウェアをどのように捉えるかという**図-1** を示します．一昨日東工大の吉田先生の「感性と論理」という，非常におもしろい特別講演がありましたが，それに合わせて感性と論理という軸を引いてみました．ソフトウェアの設計では，最初は論理的にはあまりはっきりしていない，たとえば広島工業大学の入試の処理を EDP 化するとか，それをある予算範囲でやるとかいう，感性的な「早く」とか「よく」とか，そういう感覚で要求が出されるわけです．それを，最終的には論理的に十分な情報をもった

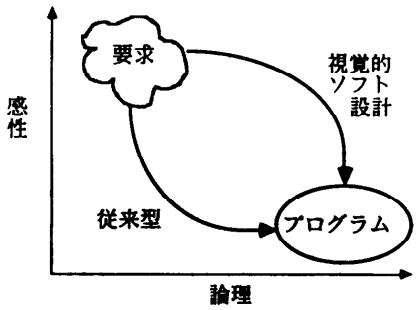

図-1 視覚的ソフト設計のとらえ方

プログラムにしていく．すなわち，感性的ではあっても論理的にはあまり情報がないもの（要求）から，感性には訴えなくても論理的にはっきりしたもの（プログラム）にもっていくというのがソフトウェア設計だといえます．

その場合に，要求からプログラムへのもっていき方なんですけれども，従来型では，まず要求があって，それを分析していろいろな文書にするわけです．要求仕様書とか，設計書とか，感性よりむしろ論理を重視した文書が一生懸命書かれるわけです．そこで，要求からプログラムに至るのに，まず感性が落ちてくる．それを視覚的ソフトウェア設計では，できるだけ感性的な，直感的な分かりやすさを保ったままで論理を深めていき，最終的にはプログラムにまで至るというようなことを狙っているといえます．

このような視覚的ソフトウェア設計の支援としてはいろいろあると思いますが，私は大きく三つに分類しました．最初は要求から設計に至る段階の支援です．多くの場合，要求というのは，最初頭の中に思い浮んだいくつかの例によって表現されます．そういう例集合から一般化した規則にもっていくというのが設計といえます．別の言葉で言いますと，定数の世界から変数の世界にもっていくのが設計です．ここで，例示は視覚的な表現が適していることが多いといえます．したがって視覚的な表現による例集合を入力して，一般化規則を見い出すための支援が考えられます．こういう仕事の一つとして，Programming by Example という例，によってプログラムを生成していくという研究があります．

第2は，このようにしてできた一般規則のチェックの支援です．このチェックというのは，一般化規則から視覚的な例を導き出して，その例を人間が見て正しいかどうかを確かめることといえます．これには Program Visualization，プログラムの実行過程の視覚化という研究があります．

第3は，このようにして確認した設計からプログラムを出していく支援があります．そこでは，Visual Programming といいますか，視覚的な設計からプログラムを生成していく研究があります．

次に，視覚的ソフトウェア設計の実用化の要件について述べます．まず，入力と出力の差が大きいこと，少ない入力でたくさんの出力を出すことです．特に視覚化では，同じことを示すのに絵と文とでどっちが簡単に入力できるか，というところで問題になってくるかと思います．それから，実際的なシステムを扱うためにモジュール構造が扱えること，図形などが簡単に入力できること，この辺につきましては，最近マウスとかポップアップメニューとかでかなり実用的になってきていると思います．さらに，結果が美しくて作業が楽しくなる，応答が早い，というようなところが実用化の要件かと思います．

以上の一般論をもとにしまして，具体的な例を少しお話したいと思います．ここで，CCITT/SDL (Specification and Description Language) という，交換機などの仕様や設計を状態遷移図やブロック構成図で記述する言語の解析ツールを例にとり，具体的に視覚情報がどういう具合に役立つかというようなところをお話したいと思います．

SDL では，あるブロック（処理単位）の入出力信号をどのように処理していくかということを示すのに状態遷移図（**図-2**）が使われます．図-2 は交換機のある一部の機能（呼処理）を示した状態遷移図の例です．このような図を端末から入力しまして，その解析を支援していくというようなシステムは，われわれだけではなくて，いろんなところで作られております．

このぐらいの図になりますと，いまは設計者が手で書いていることが多いわけですけれども，修正などを考えますと，むしろ機械入力のほうが楽になります．マウスとメニューを使いながらこういう図が比較的簡単に，手作業と同じぐらいに，あるいは慣れるともっと早いスピードで書けるようになってきております．さらに，たとえば複数のプロセスがそれぞれこのような状態遷移図をもっているとしますと，そこからプロセス間の信号のやり取りをシミュレートして**図-3** のような信号フロー図を得るというような解析や，状態遷移図からプログラムを自動生成するというようなツールが開発されております．

われわれのシステムで特徴的な機能としまして，特

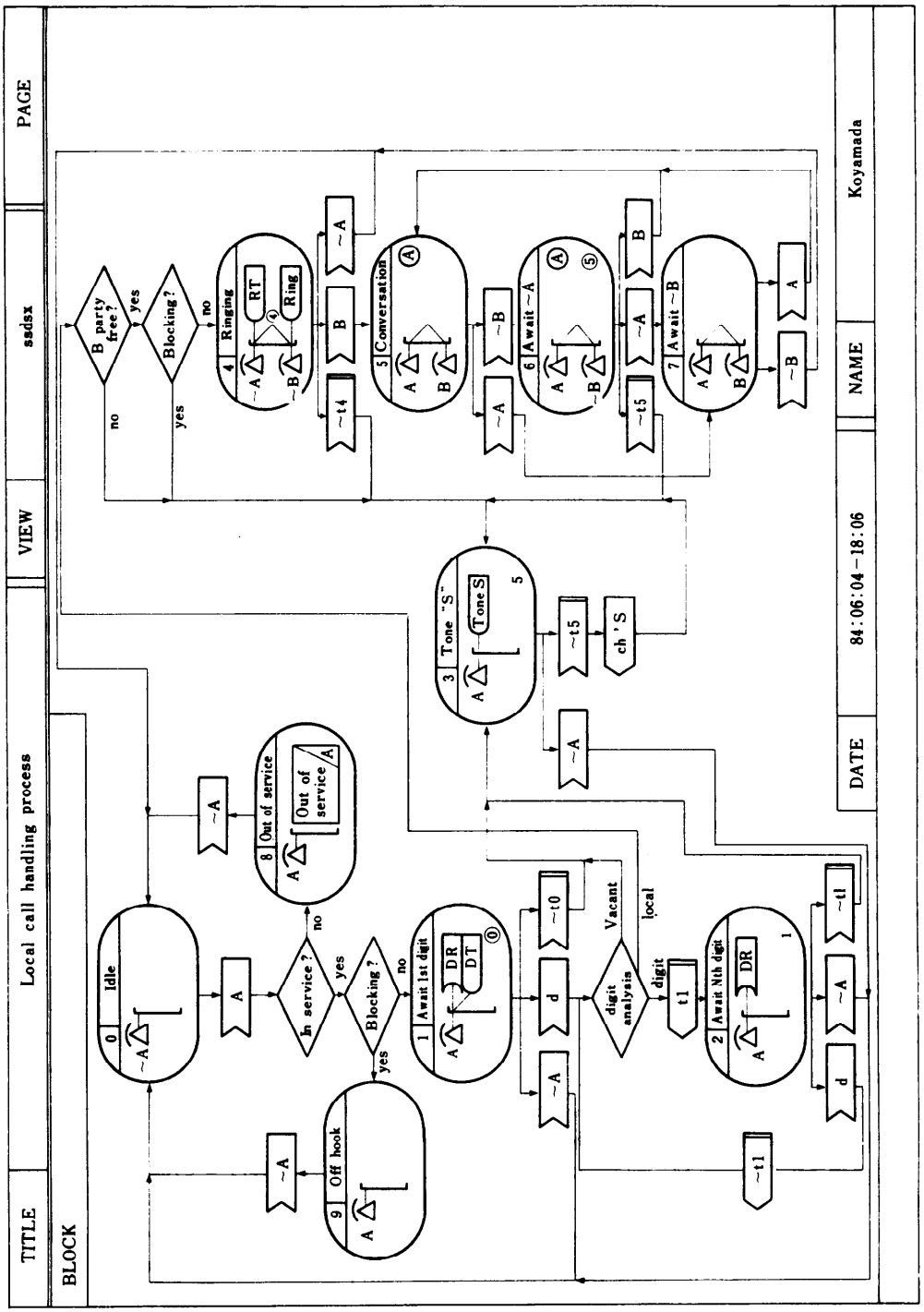

図-2 CCITT/SDL による交換機の状態遷移図

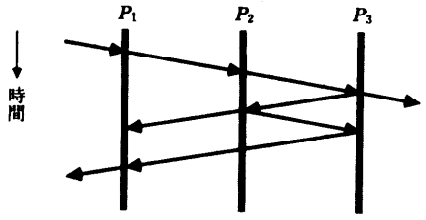

図-3 プロセス間信号フロー図

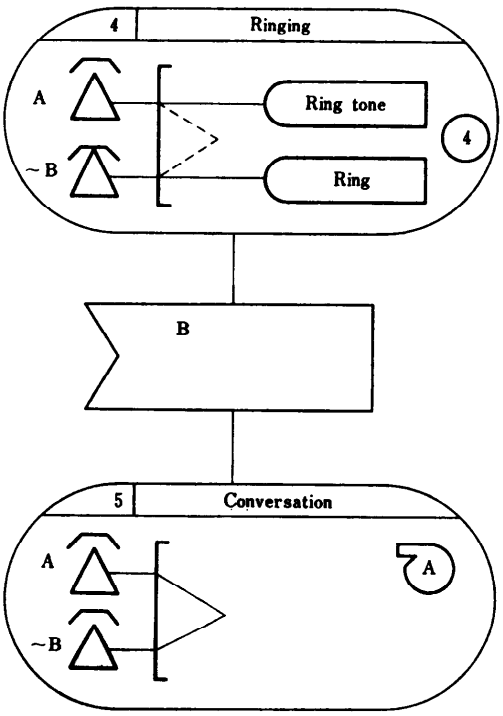

図-4 呼出し中から通話への状態遷移図

に状態遷移図の視覚的なところを活用したものを紹介したいと思います．図-4 は，先ほどの状態遷移図で，"Ringing" 状態から "Conversation" 状態に遷移する部分を拡大したものです．ここで，相手のベルが鳴っている Ringing 状態というのは，状態の中に絵がありますが，「Aさん（発信者）の電話機が上がっていて，Bさん（受信者）の電話機は下がっていて，Aさんにはリングトーンが鳴っていて，そしてBさんにはベルが鳴っている状態」を表しています．また，AとBをつなぐ破線は回線が予約されていることを，④はタイマ4が働いていることを表しております．この状態からBさんの電話が上がりますと，"Conversation" 状態に移って，「AさんとBさんの電話が上がっていて，回線がつながって，さらにAさんに対する課金が行われている状態になる」ことが絵で表されているわけです．

この絵は，交換機と電話機からなるシステムの，抽象的で，かつダイナミックな構成図を表しているというふうに考えられます．これから状態の絵の差分をとりますと，必要な処理が出てくるわけです．たとえば，Ringing 状態でBさんの電話を上げたという信号を受けますと Conversation 状態に至ります．このとき，「リングトーンを止める，ベルを止める，タイマを止める，それから回線をつないでAさんの課金をスタートさせる」というような処理が必要です．これらは，状態内の絵の差分から分かるわけです．

さらに，たとえば Ringing 状態の絵を見れば，この状態で待つべき信号はBさんの電話機が上がるということのほかに，「Aさんの電話が下がる（今上がっているから），タイムアウトになる（タイマが働いているから）」ということもあるのが分かるわけです．このように状態の絵を調べて，それが正しい状態遷移図かどうかをチェックしたり，それから必要な処理を生成したりする機能がある程度できるようになっています*．

もう少し，夢といいますか，今後の方向としての考察を三つほど述べたいと思います．まず第1に，先ほど述べた状態の中の絵の変化から処理を見い出すというのは，ちょうど4コマ漫画に対応しているということです．通常4コマ漫画というのは，ある場面と次の場面との間の差分を見て登場人物の動作を捉え，それで意味を知るというような読み方をします．そこには背景があり，それぞれの登場人物がそれぞれ異なった動作をしていく．また，それぞれの登場人物に性格があって，それに合わない動作は許されないという規則は漫画にもあるわけです．

状態遷移図の場合，それぞれの登場人物，電話機とかリングトーンを鳴らす機器とか，そういうものがそれぞれ変化していくわけです．すなわち，登場人物である抽象装置の状態の変化を絵にしたものということになります．このような絵の差分でソフトウェアを作っていくという Programming by "漫画" といいますか，そういうことができればと思っております．

第2に，先ほど信号フロー図をお見せしましたが，ここでは縦軸が時間になっています．これを映画のフィルムだとしますと，プロセスをボックスに表し，実際に CRT 画面上にもっと動的なアニメーションがで

* Shigo, O. and Koyamada, M., "Designer's Work Environment for Communications Software", GLOBECOM '85.

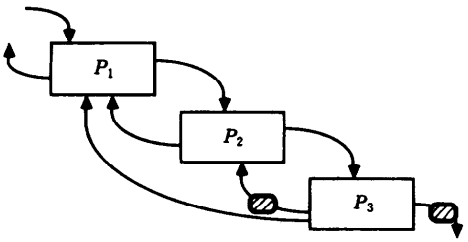

図-5 プロセス間信号フローのアニメーション

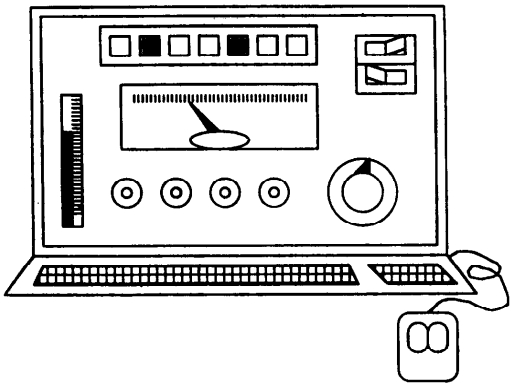

図-6 ソフトマシンによる操作パネル画面例

きます（図-5）．プロセス間のチャネルに信号が流れて，たとえば，P_3 から P_2 に向かって信号が流れていく．この信号が P_2 に着きますと，P_2 が P_1 と P_3 に対して信号を出していくというようなダイナミックな絵（映画？）が出るわけです．逆に，Programming by Example では，P_3 から信号が P_2 に入ったら P_1，P_3 へ信号が出て，P_1 が外部に信号を出すというようなことを動的に指示すると例示ができるわけです．このような信号フロー図的な例から一般化して状態遷移図に直していくというような研究も進んでおります*．

最後に，高機能のグラフィック端末を使ったソフトマシンの活用について述べたいと思います．ソフトマシンというのは画面上に表示された操作パネルのことで，制御システムなどにはよく見かけます．従来はハードの操作パネルがあって，そこにいろいろなボタンスイッチやメータなどがついていましたが，最近では，端末の画面に従来の操作パネルと同じような絵が出ていまして（図-6），ボタンを押す代わりにマウスでボタンの部分をクリックすればよいようになっています．ですから，ユーザは昔のハードの操作パネルを使っているのと全く同じような感覚で使えます．これを活用しますと，一つの画面の中にいくつもの操作パネルがあり，たとえば電話機の絵やレーダの絵があり，その上である操作を指示すると信号が流れていってレーダがくるっと回るとか，そういう視覚的なインタフェースが設計の段階でのチェックなどに使えるんじゃないかと思います．この辺はまだまだおもちゃ的な感じがするんですけれども，概略設計のレベルであれば，そういうおもちゃ的なもの，すなわち模型的なものでも案外便利に使えると思います．

計算機というのは万能マシンなんですけれども，画面にどんな操作盤でも書けるようになれば，視覚的にも万能マシンという感じがしてきますし，いろいろなシミュレーションなどが視覚的にできるんではないかと考えてます．

司会 ありがとうございました．今すぐにでもフロアから質問が出てきそうな気配もいたしますけれども，後がつかえておりますので，後三つ話を続けて伺いたいと思います．

この次は，ビジュアルなデバッガ・システムをご紹介いただけると思います．NTT の細谷さんは，生産技術研究所のプログラム言語研究室長として言語処理プログラム及びツールの開発に従事なさっておられます．よろしくお願いいたします．

視覚的デバッガ・システム

細谷 先ほどのお話にもありました，プログラミングにおけるグラフィックの利用の一環といたしまして，プログラムをいかにビジュアルにするか，つまり，プログラムを分かりやすくするのにどうやったらビジュアル情報を使えるかということを試してみたシステム（VIPS）をご紹介したいと思います．

VIPS は，Visual and Interactive Programming System の略です．これは，本来は設計から保守までの全ライフサイクルをビジュアル化することによってソフトウェアの生産効率を高めることをねらったシステムですが，まずは，現在できておりますデバッガ部分についてご説明したいと思います．

デバッガ自体は従来からありまして，最近ではシンボリック・デバッガといって少なくとも言語のシンボルのレベルでデバッグができる，つまりアセンブラ・レベルまで戻らなくてもいいというものが常識になっ

* 伊藤，市川，柴崎，梶原「交換ソフトウェア設計・管理システム SDE の構想とその適用性について」信学技報 SE 84-82.

てます．そういう従来のシンボリック・デバッガは，トレースの情報とかダンプの情報をテキストの形で出力するもので，すべてテキストがベースになっています．たとえば，数値データについては実際に値で出てくるので分かりやすいのですが，ポインタについてはアドレスそのものが出てきます．そうなると，これは正しいのか正しくないのか，すぐには分からない．結局はコアイメージがどうしても必要になってくる．そういうことで，高級言語を使いながらもどうしてもアセンブライメージを意識しなければいけないということで，バクの究明が困難だという面があったわけです．

これを分かりやすくするために，実行状況を図形で表すことを考えました．図形と言うものは情報量も多く表示できるし，また人間の感性が図形に向いているということから非常に有効に使えるのではないか，それによってスタティックなものだけではなく，一種のアニメーション，つまり動きそのものを表示することによってデバッグを効率化できないだろうかと考えたわけです．

デバッガをビジュアル化しようという動きはこれまでにもいくつかあります．大きく分けて，文字レベルのものとグラフィックを使ったものとがあります．文字レベルのものは，表示内容はテキストイメージなんですが，画面を利用して二次元情報として表示する．ですから，たとえばソースプログラムが画面に表示されて，その中の実行されている文が次々と反転表示されるということが行われます．このタイプのものは，パソコンなどの上にのってすでに商品として世の中に出ています．

それからもう一つのタイプとして，これは研究レベルのものが多いんですが，データやプログラム構造などをグラフィックで表示して読解性を高めようという動きがあります．そのうちの一つが，われわれが試作した VIPS です．図-7 に VIPS の一画面を示します．本当は，動いているようすをお見せしないと実際どうなっているか分かりにくいのですが，この図はある瞬間に止めたものです．大きく分けて，左側がデータを表示している部分で，右側が手続きを表示している部分です．

このシステムでは，操作はメニューとマウスで行われます．本図中の Interaction window はユーザがシステムとインタラクションを行うウィンドウです．Block structure window にはどういうふうに手続き

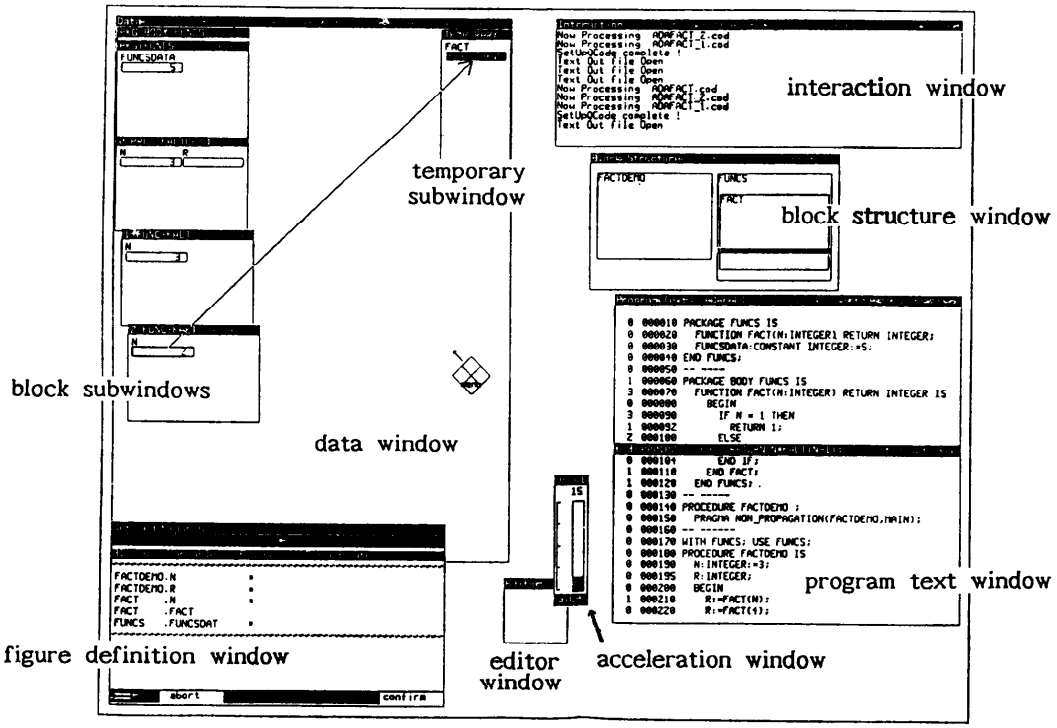

図-7 MULTI-WINDOW SCREEN

が順々に呼ばれてきたか，いま現在どの手続きのどこが実行されているかということがグローバルに表示されています．Program text window には，Block structure window と同期して現在実行されている手続きのテキストが表示され，その中の実行されている文が反転表示されてプログラムの動きに従って動いていきます．

Data Window 中にはデータが表示されます．ブロック構造をもった言語の場合は，生きているブロックのデータが Block subwindows に表示されます．また，現在実行中のプログラムのテンポラリ・データが Temporary subwindows の中に表示されます．そして，代入があった場合には，代入元から代入先へ矢印が表示されます．このように，マルチウィンドウを利用して，複数のプログラムのデータや手続きを一つの画面の上に表示して全体を把握しやすくしています．

最後に，データを表示するための図形の形を定義する Figure definition window があります．どのデータを表示するか，そのデータをどういう形で表示するかということをここで定義します．

なお，ダイナミックにエリアが割り当てられていくヒープ域は List subwindow に表示します．図-8 には，次々とエリアが割り当てられ，それらがチェーンでつながっている状況が表示されています．表示エリアがいっぱいになると，各要素の表示を少し小さくします．いくら小さくしても入りきらないときは，要素中の情報表示を少し省略して，常に全体を表示できるようにしておきます．これによって，リスト構造が思ったとおりできているかどうかが一目で分かります．

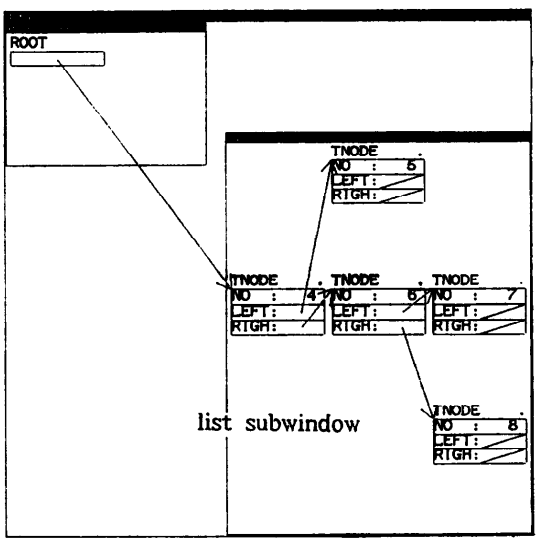

図-8 LINKED LIST EXAMPLE

もっとも重要なことは，データをどんな形で表示したらいいかということです．本システムでは，スカラ，アレイ，リスト，レコードなどについて標準的な図形が定義されています．デフォルト時には標準図形が表示されます．さらに，利用者が図形を定義する機能があります．ユーザは，あるデータ構造はスタックであるとか，あるデータは棒グラフだとか，いろいろ頭の中では実際のデータの形を想定していますが，プログラムを書くときはスカラ，アレイ，レコードというような一般的な言語のノーテーションに変えてしまっています．

この図形表示機能を使えば，たとえばインタナルソートの場合，配列の値を棒グラフで表示すると，実行中に順々に棒グラフが並んでいくようすが分かります．また標準図形の代わりにバーチャートを使うとか，スタックの有効な部分だけを表示するということができます．あらかじめ図形を定義・登録しておかずに実行中に定義・登録することも可能です．これによって意味に合った表示ができるわけで，これが非常に有効だと考えています．

本システムは，プログラムを解釈実行する ADA インタプリタと，図形の表示を行う FDL インタプリタから構成されています．FDL は ADA 流の図形定義用の言語です．この二つが連動して Data window にデータを表示します．

実際にどう動くかを，式を解析してポリッシュ形式に変換する例で説明します．A+B/C−D という式を入力し，スタックを用いて解析し，最終的にポリッシュを出力します．図-9 に，初期状態でリストに式が入っている状態を示しています．ある程度実行が進んだ状態が図-10 で，スタックにはプラスとマイナスが入っており，カレントシンボルにはマイナスが入っており，そしてポリッシュ出力にはAとBが出ているという状況です．本図では，スタックについて，デフォルトの表示のほかに自分で定義した図形表示も使われています．これを見ながら本当に自分の思っているとおりにプログラムが動いているかということをデバッグしていくわけです．

本システムは，デバッグ以外にも教育に使えると考えています．たとえば，リカーシブな動きとか，コンカレンシなどは初心者に口で説明してもなかなか分か

PARSE-2

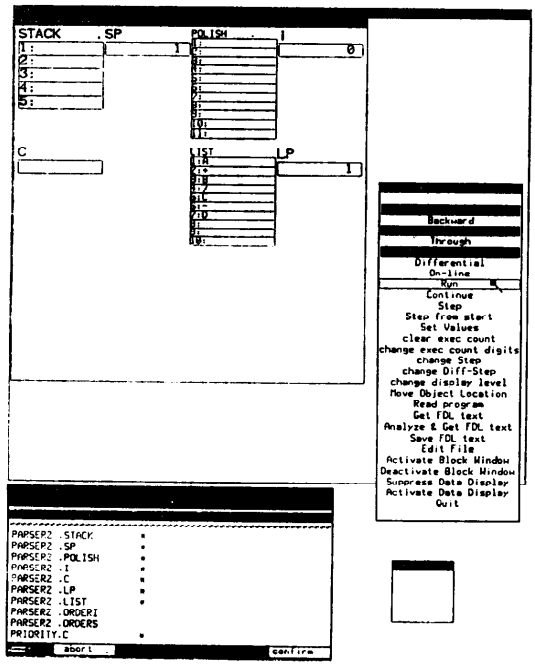

図-9

PARSE-7

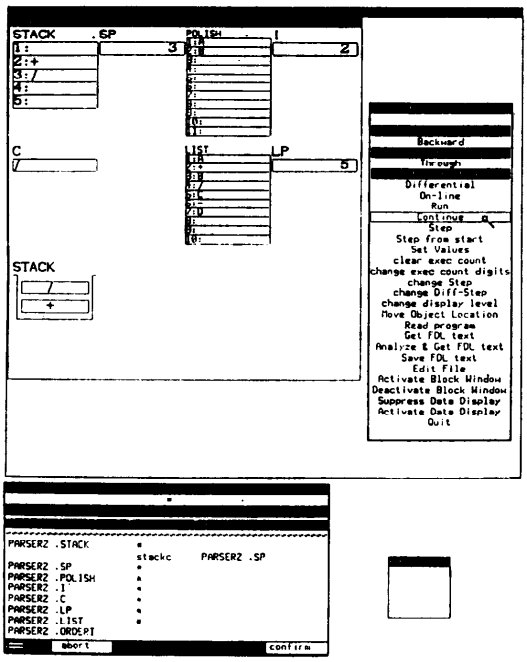

図-10

らない.「百聞は一見にしかず」ということで,そういうものが動いているようすを見せてやる.普通の教育用のビデオテープですと固定的なパターンしか表示できませんが,これだと生徒がつくったプログラムがそのまま動いて,自分のプログラムはこう動くのかというのが分かるという点がよいと思っています.

それからもう一つ,紫合さんの話とも絡みますが,設計工程にも利用できると思っています.たとえば,フローチャートのうえでプログラムを実行するとか,ブロックダイヤのうえで実行する.あるいは状態遷移図のうえで実行する.そういうものにも発展させていきたいと考えています.ただいくつか問題点があります.一つは,ワークステーション上の実行時間で,いろんなことをやろうとすればするほど時間がかかる.あまり時間がかかると使いものにならない.そこで,できるだけ高速化したいということでワークステーションの高性能化が必要になります.一方,実際のプログラムの生産現場で使おうとすると,どうしても低価格化が必要になります.1人に1台ずつ配ろうとしますと,大幅な低価格化が必要となります.相反することですけれども,この辺は最近の技術の進歩に期待しているわけです.

それから3番目がデータの意味に合った表現法ということで,いまは典型的なもの以外は全部ユーザが定義するという方法ですが,本当はユーザが定義するのではなくて,データの意味をシステムが解釈してその意味に合った形で表示することができたら非常に有効ではないかと思います.そのためには,設計チャートから情報を取り出してくるという方法も必要と考えています.

司会 どうもありがとうございました.それでは,次に IBM の鷹尾さんにお話を伺います.鷹尾さんは,サイエンスインスティチュートにおられまして,データベース,画像処理,文書処理や通信ネットワークなどいろいろなことを手掛けていらっしゃいます.それから,1985 年から 1986 年にかけて米国のワトソン研究所にいらっしゃいまして,その間にはワークステーション・プロジェクトの企画,立案に従事なさったということでございます.お帰りになりましたあと研究所の名前が変わりまして,IBM の東京基礎研究所.そこで,システムテクノロジ担当として,分散システム,データベース,それにオフィスシステムのプロジェクトに従事なさっていらっしゃいます.よろしくお願いいたします.

エンドユーザとの親和性

鷹尾 私は，いままでの紫合さんや細谷さんとちょっと立場が違いまして，オフィスシステムの中でエンドユーザに対してなにを提供できるか，あるいはなにが必要なのかということを考えております．したがって，今日お話するのは，エンドユーザのためのプログラミング・ファシリティをどのような形で提供したらよいかということです．私の話は，具体的なものをつくって，そしてこういう結果が出たということではありません．むしろ，問題提起ということで，皆さんにこの分野のことを考えていただくきっかけになれば幸いだというつもりでお話いたします．

まず，言わずもがなかもしれませんが，なぜエンドユーザ・プログラミングが必要なのかについてお話したいと思います．

自分の計算機のプロファイルを設定する，あるいは，パーソナライズするためにプログラミングの要素が入ってきます．それから，典型的な，何度も繰り返すような，いわゆるコマンドプロシージャとか，同様なものですけれどもデータベースに対する Predefined Query をつくっておいて，後は簡単にそれを呼び出す．それから，よくいわれる作表とか，レポート作成です．さらに文書作成ですけれど，いま大部分は静的な文書なんですけど，この文書の中に時間軸がいったん入ってくると，その作成作業はプログラミングそのものだと考えられるわけです．したがって，エンドユーザ・プログラミングは今後どんどん重要になるだろうというのが私の見方です．

エンドユーザ・プログラミングが必要だということは以上にして，それではなぜビジュアルかですが，これも多分多くをいう必要がないでしょう．われわれのオフィスを見ても，パーソナルコンピュータ，あるいはワークステーションがごく普通にあり，すべてがスクリーン・オリエンテッドであるし，また，このスクリーンの上のものを直接指示して操作を行うようになっているわけです．

ビジュアルの意味を，なんらかの形で二次元的に表示されたものに対して操作を行うものであるというように広く定義すると，これからは，ビジュアル以外の方式を考えるのはほとんど意味がなくなると思います．つまり，いまさらリニアシンタックスに戻る積極的な理由はなにもないということです．ここまでで，エンドユーザのための Visual Programming というのは必要であるし，また非常に自然な今後の方向であろうということがいえたものと考えてください．

ここで，話を展開するために，一つの論文を引用させてください．それは，今年の ACM SIGCHI のコンファレンスでトロント大学の Brad Myers という人が Visual Programming の Taxonomy について提案した論文でして，なかなかきれいに整理されているので，これを出発点にしたいと思います．

Brad Myers によれば，まず，一般に言われる Visual Programming は大きく二つに分けられる．第1は，先ほど紫合さんのお話で出てきました Program Visualization です．つまり，通常のプログラミング言語，たとえば，Cや Pascal で作成したプログラムを視覚化する，すなわち，グラフィック表現するというものです．それから2番目は，本当にプログラムそのものを視覚的につくっていく．アイコンを用いたプログラミングはその典型的な例で，後ほど平川先生のほうから詳しいお話があると思います．以後，私の話ではこの後者，すなわち狭義の Visual Programming に焦点をあてていきます．

Brad Myers は，この狭義の Visual Programming を，さらに Programming by Example (PBE) かそうでないかということで分類しています．

この Programming by Example ですが，これはある意味で非常に重要で，多分 Visual Programming が本当に使われるようになったときの一つの主流になるだろうという期待をもっています．

ここで，Brad Myers の Taxonomy を少し修正しようと思います．それは，Visual Programming といったときに，いま多くの人が使っているスプレッドシートを無視するわけにいかないと思うからです．すなわち，スプレッドシートを考慮したカテゴリを追加します．スプレッドシートは，ある意味で，各セルの中にそのセルの値が満たすべき Constraint をとり込んでいるわけです．ここで，スプレッドシートのように Constraint を指定してプログラムをつくっていくようなものを，Programming by Constraint (PBC) と呼ぶことにします．

したがって，**表-1** に示しましたように，エンドユーザのための Visual Programming を分類する場合に二つの軸がありまして，PBE かそうでないか，それから PBC かそうでないかということになります．

表-1 Visual Programming for End-Users: Taxonomy

- Programming by Example (PBE) (vs non-PBE)
  - Programming by Rehearsal
  - Programming by Demonstration
- Programming by Constraint (PBC) (vs non-PBC)
  - Constraint-based Programming
  - Declarative Programming

	non-PBE	PBE
non-PBC	Flowchart Data Flow Diagram	Rehearsal World ....
PBC	Spreadsheet Query by Example	ThinkPad ....

PBE の場合には，Programming by Rehearsal であるとか，Programming by Demonstration という言葉も使われています．それから PBC の場合には，Constraint-based Programming とか，少し広義になるかも知れませんが Declarative Programming という場合もあります．

このような二つの軸を考えますと，図に示したように 2×2 で4つのセルが出てくる．左上のセルは，フローチャートとか，データフローダイアグラムなどのプログラムの視覚的表現を用いるものです．つまり従来のプログラミングの概念を知っている人を対象として，視覚化によって効率よくプログラムの開発やデバッギングを行おうとする領域です．したがって，プログラミングに関する知識を前提にしていますので，エンドユーザにとっては負担の大きいものになるのではないかというのが私の考えです．つまり，エンドユーザ・プログラミングという立場からは，むしろ PBC および PBE を重点的に考えるべきだと思います．そして，それ以外の領域は，プロフェッショナル・プログラマのためのツールということでカバーすべきだろうと考えます．

それでは次に，PBC, PBE の話に移らせていただきます．まず PBC ですけれども，最初に申しましたように，これは多分 Visual Programming と呼ばれるものの中で一番成功しているし，実用に供されているものだと思います．特にスプレッドシートは，ご存じのようにオフィスにはあふれておりますし，また，データベース言語として Query by Example (QBE) や Forms-oriented Languages がすでにいくつか実用化されています．スプレッドシートがなぜこれだけ普及し，成功しているのかという点について，私もはっきりした解答をもっているわけではないんですが，多分一つは，従来のプログラミングではかなりダイナミックな振舞いをユーザが意識しなければならなかったのに対して，スプレッドシートでは非常にスタティックな見方を与えている点にあると思います．言い替えれば，内部での計算はダイナミックなのですが，その複雑なダイナミックな動きを全部隠してしまっているということが一つの重要な要素だろうと思います．

ところが，全部スタティックな見方をさせますので，逆にダイナミックな問題を扱おうとすると非常に困るということがあります．それから，いまのところ表の計算とデータベース操作という，非常に限定された問題領域でしか考えられていません．

次に PBE ですが，本当に実用化されたものは，私の知るかぎりないと言っていいと思います．ただ非常に有望だと思います．特に，ダイナミックな動きを入れたエンドユーザ・プログラミングというものを考えた場合にはこれしかないのかもしれないと思っております．ある意味で，この PBE というのは，最近流行の Direct Manipulation に基づくプログラミングだと思います．言い替えれば，抽象的な概念ではなくて，スクリーン上に映されている具体的なものを動かしながらプログラムをつくっていくということですから，非常に分かりやすいわけです．

この PBE にはいくつかレベルがあると考えられます．1番目は，単純な Recording Playback です．Demonstration のためには結構役立ちますし，かなり利用されていますが，これだけでは分岐のない1本の流れのプログラムしかできないわけですし，その中で扱われる対象がすべてコンスタントであるということで，プログラムと呼ぶには単純過ぎるかも知れません．

その次のレベルはバリアブルを導入することだと思います．ただ，このときも実際に例を使いながらプログラムをつくっていくんですけれども，そのときに，たとえばコンスタントとバリアブルの区別をどうするかは，必ずしもそう単純ではありません．それから，実行時にそのバリアブルと実際の値をどうバインドするかという話もそれほど自明な問題ではなくて，まだまだ考えなければいけないと思います．

それから3番目．本当にプログラムと呼ぶためには Conditionals を入れなければいけないんですけれども，現在は大変なことに，すべての if-then-else のパ

スを指示してやらないといけません．ただ，これではあまりにも負担が大きいので，この点ももっと工夫をする必要があると思っております．

最後に，残念ながら結論がないんですけど，私個人の期待としては三つあります．つまり，PBC の方向を追究する．それから PBE を追究する．それから PBC と PBE のよいところをとってなにか新しいパラダイムをつくれないかという三つの方向です．

ここで，PBE については，本当に使いやすいシステムを実現するためになんらかのインファランスの機能を入れないといけないと感じています．いわゆる Automatic Programming の再来を期待することになるのかも知れませんが．

それから，PBC のほうも Constraint の指定をより高水準にしようとすると，どうしても知識ベースにつながっていかなければいけないのかなと，ぼんやりと感じている段階です．私自身は，必ずしも AI の流れに乗らなければならないと思っているわけではないんですけれども，初日のパネルをお聞きして，やはり AI に期待する部分がかなりあるというのがいまの率直な感想です．まとまりのない話でしたけれど，以上でとりあえず私の話は終わらせていただきます．

司会 Visual Programming とはなにかということについて，鷹尾さんご自身の定義に従いますと，きわめてノン・ビジュアルなやり方で明解にご説明いただいたような気がいたします．（笑い）どうもありがとうございました．

それでは最後に広島大学の平川さんにお願いします．

アイコニック・プログラミング

平川 私は，アイコニック・プログラミングというタイトルで少しばかりお話させていただきます．市川先生のお言葉を借りると，若干お遊び的な要素もあるかと存じます．

視覚言語の分類については先ほど鷹尾さんからお話がありましたけれども，まず私なりに Visual Programming について分類を行い，その後にアイコニック・プログラミングとはなにかという話をさせていただきたいと思います．

マン・マシン・インタラクションの過程に視覚情報を利用するアプローチとしては，処理結果をグラフとか，あるいは表の形で出力するといったものが以前からあったわけです．その後たくさんの研究が進められてきていますが，それらは大きく二つの流れにまとめられるんじゃないかと思います．

最初の一つは，オブジェクト間の構造を視覚化するものです．QBE やフォーム言語に例をみることができます．細谷さんのご発表にありました VIPS もこの範疇に入ると思います．

これに対して，オブジェクト自身を視覚化するといったアプローチがあります．具体的な例としましては，アイコンをあげることができます．オブジェクトの視覚化された抽象表現，いわゆる絵シンボルに，ファイルであるとか，プログラムコードといったオブジェクトの意味記述が結び付けられており，アイコンをディスプレイ上でポイントすることによって，そのアイコンに結び付けられているオブジェクトの意味記述がアクセスされるようになっています．

ところでアイコンは，ゼロックス社の Alto システムという計算機に最初に導入されて以来いろいろな計算機やワークステーションに導入されてきていますが，それらのシステムにおいてはデータとかシステム・コマンドの視覚化といったシステム実行環境における視覚情報の利用にとどまっています．既存のアイコンを組み合わせて新しいアイコンに組み上げる機能，つまり，プログラミング機能というのは持ち込まれていません．

このアイコンというものをプログラミング環境に持ち込むとどうなるか．そうすることによってプログラムの作成が簡単に行えるようになるんじゃないかといった期待をもって進められている研究がアイコニック・プログラミングであると，ここでは定義します．

ある特定の分野で用いられるアイコニック・プログラミングについては，すでにいくつか実用化ならびに市販化されています．たとえば，コンストラクションゲームキットをあげることができます．ピンボールゲームはその一例ですが，ピンボールゲームを構成するコンポーネントがアイコンの形で用意されており，それをディスプレイ上で組み合わせることによってユーザは自分の希望する自分だけのピンボールゲームを定義することができるようになっています．

こういった専用のものに対し，汎用のアイコニック・プログラミング言語についてはまだ研究段階にあるといえます．

その一例といたしまして，現在われわれが開発中の

HI-VISUAL という言語を取り上げ，これについて説明をさせていただこうと思います．まず最初に HI-VISUAL におけるアイコンの定義について簡単に説明いたします．アイコンは，絵シンボルを表す External part とオブジェクトの内容を表す Internal part の組として定義されています．また，アイコンを7つのタイプに分類し，絵シンボルを囲む枠の形によって，それぞれのアイコンがどのタイプに属するかといったことが目で見て分かるようにしてあります．

さて，実際に HI-VISUAL においてプログラミングがどのように行われるかということについて，ある画像処理プログラムの作成を例にとって説明させていただきたいと思います（図-11）．

まずユーザは，マウスなどのポインティング・デバイスを用いることによって，アイコンメニューの中から希望するアイコンをポイントし，それをプログラミング領域上の希望する位置に移動・配置します．ここで，配置されたアイコンが実行可能であればシステムではそれをただちに実行し，その実行結果を表示します．ユーザは，この結果を見ながら次の操作を決定することができます．

たとえば，入力画像 (CAMERA OUT) に二値化するための機能 (BINARIZE) を接続しますと，その処理結果 (BINARY OUT) が表示されます．このように HI-VISUAL においては，ユーザはアイコンを用いて視覚的に，かつインタラクティブにプログラミングを進めることができます．鷹尾さんの分類に従いますと，Programming by Example のカテゴリに属するかと思います．

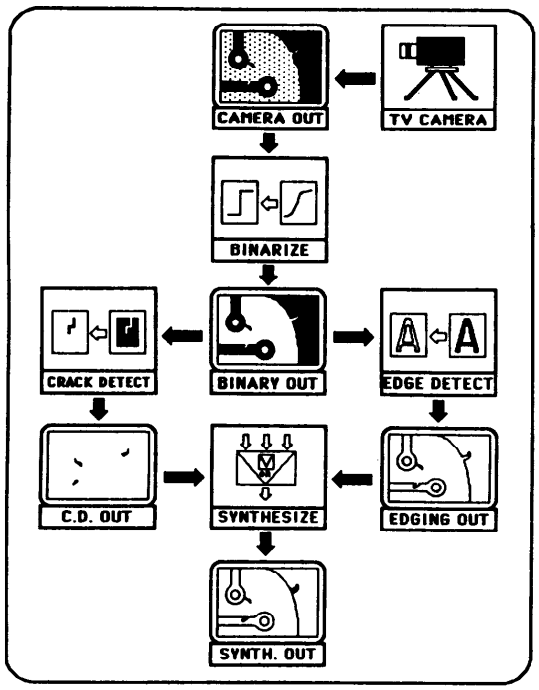

図-11 HI-VISUAL プログラム例

このような操作を繰り返し行うことによってプログラミングが進行するわけです．そして，でき上がったプログラムは一つのアイコンとして定義可能であり，それ以降は新しく定義されたアイコンを用いて，より抽象度の高いプログラムを作成することができるようになります．

それでは，プログラムがシステム内でどのように実行されるかということについて簡単に説明します．

プログラムはアイコン・ディスクリプタを用いて管理されています．アイコン・ディスクリプタは，アイコンの Internal part と External part，ならびに他のアイコンとの接続関係を示す Relation part からなっています．アイコンがディスプレイ上に配置されますと，それに対応したアイコン・ディスクリプタがシステム内に生成されます．プログラムの実行に当たっては，アイコン・ディスクリプタを参照しながら接続関係に従ってアイコンの Internal part を実行します．

ここで，アイコンが複数のアイコンの組み合わせによって定義されている場合には，このアイコンは，それを構成しているアイコンの組み合わせに展開された後に実行されます．すなわち，HI-VISUAL におけるプログラムの実行は，システムにあらかじめ用意されているもっとも基本的な機能（プリミティブ・アイコン）のレベルに展開された後に実行されるようになっています．

ところで，HI-VISUAL をはじめとするアイコニック・プログラミング言語においては，プリミティブ・アイコンに相当するものにどういうものをもってくるかによってシステムの適応範囲が決まってきますが，われわれのシステムにおいては，プリミティブ・アイコンの定義をユーザに開放するために，Primitive binder というツールを用意しています．つまり，オフィス処理用のプリミティブ・アイコンを用意することによってオフィス環境に，画像処理用のプリミティブ・アイコンを用意することによって画像処理環境に適用することができるようになっています．

それでは最後に，アイコニック・プログラミング言

語に関して，実際にシステムの開発をとおして気づいたことをまとめておきたいと思います．まず利点といたしましては，アイコンの意味が絵シンボルとして視覚化されていますので，ユーザは一見してその意味を把握できるといった効率のよさがあげられます．また，アイコンによるオブジェクトの視覚表現は，情報伝達手段として万国共通ではないかと思います．

しかしながら，その半面，ある絵シンボルをユーザに見せた場合に，それから想起される意味が必ずしも一致しないという問題点があります．また，アイコニック・プログラミング言語を含めて，視覚言語ではグラフィック処理が必要となります．そのようなことを考えてみますと，次のような研究が将来的に必要ではないかと考えています．

まず，CPU を含めたハードウェアの性能向上が必要です．特にグラフィック表示であるとか，グラフィック操作専用のハードウェアの提供が必須であると考えています．これについては，最近高品位ディスプレイ装置であるとかグラフィック制御用の LSI が盛んに開発されており，ハードウェア面では近い将来にかなり満足のいくものができるのではないかと考えております．

重要なのはソフトウェアサイドの研究でして，たとえば，オブジェクトをどのような絵シンボルで表現すればいいのか．これについては工学の分野だけでなく，心理学などを含めたほかの分野と共同で研究を進めていくことが必要だと思います．また次に，アイコンの意味解釈は，システムの使用環境であるとか，状況であるとか，ユーザによって適宜変更されるべきだと思います．そのような機能を実現するためには知識ベースシステムの導入が必須であると考えています．

さらに，アイコンの合成機能の拡張が必要だと思っております．具体的に言いますと，これまでに提案されているアイコニック・プログラミングシステムにおいては単にアイコンを並べることだけが許されていましたが，今後は，たとえば二つのアイコンを重ね合わせることによって新しいアイコンを合成するといった機能が必要ではないかと思っています．

質 疑 応 答

司会 どうもありがとうございました．これでパネリストの方々のお話が終わったわけでございますが，予定時間を大幅に上まわっておりまして，皆さん多分フラストレーションがたまっておいでじゃないかと思います．早速，皆さんのご質問をお受けしようと思います．

牧之内(富士通研究所) 非常におもしろく聞かせていただきました．二つばかりちょっと質問させていただきたいんです．

まず，平川先生の話なんですけど，これからエンドユーザの言語という感じでアイコンというのは非常に重要であろうと思うわけなんです．その先駆けという研究で，非常におもしろく聞かせていただいたんです．ただ，プログラミングといいますと，非常に細かいレベルでコードを書いていかないといけないわけです．ところが，先ほどの絵から見ますと，非常にハイレベルな，すでにプログラムされてあるものをつなげていくという感じを強く受けます．本当にあれでプログラムできるのかというのが僕の疑問なんで，その辺の見通しをお聞きしたいと思います．

それからもう一つは細谷さんの話なんですけれども，非常にプログラムの細かいレベルでビジュアライズされているということは分かりますけれど，それは，たとえばわれわれが実際にプログラミングしているときにいろいろ工夫してデバッグしていることをただマルチウィンドウで表しただけなんじゃないかという気がします．それで，本当にあれがどれぐらい実用的なものか，プログラミング・ライフサイクルのどの局面でどれくらい効果があるのか，という辺を少しお聞きしたいと思います．

司会 間違いなく出てくるだろうと思っていた質問が出てまいりまして，どうもありがとうございました．(笑い) どちらからでもいいんですが…，平川さんからお願いします．

平川 この質問については，どういったプリミティブ・アイコンを用意するかといったことが絡んでくると思います．われわれがいま開発している HI-VISUAL において，たとえばアセンブリ言語のレベルのプリミティブ・アイコンを用意すれば，確かに非常に細かい低レベルのプログラムが組めるわけですけれども，われわれはそういったところにこの HI-VISUAL を適用することは想定しておりません．

HI-VISUAL がその機能を十分に発揮するのは，コンピュータについてよく知らないエンドユーザ・サイドだと思っています．プリミティブ・アイコンとしてあくまでも抽象度の高いプログラムを用意しておき，それらを組み合わせてより抽象度の高い応用プログラムを視覚的に作り上げることができる．そういっ

たことを目指してシステムを設計しています.

司会 とにかく，私が聞いておりましても Visual Programming-Why Not？というような強腰の表現が非常に印象的なんですが，いまのような質問をお受けしますと，当面はエンドユーザ，あるいは初心者向きといった逃げも一方で用意されているようでございまして，(笑い) まだ納得がいかないということもあると思います．またこの次のお答えをいただいた後でも結構です．どこまでプロフェッショナルユーズに耐えられるんだろうかということを，形を変えてどんどんご質問いただいてもよろしいかと思います．それでは，細谷さんお願いします.

細谷 ご質問は，このシステムが本当に実用的か，いままでの作業とあまり変わらないんじゃないかということだと思います．われわれがこれをつくった狙いは，実用を狙おうということで，グラフィックとか，ビジュアルというものをいかにしたら実用の世界にとり入れることができるかという観点で試作したわけです．おっしゃったコメントが，まさにわれわれが狙っているところであるというふうに思っています.

つまり，現在やっている作業，たとえば人がフローチャートを手で書いたり，あるいはボックスを手で書いてその中に値を入れたり，リストに手で印をつけたりとか，そういうふうに人間がデバッグ作業とかレビュー作業を行っているときに手で行っていることを機械化して，その分だけ人間の手作業を軽減し，かつ確実にしていこうというのがまず第一の狙いです．そういう意味で確かに飛躍はないかもしれませんが，少なくとも現在の作業が楽になるということは確かだと思います.

後，どのフェイズで有効かということですけれども，当然のことながら，まず第1がテストやデバッグ工程を狙っているわけです．もう一つはレビュー工程です．たとえば，コードレベルだけではなく，もう一段上のブロックレベルでのトレースもできますので，レビューで役立つと思っております.

それからもう一つの大きな狙いは保守工程です．つまり，つくった人がいなくなってしまうことがあるわけで，後から来た人は，本当は設計ドキュメントを読んだら全部分かるというのが一番いいと思うんですけれども，特に細かいレベルになりますと，いったいどう動くんだろうか全然分からないけどバグ票が出てくる．本当にこのように直していいかどうか分からないけど，見れる範囲で見てプログラムを修正してしまう．VIPS により動きをトレースすることによって，他人のプログラムを理解したうえで直すということができるんじゃないかと思います.

私自身がこれを実用に使ううえで一番問題かなと思っているのは経済性でございまして，ある程度速いワークステーションを非常に安価に提供できるかというところにキーがあると思います．ただ，これはある程度ときが解決してくれるんじゃないかと思っております.

司会 どうもありがとうございました．細谷さんのお仕事は，プロフェッショナルなソフトウェア生産環境で視覚情報を有効に取り上げ，利用していこうということでございますから，そのよしあしについてもプロフェッショナルなプログラマが評価なさるわけで，その辺りに厳しさがあります．ご質問の意図もそういうことだったと思いますが，いかがでございましょうか．ご質問といまのお答えとは.

牧之内(富士通研究所) 大体，予想された答えだと思うんですが，(笑い) 私も舌足らずで質問の意図がうまく伝わらなかった面があるんですけれども，もうちょっと深く質問させていただきますと，それじゃ，たとえばテストの生産性が実際にどれくらい上がったかということ，その辺を僕は知りたい．つまり，僕はあれを見て，たとえばシステムプログラムをつくるときに本当に生産性が上がるのかという問題，ちょっと疑問があるんです．その辺をお伺いしたいんですが.

また，ビジュアルにするということは，たとえば設計過程ではかなり役に立つんじゃないかという気がするわけですが，しかしデバッグとなりますと，たとえばシステムプログラムなどの場合に，ただ個人に「デバッグしたか」と聞いて，「デバッグしました」ということだけでは済まされないわけです．必ずテストのデータを用意して，どれだけ本当にテストしたのかということがはっきりつかまえられないと，そのシステムはリリースできないわけです．だから，デバッグの場合に，視覚化の評価としていったいどういうことがやられているのか，その辺のことをちょっとお聞きしたいと思います.

細谷 評価という意味ですけれども，大規模なシステムプログラムに適用して測ったという段階ではありません．プロトタイプができたばかりで，いまのところ例題レベルで評価しています．そういう意味では，本当の大規模なシステムプログラムの中に入れたときに，インタプリタで実行していますので，タイミング

が絡んでくるようなものについてどこまでデバッグできるかは問題です．プログラミング言語で Concurrency をきちんと定義して，タイミングに関係なくプログラムが実行されるというつくりにすればいいんですけれど，現実のシステムプログラムというのは全然そんなものではないんです．アセンブラで書いた多くのモジュールが全部組み合わさって一つのシステムになっていまして，タイミングまで含めてきちんと検証できない状態です．

後もう一つ，ビジュアルにすることは管理にも役立つと思います．たとえば，テストやデバッグの管理にしても，テストの網羅性，それからどこまで進捗したか，バグの発生傾向はどうか，それから出荷の管理，つまり，この修正がこのバージョン入っているのか入っていないのか，そういう項目が数多くあるわけです．こういうものについても視覚化という技術は使えると思っております．そういう意味で，Visual Debugger はその一部をやっているということで，全体として Visual Interface Programming にもっていきたいと思っております．そのときは，当然管理の視覚化が重要な要素になってくると思います．

司会 ありがとうございました．初めの質問に戻りまして，結局その Visual Programming がどこまでプロフェッショナルなユーズに耐えられるものになるんだろうかという点について，「エンドユーザのための」という枕詞を置いてお話いただきました鷹尾さんにもちょっとご意見を伺いたいんですが，もし時間が要るようでしたら，別の質問を受けてから後でお伺いしたいと思います．

ほかにご質問いかがでしょうか．

菊野(広島大学) 紫合さんにお尋ねしたいんです．先ほど，例集合から一般化規則があって，一般化規則から最終的にプログラム言語が出るというお話がございました．またその具体的な例として交換システムの話があったと思うんですけれども，その具体例を見ておりますと，きつい言い方をしますと，一般化規則でもうでき上がっているものを単に図形として書いて見せただけではないかという印象が非常に強かったんです．Programming by Example というのは非常に難しいというご指摘がありましたが，たとえば先ほどの例の場合に，完全な例集合というものを本当に与えることができるのか．もし，例集合を与えて，それに沿って一般化規則をジェネレートして最終的にプログラミング言語をつくったというのであれば，例集合の例をお見せいただきたいんですけれど．

紫合 先ほどの例では，われわれはまだ例集合から一般化規則の生成をやっていません．われわれがこれからやろうとしていますのは，例集合というのは，あの場合ですと電話機を上げてダイヤルをまわしたら相手のベルが鳴るとか，そのときツーという音が聞こえたとかいうようなものです．そういうのを思い付くままいくつか与える．さらに，そういう全体的な流れとは別に，たとえば電話をずっと外したままにしておくとハウリング音を鳴らすというような一般的な規則を付け加えていくというのが実用的かと思います．すなわち完全な例集合を最初から与えるのではなく，いくつかの主な例とローカルな一般的規則から，全体的な規則，たとえば状態遷移図を生成し，さらに足りない部分を追加していくというのが現実的かと思っています．

菊野(広島大学) 今の話と若干関連があるんですけど，Visual Programming というのはむしろ最近ソフトウェア工学のほうで言われています Rapid Prototyping にある意味で非常に向いているんではないかと常日ごろ感じているんですけど，そういうご指摘がパネリストのどなたからもありませんでした．もしご意見があればお聞かせいただきたいんですが．

司会 次々にいいご質問をいただきまして，ありがとうございます．さっきの質問，鷹尾さんがご用意いただいていると思いますので，そのお答えを聞きながら，いまのご質問のお答えをご用意いただきまして，その後でお答えいただこうと思います．

鷹尾さん，お願いします．

鷹尾 私自身，回答をもっているわけではないんですけど，実用性に関して大きくプロフェッショナル・プログラマとエンドユーザに分けた場合に，むしろプロフェッショナル・プログラマ用の Visual Programming 環境のほうが比較的近い将来に役に立つものが出てくるのではないかという気がします．本当にソフトウェアの生産性が上がるかどうかというのは分かりませんし，それはきちっと評価しないといけないことだとは思うんですけれども．ただ一つのアナロジとして，バッチからインタラクティブなシステムに移行したときのことがあげられると思います．つまり，インタラクティブなシステムになってずっと使いやすい環境になったんですけれど，そのために，たとえばきちっとチェックもせずにコンパイルをするというようなことが起こっていて，本当に生産性が上がっ

たかどうか疑問なケースもあったと思います．けれども，私はそれはそれでいいと思っています．つまり，プログラマにとって使い勝手のよい環境ができたということで，長期的に見ればきっとそれは生産性向上に結び付いているだろうという多少楽観的な見方をしています．ということで，私の心配はむしろエンドユーザにあります．

プロフェッショナル・プログラマですと，プログラムをつくることが仕事ですから，多少問題があっても文句を言いながらでも使っていくわけです．ところがオフィスのエンドユーザというのは非常に気まぐれですから，自分が気に入らなければ，あるいはちょっと使い勝手が悪いと思えば一切使いません．そういう意味でエンドユーザ向けの Visual Programming 環境がどこまで浸透し得るかという点については多少不安がありまして，まだ確信がもてない状態です．

司会 非常にユニークなお答えをいただきましてどうもありがとうございました．途中でプロトタイピングの話にいくのかなという気もいたしましたけれども．(笑い)

それでは，先ほどのご質問で，Visual Programming というのを現実的観点からプロトタイピングという概念で捉えてもいいんじゃないかというご指摘がございましたので，それについてどなたからお答えいただけるでしょうか？

繁合 Visual Programming というのは，ある意味で非常にプロトタイピングに向いていると思います．まず性能が出ないでいいとか，(笑い) おもちゃみたいなものでもいいとか．現に，まずプロトタイピングには使えるだろうと思っています．

司会 どうも，非常に明快なお答えだったようですが，(笑い) もう一つ．

平川 たとえば，アイコニック・プログラミングを考えた場合に，プログラムがアイコンの形で分かりやすく提示されていますので，プログラム自体がドキュメントとしての役目を兼ねることができると思います．そういった意味で，Visual Programming は，私はプロトタイピングというよりもメインテナンスに有効ではないかと考えているんです．

細谷 その問題は多分性能の問題に帰着するんじゃないかと思います．だからプロトタイピングというのは，すぐつくれるけど性能的には後でつくり直さなければだめだという話になるんではないでしょうか．そうすると，平川先生の例で，たとえば画面のサイズが変わったり，少し機器の構成が変わったりしたときに，それをユーザに意識させないでシステム側でインタラクティブに全部やってくれるというのがここでいうプロトタイピングではないか．ところが実際の世界では1ステップでも削るといったような，少しでも速くしたいという要求があるわけです．その場合に，そういうパラメータをどうやって指定するのか教えていただきたいと思います．

平川 それについては，ご指摘のとおり非常に難しい問題だと思うんです．たとえばパラメータの受け渡しを考えた場合に，データの抽象度をできるかぎり上げておいて，ユーザにはできるだけ意識させないで，なるべくシステムのほうで受け渡しの管理を行うような仕組みが最低限必要になるだろうと思います．具体的にどういう形で実現するかということについてはいまここではお答えできないんですけれども，将来的にはそういった方向に必ず向かなければいけないだろうと考えています．

細谷 一種の仮想化ですか．

平川 そうですね．仮想化という形が必要だと思います．その辺は，市川先生が最初に「こういったことで締めくくろうかな…」と言われたソフトウェア工学の話とも絡んでくると思うんです．

司会 まだ時間は十分ございますので，続いてフロアからご質問をお受けしたいと思います．

松下(北海道開発コンサルタント) 細谷さんにお伺いしたいんですが，たとえば FORTRAN とか，COBOL, PL/I といった既存の汎用言語でつくったプログラムの必要な部分を，ソースそのままでデバッグするということは考えておられるんでしょうか．

細谷 試作版は Ada 用ですが，他の言語にも同じ手法が適用可能です．

松下(北海道開発コンサルタント) Ada のみでつくったプログラムのデバッグということですね．

細谷 原理的にはどの言語でも可能ですが，現在のシステム自身は Ada のみを対象としています．ただ，ユーザのデータの表示形式を定義する部分については，どんなタイプの言語にも使えるようになっております．

汎用にするという観点からは，プログラムコードのレベルでなく，フローチャートレベルで動かすという方法が考えられます．しかし，デバッグ時には，フローチャート上でエラーの場所が分かったとしても，結局はソースを直さなければいけないということで，

プログラム言語の意識が必要です．言語独立でデバッグするには，Visual Programming，つまりもともと記述するときから Visual で汎用に記述するという形をとらなければ難しいと思います．

司会 ありがとうございました．ほかにご質問ございますでしょうか？

住田(NTT 通研) アイコニック・プログラミングの話，大変興味深く聞かせていただいたんですけど，私はプログラミングとかソフトウェアの分野で働いているものではなくて，どちらかというとモデリングの話というか，そういうのに興味をもっているものです．アイコニック・プログラミングの話を聞いたときに，なんというか，モデリングの話にかなり近いんじゃないかという気がしました．絵をいくつか用意して，その組み合わせでなにか自分のやりたい仕事を表現するというふうに理解したんですけれど，多分そのときに重要なのは絵としてどういうエレメントをもってくるかということで，一番少ないエレメントでたくさんのものが記述できるのが一番いいと思うんです．ところが，そうするとアイコンの選び方によって全然表現できないものが出てきて，結局ユーザがもっともプリミティブなものを改めて定義しなければ表現できなくなってしまって，もともとというんですか，せっかくアイコンを用意した意味がなくなってしまうような気がします．そういうことはないんでしょうか？

それからもう一つ，絵の組み合わせについても，普通の言語の場合の文法みたいなのがあると思うんですが，そういうのはどういうふうになっているんでしょうか？

司会 それじゃ，お答えをお願いします．

平川 簡単にアイコンが定義できるかということが関連すると思うのですが，確かにそこが非常に重要だと思います．東芝の松村さんのやられたご研究なんですけれども，実際にアイコンを使ったシステムを作成されておられまして，アイコンをユーザに見せて，その意味がどの程度分かるかということについて実験をされた結果です．それによりますと，ものを表しているアイコンについては，そのアイコンの意味を正しく伝えるアイコン・イメージが定義できると言えます．

これに対して，オペレーションを表しているアイコンについては実際に目に見える形がないわけで，そういったものをアイコンとして表現しようとすると，良くて7割ぐらいの理解度にとどまっている．そういったデータが得られているわけです．

ですから，なんの支援もなしにアイコンの内容を正しく認識させるようなアイコン・イメージを用意するというのは，特にオペレーション・アイコンについては難しい．その辺につきましては，一番現実的なレベルでいえば，ユーザとの対話的手段を持ち込むことによってなんとか回避できないかなと思っています．その次のレベルとしては，知識ベースシステムを持ち込むことによってシステムが状況に応じてアイコンの解釈を動的に変更する，そういったことができればありがたいなと思っているんです．

司会 どうもありがとうございました．それではまた新しいご質問いただきたいと思います．

南川(東芝) 私は，しばらく前にご質問なさった牧之内さんほどペスミスティックには考えてないんですけれども，ただ，こういった試みを成功させるためには，制御部というものがちゃんと定義できるかどうかが条件になるだろうと思うんです．いままでの成功例でも分かりますように，コンクリートな概念をビジュアルに表すというのはうまくいくと思うんですけれども，制御のところがどうしても引っ掛かってしまう．そういった意味で紫合さんにちょっとお伺いしたいんです．ステートとステートの間の差分という言葉を使われたと思うんですが，そういう差分のようなものを軸にしてうまくやっていくといったようなアイデアがおありなんでしょうか？

紫合 先ほどの平川さんの説明にもありましたように，対象の絵は描きやすいんですけれども，操作の絵というのは描きにくい．それで，たとえば電話機の絵は描けるんですけど，電話を上げるという絵は描きにくい．だけど，電話が下がっている絵があってその次に電話が上がっている絵があれば，電話を上げるという操作を表現できるわけです．また，たとえば電話にベルがつながっている絵があって次の絵ではベルがなくなったとします．さらにベルを鳴らす装置の仕様を見ますと，鳴っている状態から鳴ってない状態に至るにはストップベルという信号を受け取るというようなことが書いてある．それらをもとにしまして，電話のベルが鳴っている状態から鳴ってない状態にするにはストップベル信号を出してやればよいことが分かるわけです．

このように各オブジェクトの状態を絵にしまして，それが変わっていくようすを絵で描きます．絵が変わるにはなんらかの操作が必要ですが，その操作を二つの絵の差分から生成することができます．これは交換

機なんかの場合には比較的うまくいく．というのは，仕様の対象となるオブジェクトの種類が比較的少ないからかもしれません．

最近の抽象データタイプやオブジェクト・オリエンティッドによる場合でも，オブジェクトを表す丸やボックスが線で結ばれたスタティックな絵だけでは分かりにくいですね．それを説明するときは，もっとダイナミックな絵を描くことが多い．ただし，ダイナミックな絵では抽象的なものは少ないですね．スタックなんかは比較的抽象的な標準的な絵がよく描かれます．抽象的な絵があるとその概念がまとまっている．逆にきれいな概念に対しては抽象的な絵が描けるんじゃないかと思っております．

司会 非常にいいお答えをいただきました．どうもありがとうございました．

中前(広島大学) プログラミングについては全く素人で，非常にとんちんかんな質問をすることになるかと思うんですけど，図形情報と文字情報というものを考えた場合に，図形情報の非常に大きな特徴というのは具象的であって，直感的であって，二次元的である．だから，初めに全体図を見て，それから後で詳細を見る．それに対して，文字情報というのは論理的であって，抽象的であって，一次元的な情報である．したがって，文字情報だけでなにか処理をしようとすると非常に難しい問題であれば誤解を生じる．ところが，一方，図形情報で非常に複雑なものを表現しようとすると，そこで錯覚が起きたりするんじゃないかという気がいたします．

ところで，シンボリックなものがだんだん抽象化されて文字になったという歴史的な経過からみますと，二つのよさを兼ね備えたようなプログラミングというのはないんだろうか．その辺は，どうなんでしょうか？

司会 いまの中前先生のご質問，ノン・プログラマという立場をお取りになりましたので，先ほどエンドユーザ向きというような枠組の中で非常に多角的な考察をなさいました鷹尾さんにお答えをご用意いただきたいと思います．

鷹尾 私は，先ほど申し上げましたように，ビジュアルがただちに絵を意味するものとは思っていません．スプレッドシートみたいなものも十分ビジュアルだと思っております．枠組はグラフィックスタイルだけど中身は完全に文字情報なんです．もちろん，ユーザ・インタフェースレベルではアイコンはかなりうまくいっていますけれども，プログラミングに関してはこれから模索していかないといけないと思っています．ただ，少なくともいままでの経緯を見ているかぎり，全面的に絵で置き換えるということではなくて，かなり段階的に進まなければいけないんではないかというふうに感じております．

それから，文字か絵か…という話なんですけれども，文字といってもアルファベットと漢字とではかなり違うと思います．実際，私たちはインタラクティブなシステムをつくってアイコンのメニューを用意したのですが，非常に評判が悪くて，これだったら漢字で書いてくれたほうがよっぽど分かるといわれたケースがあります．

漢字は，ある意味で文字と絵の特性を兼ね備えているからかも知れません．現在は，文字から絵という連続したスペクトラムがあって，そのどこかに目的に応じて最適な点があり得るんではないかというふうに感じています．いまのところ，プログラミングに関しては，かなり文字寄りの世界ではなかろうかというのが私個人の印象です．

おわりに

司会 どうもありがとうございました．質問をお受けするのが予定より若干遅れましたため，まだ不十分かも知れませんが，この辺で打ち切らせていただきます．始めに予告いたしましたようなきちんとしたソフトウェア的な考察を行う時間がございませんので，それはご勘弁いただいて，一言雑な感想を述べさせていただきたいと思います．

つまり，Visual Programming というのは，見やすい，使いやすい，親しみやすい…．「見やすい」というのは，広島では必ずしもビジュアルという意味ではないようでございまして，「やさしい」，「どうってことない」という意味だそうでございますけれども，(笑い) …とにかく，見やすい，使いやすい，親しみやすいということから，エンドユーザ向けというような逃げも用意されているなんてことを申しました．それに対して，どこまでプロフェッショナルユーズに耐えられるようになるだろうかということも再三申しました．これについては皆さんにも随分と関心をおもちいただいて，またパネリストの方々にもお知恵を絞っていただきましたけれども，いまのところあまりはっきりした解答が出ていないようです．

しかし，ここに座りますと，ペスミスティックな立

場はとれないわけです．無理しても楽観的な立場をとらなければいけませんので，(笑い) とにかく，先にはプロフェッショナルユーズにも耐えられるようになるものといたします．そうしますと，“しつけ”，“しつけ”とこれまでのソフトウェア工学でいっていましたようなことではなくて，プロフェッショナルな世界にも新しい遊びの要素が入ってくるんじゃないかと思います．そしてその結果ソフトウェアの質が変わるんじゃないか，あるいはソフトウェアが提供する仕事の質も変わるんじゃないか．なにかが変わることを期待して，皆さまにはエンドユーザ向けとプロフェッショナルユーズとの間の現実的なギャップをなんとか埋めるようなアイデアを出していただきたいし，またお力をお借りしたいと考えております．

大会最後の日の，しかも終わりの時間に大分近づいてまいりましたけれども，大勢討論に参加していただきまして本当にありがとうございました．では，これで終わりたいと思います．

総合司会 それでは，最後に拍手でもってパネリストの皆さんに感謝の意を表して終わりたいと思います．どうもありがとうございました． (拍手)