プログラム可視化システム†

市川 至†† 小野越夫†† 毛利友治††

本論文では，プログラム可視化システムについて述べる．まず，プログラム可視化システムの背景を説明し，次に，プログラム可視化手法と，プログラム可視化システムの基本構成について述べる．特に，本システムで用いるプログラムと図形との対応方法と，システムの基本アーキテクチャとしてのトレーサ埋め込み方式，システム内部に作られる内部図形構造について示す．最後に，対象プログラミング言語を ESP として，PSI 上に作成した実験システム上での可視化実行例について述べる．

1. はじめに

ソフトウェアの再利用や，設計とプログラムの一貫性の確認を支援するために，そのソフトウェアの動作を理解することが必要となってきている．従来は，文字情報を用いた方法がとられていたが，ワークステーション技術の発展により，図形情報を有効に使用できるようになってきており，図形的表示によるプログラムの構造の理解や，アニメーション表示によるプログラム動作の理解を促進する研究がなされている[1]．

このひとつのアプローチとして，ビジュアル・プログラミング環境がある．これは，図形情報を利用してプログラミングを行うことをめざしたものであり，図形的なプログラミング言語や仕様記述言語などが用いられる．例えば，PegaSys[2] は，図形を用いた記述を形式的なプログラム設計書とするものである．また，別のアプローチとしては，VIPS[3] や PROEDIT2[4] のようなビジュアル・デバッガがあげられる．これは，プログラム動作をプログラミング言語レベルの抽象度の図形を用いて表示するものである．

我々のアプローチに近いものとして，アルゴリズム・アニメーションがある．これは，プログラムの動作を図形を用いて表示することで，処理アルゴリズムを理解させるものである．例えば balsa[5] では，各種のソートアルゴリズムについて，その処理の模様を表示し，動作原理や効率を比較して理解させる．

我々のプログラム可視化システムは，ソフトウェアの再利用や，設計とプログラムの一貫性の確認を支援するために，設計段階で用いられる抽象度を持った図形を用いて，プログラムの実行動作を可視化するものである．この図形の抽象度は，PegaSys などで使用されるような図形と同じであり，VIPS, PROEDIT2 のものよりも抽象度が高く，ユーザによるプログラム動作理解がはかられるものと考えている．さらに，プログラム可視化システムは，複数の異なったビュー (View，視点) からの同時の可視化をサポートして，人間の理解をより促進する．

このように，プログラムの動作を設計段階の図形を用いて理解できることで，プログラムと設計との一貫性の確認支援に利用でき，また，設計の検討支援にも利用可能である．さらに，将来，プログラム部品の動作理解を，可視化の技術を用いて行うことによって，プログラムの再利用を支援することも考えている．

本論文では，プログラム可視化システムについて述べる．まず，プログラム可視化システムの背景を説明し，次に，プログラム可視化手法と，プログラム可視化システムの基本構成について述べる．特に，本システムで用いるプログラムと図形との対応方法と，トレーサ埋め込み方式，内部図形構造について示す．最後に，実験システム上での可視化実行例について述べる．

2. プログラム可視化手法

2.1 プログラム可視化の環境

本研究では，**図1**のような環境を仮定している．仕様記述や設計の段階において，図形が設計図として何らかの形で使用されており，その仕様・設計からプログラムが作成され，このプログラムと元となった設計図との間の対応関係が何らかの形で用意されている．この設計段階での図形を利用して，この対応関係を用いて，プログラムの動作を可視化する．

これにより，我々の可視化システムは，**図2**のように，(1)対象プログラム，(2)設計図の定義，(3)図

---

† A Program Design Visualization System by ITARU ICHIKAWA, ETSUO ONO and TOMOHARU MOHRI (FUJITSU LIMITED).

†† 富士通(株)

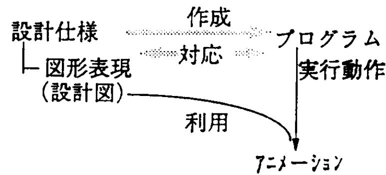


図 1 プログラム可視化の環境
Fig. 1 The environment of the program design visualization.

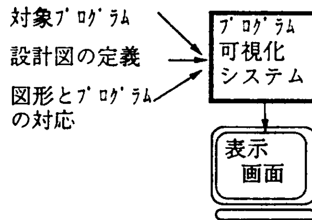


図 2 プログラム可視化の入出力
Fig. 2 I/O of program design visualization.

形とプログラムとの間の対応関係が入力として与えられ，出力として，動画表示が得られるものとなる．

プログラムが意図した動作をしていない場合に，それがプログラムに原因するものかどうかが問題である．我々は実験で，正しく動作し可視化が行われているプログラムにバグを埋め込み，異常な表示が行われることを確認した．本論文では，この問題については次のように仮定する．

[仮定-1]

本プログラム可視化システムは，与えられた正しい入力に対しては正しい動作をすることを保証するが，対応関係の正当性については保証するものではない．

対応関係についての正当性については，プログラム可視化システムに与える前に検証されているものとする．

2.2 対応関係

対応関係は，プログラムの世界と図形の世界を結びつけるものである．本研究での対応関係は，静的な対応関係と，動的な対応関係に分けて定義する．プログラムと図形との対応は，「可視化指示」により定義される．可視化指示は，動作の対応をとる「可視化命令」と，要素の対応をとる「写像定義」からなる．

写像定義は，プログラム中の要素を表す表現式 p と，図形の要素を表す表現式 g とを「$p \rightarrow g$」の形で定義したものである．ここで，p は，プログラムでの表現式で表し，g は，設計図の中でのラベルを用いる．同種ラベルの図形が複数の設計図に現れるときは，どの図の図形であるかを区別するために，設計図のラベルを付加する．

動的な対応関係は，設計図上での図形の動作と，プログラム上での実行動作との対応を定義する．我々は，プログラム上での実行動作列と，設計図上の動作列とについて，それぞれの部分列の間の写像関係を定義することにした．

可視化命令は，プログラム中の実行動作列 S と図形構造での可視化動作 A との対応を，Hoar の公理形式「$P\{S\}Q,\ A$」で定義したものである．ここで，P, Q はそれぞれ，S の実行の直前，直後に成立すべき，プログラム状態に関する陳述（条件式）であり，その意味は「S の実行前に P が成立し，実行後に Q が成立するならば，A が行われる」と解釈する．

実行動作列 S の定義は，

1) $Si{*}{*}{*}Sj$: Si の実行後に Sj が行われる．

2) $Si\langle{*}Sj{*}\rangle$: Si の実行中に Sj が行われる．

の2つを基本として構成される．

また A は，設計図の世界における，設計図中の図形要素の変化を図形の世界の記号によって記述したものである．ただし，A の中で，P, S, Q に出現するプログラムの世界での記号をそのまま用いることができ，その場合には，写像定義によって図形の世界に写像されて解釈されるものとする．

2.3 プログラム可視化のアーキテクチャ

プログラム可視化は，対象プログラムの動作を設計図の上での動画として表示するものである．対象のプログラムそのものは実際には実行せず，設計図の上での動画のみを生成する，疑似実行のアプローチもあるが，我々は，対象プログラムを実際に実行し，その実行動作を抽出し解析して設計図上で動画を表示するアプローチをとる．

このアプローチに基づき，対象プログラムの実行をかなり細部まで実時間で可視化するとなると，まず問題点として，実際の実行に追いついて可視化表示が行えるかどうかがある．次の問題点として，もしそれが可能であったとしても，そもそも，そうした表示に，人間の理解が追いつけるかどうかがある．本来，可視化することによって，プログラム動作の理解を支援することが目的であるので，可視化した結果は，人間の理解できる速度で変化させて表示するべきである．このため，計算機上での実行速度をどこかで人間向きの速度に時間を引き伸ばすことが必要になる．

この時間の引き延ばしを行う基本方式は大きく分けて次の2つの方法が考えられる．

（a） 対象プログラムを実行して実行動作の情報を検出し，この情報の時系列を「ログ」として二次記憶上に記録しておく．次に，このログをもとにして，実行とは独立に，時間を引き延ばして可視化を行う方式．

（b） 対象プログラムを可視化画面と同期をとりつつ実行させ，その実行動作の情報を検出して，この情報をもとに時間を引き延ばして可視化を行う方式．

プログラムの論理的動作を可視化する場合には，時間の引き伸ばしの不均一さは，さして問題とはならない．しかし，対象のプログラムの同期，効率などの性質を可視化する場合には，時間の引き伸ばしが均一である必要があり，さらに，可視化をすることによって対象プログラムの実行動作に影響がないようにしなければならない．本論文では，論理的動作を可視化することに主眼を置くことにし，後者（b）の方式をとることにする．

この方式の元でプログラムの実行動作を可視化する基本アーキテクチャとしては，次の3つの方法が考えられる．

（1） まず，対象言語のインタプリタ，シミュレータといった実行環境を用意し，この上で対象プログラムを実行し，実行動作の情報を得て，これを解析し可視化する方法が考えられる．この方法は，より詳細な情報を得ることができるという利点があるが，実行情報を抽出するための実行環境を作成する必要がある．

（2） 次に，対象プログラムに対して，対象プログラムに可視化を行うプログラム素片を直接埋め込む方法も考えられる．この方法は，埋め込みを行った対象プログラムの実行にともない直接的に可視化を行うので，他の2つの方法と異なり，実行動作の情報について解析する必要がないという利点を持つが，複雑な実行動作に対応できないという欠点を持っている．

（3） これら2つの中間的なアーキテクチャとして，「トレーサ埋め込み方式」が考えられる．これは，まず対象プログラムに対して，実行情報を収集するためにソフトウェア・プローブとしてトレーサを埋め込む．このトレーサを埋め込んだプログラムを実行してトレーサから実行動作の情報を得て，解析し可視化する方法である．この方法の利点は，実行環境全体を作成する必要がなく簡単に実現できる点であるが，（1）と比べると必ずしも詳細な情報が得られないという欠点がある．

本プログラム可視化システムでは，実現が比較的容易で，かつ，複雑なプログラム動作にも対応可能な（2）のトレーサ埋め込み方式を可視化基本アーキテクチャとして採用した．

図3にトレーサ埋め込み方式のプログラム可視化システムの概略を示す．この方法は，実行情報を収集するトレーサを対象プログラムに埋め込み，実行動作を検出し，これを前節で述べた可視化指示を用いて解析し，画面に図形の動画として表示する．対象プログラムの実行動作列の検出は，いくつかのプログラム上の点の制御の通過によって観測するため，対象プログラムと解析・表示機構とがコルーチン的に動作して相互に同期がとられる．

2.4 設計図の構造

設計図は，「絵素」と呼ぶ単位図形から階層的に構成される．絵素についての定義は「絵素辞書」に格納され，設計図の定義は「設計図ライブラリ」に格納される．これらの定義は，各図形の形状，操作，構成それぞれについての定義からなる．

設計図の定義では，部分図形として扱われる設計図を用いることができ，こうした設計図を，全体の設計図に対して，部分設計図と呼ぶ．

絵素辞書では，スタック，リスト，キューといった，ソフトウェア設計でよく使われる抽象データ型については，基本要素としてシステムが提供している．

設計図ライブラリでは，設計図は，絵素，および，他の設計図（すなわち，部分設計図）を自分の部分として参照して非回帰的に定義される．この参照関係によってできる，各絵素と設計図の関係を「概念的構造」と呼び，この概念的構造で極大となる設計図を「トップレベルの設計図」と呼ぶことにする．

各設計図は，それぞれ固有のビューに対応している．ここでビューとは，対象のプログラムをどのような観点や視点から見ているかを表すものとする．例えば，対象プログラムについて，それをデータ・フロー

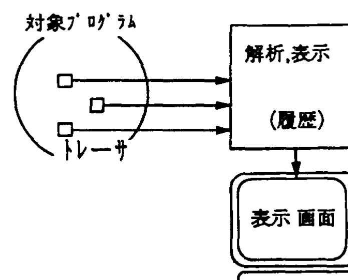


図 3 トレーサ埋め込み方式
Fig. 3 The tracer embedding method.

的な視点から見た場合の設計図と，コントロール・フロー的な視点から見た設計図の場合では対応するビューは，異なる．本システムでは，対象プログラムに対して複数の視点からの設計図を設定でき，これに対応して，対象プログラムに対して複数の多重（多次元）的なビューを設定することができる．同様に，設計図の詳細度，抽象度の差によっても，ビューは異なる．本システムでは，対象プログラムに対して複数の設計図を設定でき，これに対応して，対象プログラムに対して複数の階層的なビューを設定することができる．

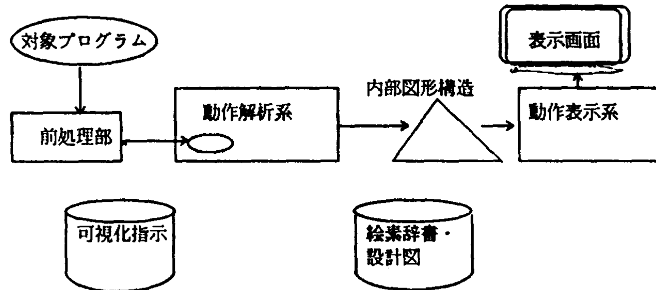


図 4 プログラム可視化システムの基本構成
Fig. 4 Basic configuration of the program design visualize system.

3. プログラム可視化システム

論理型プログラムは，従来のプログラミング言語による記述に比べて，その実行動作が，プログラムを見ただけでは理解しにくい．そこで，我々は，論理型オブジェクト指向言語 ESP[6] を対象言語として，プログラム可視化システムを開発した[7]．このプログラム可視化システムの基本構成の概要を**図 4** に示す．

システムの構成は，「前処理部」，「プログラム動作解析系」，「内部図形構造」，「プログラム動作表示系」からなる．このほか，可視化される対象プログラムに対して，可視化のための知識として，前章で述べた可視化指示，絵素辞書・設計図ライブラリが与えられる．

3.1 トレーサ埋め込み処理

可視化システムは，前処理部によって処理された対象プログラムの実行動作を，プログラム動作解析系において検出・解析し，表示図形のモデルである内部図形構造に対する可視化動作を合成し，プログラム動作表示系によって表示画面上のアニメーションを行う．

可視化命令で定義される動作列を検出するのに，必要かつ十分なトレーサを埋め込む処理を行うのが，「前処理部」である．前処理部でどのようなトレーサを埋め込んだかについての情報は，プログラム動作解析部に伝えられる．

プログラム動作解析系では，対象プログラムの実行動作列を検出し，可視化命令で定義された実行動作列が実行されると，内部図形構造に対する可視化動作を合成する．

可視化命令は，「$P\{S\}Q, A$」の形式で，プログラム中の実行動作列 S と図形構造での可視化動作 A との対応を定義している．本システムでは，この P, S, Q についての実行動作情報を得るためのトレーサを前処理部で埋め込む．それらのトレーサからの実行動作情報を動作解析系で解析し，S にあたる実行列が得られると，その実行列のもとで，S の最初と最後の時点で，それぞれ P と Q の評価を行う．P, Q ともに成立するならば，A から可視化動作 a を生成する．

写像定義は，「$p\rightarrow g$」の形式で，プログラム要素 p と図形の要素 g との対応を定義している．本システムは，この定義を用いて，解析の結果得られた a の中のプログラム要素を図形の要素に写像し，図形の上の可視化動作 a' を得る．

前処理でのトレーサの埋め込みかたは，まず，実行動作列を定義する可視化命令の S を解析し，埋め込みの候補点をすべて列挙する．次に，プログラムの制御構造から不要な候補点を削除したり，トレーサをマージするなどして，埋め込まれるトレーサの数を最小にし，効率的に実行動作を検出するようにして，トレーサを埋め込む．

これらの処理を**図 5** の例を用いて説明する．ここで図 5 の (a) は，対象プログラムであり，(b) はそれに対する可視化命令であり，その意味は，

「$p(X)$ の実行中に $r(X)$ が行われた直後に，$q(X)$ が

```
.................................
p (a) :- ..................................................;
p (f (Y)) :- ........, r (f (Y)) , ......;
..........................
r (X) :- ...................................... ;
s (X, Y, Z) :- r (X), r (Y), r (Z) ;
..........................
g (X) :- p (X), ......................... ;
..........................
```

(a)

```
............
{p (X) <* r (X) *>} q (X) ; a (X)
............
```

(b)

図 5 可視化命令の例
Fig. 5 An example visualization command.

成立すると，$a(X)$ を行う」

である．トレーサの埋め込み候補点は，S 中に出現するすべての述語やメソッドについて；

1)呼び出しの直前，2)呼び出しの直後．

それらを定義する各節の本体について；

3)本体の先頭，4)本体の末尾．

もしその本体に，「！」カットがあれば；

5)カットの直後．

となる．明らかに，これらの候補点すべてに埋め込む必要はない．

カットについての候補点は，論理プログラム特有の制御であるバックトラックの処理のために用意されている．バックトラックが発生した場合には，それに従って表示画面も元に戻す必要があり，各トレーサにはバックトラックが発生したという実行制御の事象を通知する機能がある．また，動作解析系では，状態の履歴を持ち，バックトラックの発生によって，表示画面を戻すことを可能にしている．バックトラックがカットを通過する場合には，その本体の先頭から，カットまでの間に通過したトレーサはバックトラックが発生したことを動作解析系に送ることができないので，カット直後のトレーサでこれを代行するようにする．

以下，図5の場合について，可視化命令にそって各候補点にトレーサを埋め込む処理について**図6**を用いて説明する．$p(X)$ の実行中とは，(1)と(2)，(3)と(4)のそれぞれ対になるトレーサの間であり，$r(X)$ の終了する時点は，(5)～(8)のそれぞれのトレーサの時点である．そこで，$p(t)$ の実行が行われる(1)と(2)，(3)と(4)のそれぞれの間で，$r(t)$ となる(5)～(8)のいずれかを通過した場合に，$q(t)$ の判定を行い，それが成立するならば，$a(t)$ を発行すればよい．このように，各可視化命令について候補点の組合せで表し，使用しない候補点は除外する．

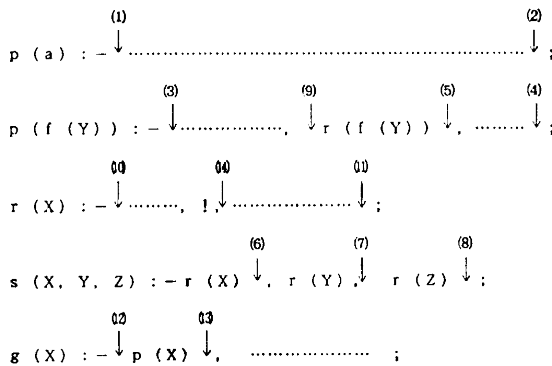


図 6　埋め込みの候補点

Fig. 6　Candidate points for the tracer embedding.

実はこのプログラムでは，$p(X)$ の実行中に $r(X)$ を実行する可能性のあるのは，第一節での $r(f(Y))$ の呼び出しのみであり，(5)のトレーサの時点でのみ検出すれば十分である．このように，プログラムの制御構造について，and-or 木を予測して解析を行い，発生しえない候補点の組合せを除外する．あわせて，各候補点でどの値を拾うかについて設定する．

埋め込み候補点の組合せから，どのトレーサによってどのように遷移するかという各オートマトンの定義を作成する．これらのオートマトンについて次節で説明する．これらのオートマトンの最適化によって，さらに埋め込みを最適化する．

最後に，トレーサを節内で問題のない範囲で移動やマージするなどして，埋め込みの最適化を行う．これまでの最適化処理は，機械的に行っているが，どうしても大局的・発見的な最適化処理が不可能であり，人間の手でさらに最適化を行っている．この点が将来の課題である．

3.2　プログラム動作解析

前処理によって埋め込まれたトレーサの上を制御が通過することで，トレーサは，自分を通過したことのメッセージを，指定された変数のそのときの値をともなって，動作解析系に対して送る．動作解析系では，このメッセージに対して，まず，自分が行う解析処理によって制御や変数の値に影響がないように，絶縁処理を行う．

次に，動作解析系では，このメッセージの情報を履歴とともに解析する．このため，動作解析系では，各可視化命令ごとにオートマトンを生成し，履歴を分散管理している．

可視化命令 $P\{S\}Q\,;A$ に対応するオートマトンは，S に関連するトレーサからの通過を通知されることによって，状態が遷移する．S の最後に相当するトレーサの通過によって，このオートマトンは最終状態に達する．各オートマトンは，最終状態に達すると，P, Q の評価を行い，両方が成立すると，A から可視化動作 a を生成する．

また，これらの遷移は，トレーサ上での制御の正方向の通過だけでなく，逆方向のバックトラックによる通過によっても行われる．例えばある状態まで遷移したオートマトンに，すでに通過したトレーサからバックトラックで通過したことが告げられると，このオー

トマトンは，元の状態に遷移することで，履歴を戻す．

一方，プログラムの制御でくり返しや再帰を行うことによって，ある可視化命令の S にマッチする部分実行動作列が $s1, s2, \cdots, sn$ のように複数出現し，しかも互いに重なりあう「多重起動」の状況がしばしば発生する．前の si が終らないうちに sj も扱えるようにするのには，この可視化命令に対して単一のオートマトンを生成するだけでは，そのオートマトンが複雑なものとなってしまう．

そこで本システムでは，多重起動への対処のために，各可視化命令について，オートマトンのクラスを定義しておき，各 $s1, s2, \cdots, sn$ について，その先頭のトレーサの通過とともに，インスタンスのオートマトンをそれぞれ生成し，動作解析系が，トレーサからの情報を各オートマトンの状態から判断して，対応するオートマトンにふり分けて送るようにした．このように，複数のオートマトンとふり分ける機構とを持つことで，1つのオートマトンで行う場合に比べて，それぞれの定義が簡単になった．

また，動作解析系で生成した可視化動作 a には，プログラムの要素が含まれており，写像定義を用いてプログラム要素を図形要素に写像する必要がある．さらに，この可視化動作をメッセージとして送るべき相手を，次節で述べる内部図形構造中にある図形オブジェクトを求め，そこに送り付ける処理が必要である．そこで，**図 7** のように，この処理を行うための「分配器」を用意した．この分配器は，写像定義と内部に持つ写像テーブルを用いて写像とメッセージの分配を行う．

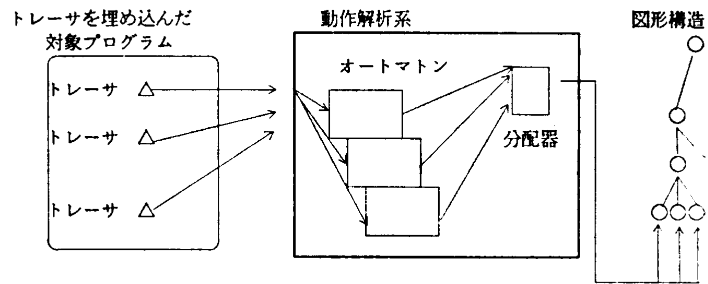


図 7 動作解析系の情報の流れ
Fig. 7 Information flow on analyzer.

3.3 内部図形構造

我々のプログラム可視化システムは，設計図からなる概念的構造の多次元・階層的なビューをサポートし，可視化された表示画面では，複数のウィンドウにそれぞれのビューにあたる図で同時にアニメーション表示される．このために，プログラム可視化システムには概念構造をシステム内部に実現した「内部図形構造」と呼ばれる表示モデルが用意されている．

内部図形構造の要素は，木構造で相互にリンクされた図形の能動オブジェクトであり，絵素辞書と設計図ライブラリの定義から作成される．構造のトップにある設計図のどれを表示するかが，システムの起動時の初期化の段階で選択され，選択された設計図は，自動的に対応する内部図形構造に図形オブジェクトを再帰的に作成する．

絵素に対応するオブジェクトは，分配器からの可視化動作のメッセージを受けると自分に定義されている操作，形状の定義から，図形情報を作り出し，自分の上位のオブジェクトに送る．

設計図に対応する図形オブジェクトは，**図 8** のように，下位のオブジェクトからの図形情報を受けとり，自分に定義されている構成の定義から図形情報を合成し，上位のオブジェクトに送る．

このほか，設計図に対応する図形オブジェクトは，あたかも絵素としてふるまい，自分に定義されている形状の定義から，図形情報を作り出し，自分の上位のオブジェクトに送るように切り替えることもできるようにする．この切り替えにより，画面上の設計図は，部分設計図の図形を合成／分解して見ることができるようになる．

また，設計図に対応する図形オブジェクトは，それぞれのビューに対応して合成した図形を，アニメータを通じてウィンドウに表示する機能がある．アニメータとの間は，「集配器」と呼ぶ機構で結ばれている．集配器での切り替えによって，**図 9** の(a)から(b)のよ

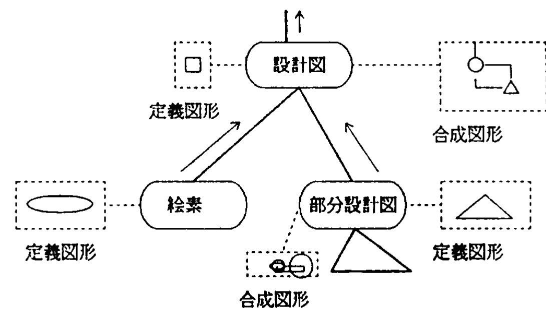


図 8 内部図形構造
Fig. 8 The internal graphic objects structure.

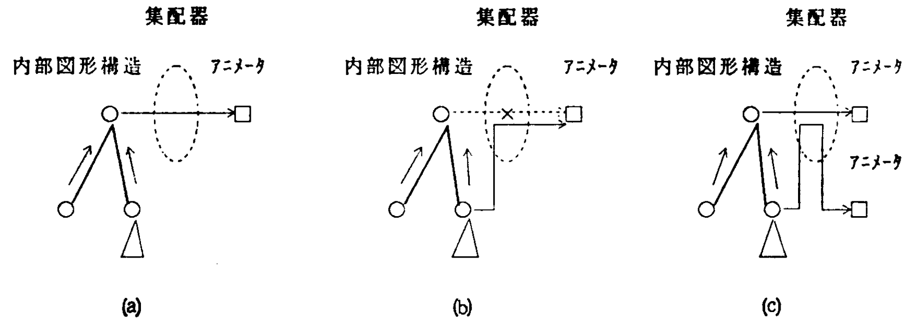


図 9 集配器によるビューの切り替え
Fig. 9 View switching using the distributer.

うに，上位の設計図を表示しているアニメータに下位の設計図を表示させるようにすると，下位にズームインすることになる．

同様に，図9(c)のように，アニメータを生成し，対応していない設計図を新たに対応させ，新しいウィンドウを開いた場合は，別ウィンドウでのズームとなる．このように，集配器での切り替えでズームし，詳細な情報を見ることができる．

また，設計図の定義においては，非回帰的に図が定義されているが，同一の図が複数の設計図に共有されている．先に述べたビューの切り替えや図形の分解／合成を行いやすくするために，内部図形構造では共有される構造はそれぞれに複写した構造を持たせ，オブジェクトに対する可視化操作のメッセージは，それぞれ複写したオブジェクトに複写して分配するようにする．このため分配器に，可視化動作のメッセージを複写・分配する機能を加えた．これにより，可視化操作は，内部図形構造の葉の方に送られ解釈され根の方に伝わっていく．

アニメータは，ウィンドウ画面へのアニメーション表示を行う．ズームの切り替えや図形の分解／合成を表示画面で図形を選択することで行うようにするために，画面で図形がピックアップされたことを検出しフィードバックする機能を持つ．

ズームの切り替えによる複数ウィンドウの表示は，ユーザの理解を支援するために有用な機能であるが，反面，画面が煩雑になる欠点がある．このため，ユーザが興味のあるときにだけ表示されるようにする必要があるが，これまで述べた本システム構成では，ウィンドウを表示するしないは，ウィンドウの生成や消去によって行われることになる．しかも，ウィンドウの生成／消去のみならず，アニメータ生成／消去も同時に行われる．

ユーザからのウィンドウを表示するしないの要求が頻繁に行われると，アニメータやウィンドウの生成／消滅によるオーバヘッドを無視することができない．そこで，本システムでは，このオーバヘッドをなるべく避けるために，いったん生成されたウィンドウやアニメータは，画面に表示しないよう要求された場合には，単に画面にウィンドウを表示しないのみとし，アニメータなどは消去せず存在しておくようにした．この状態で再び表示が要求された場合には，新たに生成する必要はなく，すでに存在するウィンドウを画面に表示するだけでよい．

4. 実験システム

論理型オブジェクト指向言語 ESP プログラムを対象とする，プログラム可視化システムを PSI ワークステーション上に試作し，可視化実験を行った．試作したプログラム可視化システムは ESP で書かれており，そのソース・サイズは，約 76 kB であった．

実験対象として，実用規模のプログラムである，並列論理型言語実行系の KL 1 ソフトウェア・シミュレータ[8]について，実際に設計で用いられた設計図を用いて可視化実験を行った．この例について，対象プログラムのソースのサイズは約 18 kB であり，必要とした可視化指示の個数 28 個，表示される設計図 11 枚，埋め込まれたトレーサ総数 69 個であった．

図 10 にその可視化実行例の一部を示す．

(a) 始動直後，KL 1 ソフトウェア・シミュレータの最上位のモジュール構成の設計図を表示した状態：この設計図では，9 台のプロセッサ (PE) と 1 つのネットワーク・マネージャが双方向に接続されている．

(b) 実行が進んだ状態で，一部のプロセッサの内部構成の設計図について，別ウィンドウに表示した状

態：プロセッサの内部は，スケジューラ（scheduler）とソルバ（solver）からなり，それぞれの間はキューで接続されている．スケジューラとソルバの間のキューにはデータがキューイングされており，大量の場合には「…」で省略表示されている．

（c） さらに省略表示されているキューにズームして内容を表示した状態：データについて区別する写像定義がないため，同じ図形で表示されている．ただし，通信されているデータとの区別の定義があるため，全体構成図では異なる図形になっている．

（d） プロセッサ内部構成の設計図を，別ウィンドウではなく，全体構成の設計図に表示した状態：みや

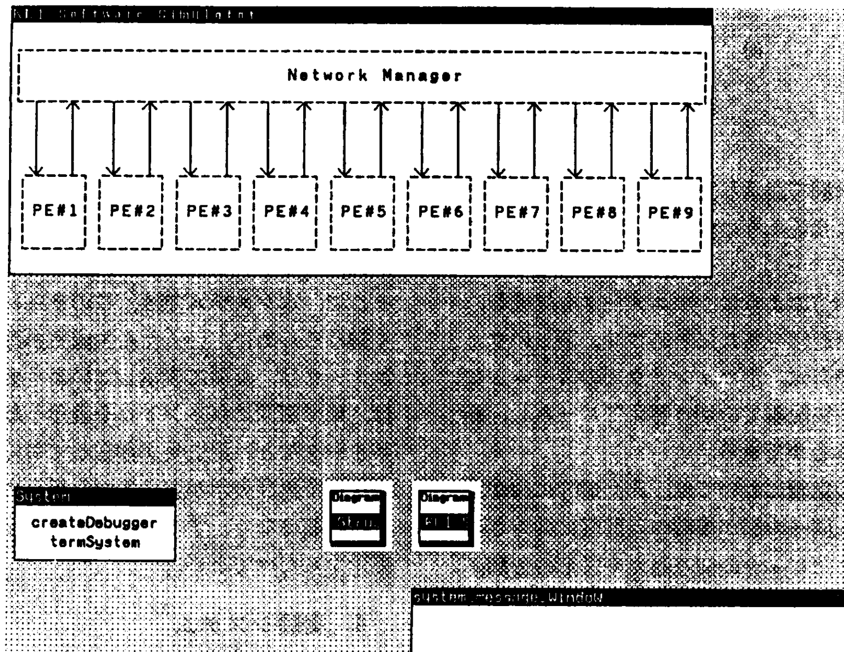


図 10 （a）

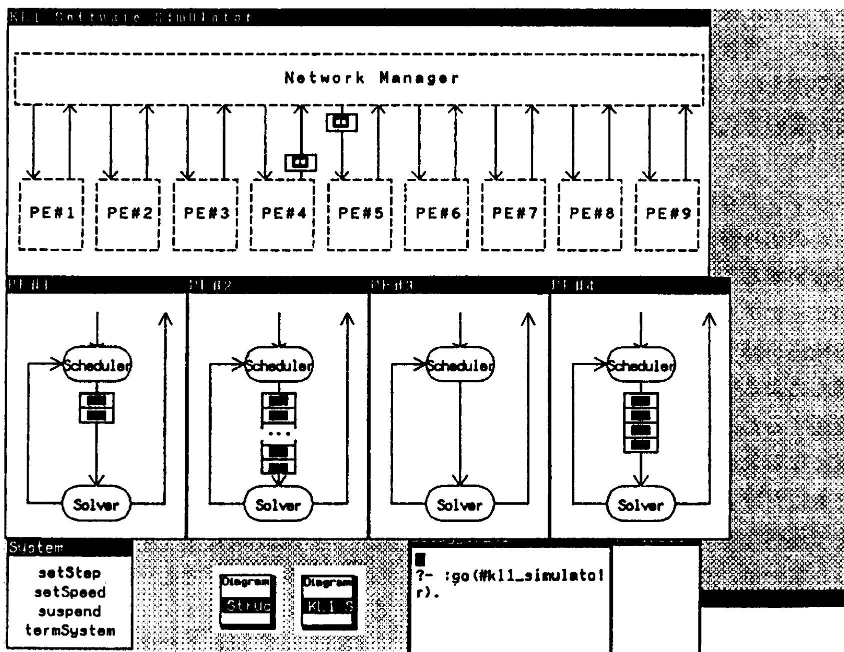


図 10 （b）

すさのために，設計図の表示スケールと表示範囲を変更して拡大表示している．

(e) (c)の状態で処理状況の設計図を表示した状態：処理状況の図には，KL1の下にPE♯1～PE♯9とNM（ネットワーク・マネージャ）があり，各PEには，scheduler と solver があるという処理モジュールの階層を示しており，現在実行されているモジュールが反転表示されている．全体構成やプロセッサの内部構成は，データフロー的な観点からの図であるが，処理状況については，コントロールフロー的な観点からの図である．

実験の結果，本論文で述べた方式によると，実用規模のプログラムに対しても，比較的少量の可視化指示を与えることで，プログラムの動作を設計図上の動作

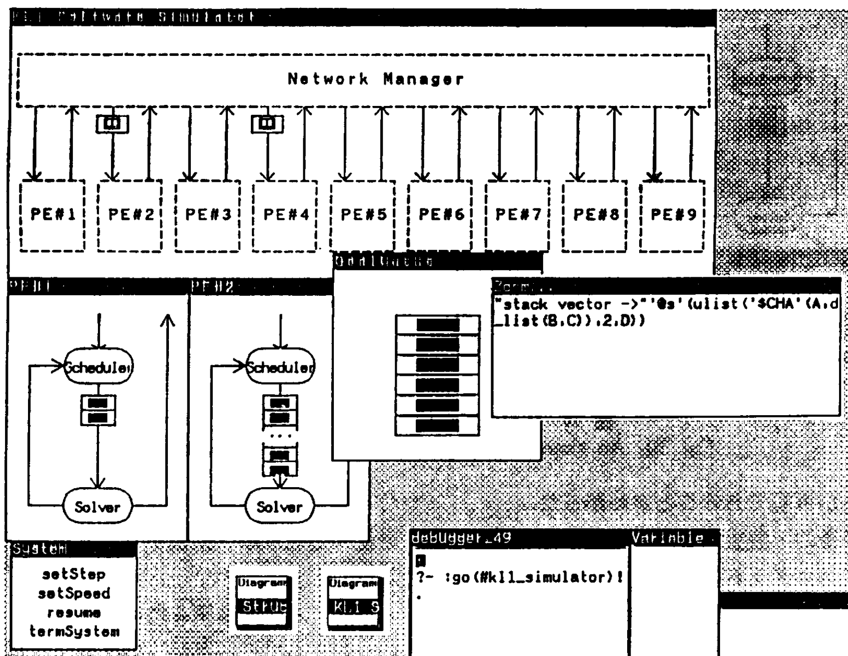


図 10 (c)

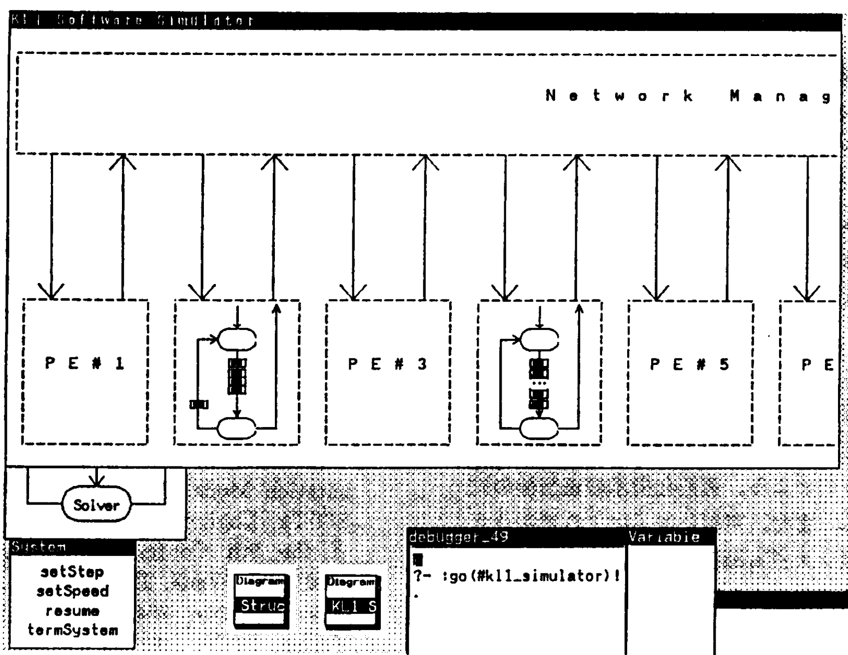
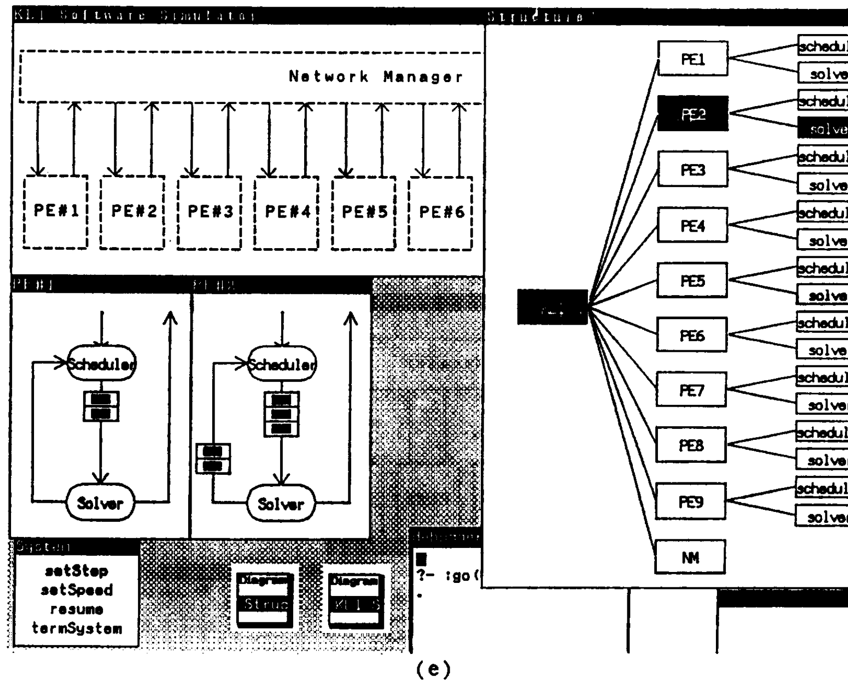


図 10 (d)



(e)

図 10 試作システムでの可視化実行例
Fig. 10 An example of visualization on the prototype.

に対応をとることが可能であることがわかった．

また，複数の多重的・階層的なビューに対応する設計図上で，同時にプログラムの実行動作を可視化することが，プログラムの動作理解に大きく役立つことがわかった．

5. おわりに

プログラム可視化システムについて述べた．プログラム可視化の基本アーキテクチャについては，トレーサ埋め込み方式を採用した．

また，ESP プログラムの動作および構造を可視化する，プログラム可視化システムの試作を行った．プログラム可視化および多重化・階層化ビューのサポートが，プログラムの動作理解に有用であることが判明した．

可視化指示および可視化システムの正当性が保証されていれば，異常な表示があった場合に，それは対象プログラムの異常となり，故障の発見にも使用できる．ただし，現在のところ，可視化指示の正当性の確認は手作業である．また，可視化指示により可視化される情報は，対象システムの動作のすべてではないが，動作理解の目的には十分なものであると考えられる．

プログラム設計システムなどとの連携により，正当な可視化指示を自動生成すること，および，対象言語を並列処理言語へ拡張することが，将来の課題である．

謝辞 本研究は第5世代コンピュータプロジェクトの一環として行われた．御支援いただきました長谷川 ICOT 第1研究室室長を初めとする方々に，深く感謝いたします．

参考文献

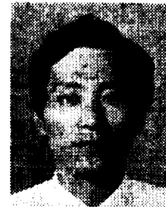
1) Grafton, G. B. and Ichikawa, T.: Visual Programming, *IEEE Comput., Special Issue for Visual Programming*, pp. 6-9 (Aug. 1985).
2) Moriconi, M. and Hare, D. F.: Visualizing Program Design through PegaSys, *IEEE Comput., Special Issue for Visual Programming*, pp. 72-83 (Aug. 1985).
3) Isoda, S. et al.: VIPS: A Visual Debugger, *IEEE Softw.*, Vol. 4, No. 3, pp. 8-19 (May 1987).
4) Morishita, S. and Numao, M.: Prolog Computation Model BPM and Its Debugger PROEDIT2, Logic Programming '86 (Proceedings of the 5th Conference Tokyo, Japan, June 1986), ed. Wada, E., *Lecture Notes in Computer Science 264*, pp. 147-158, Springer-Verlag, Berlin (1987).
5) Brown, M. H.: Exploring Algorithms Using

Balsa-2, *IEEE Comput.*, Vol. 21, No. 5, pp. 14-36 (May 1988).
6) Chikayama, T.: Unique Features of ESP, *Proc. of FGCS '84*, pp. 292-298 (1984).
7) Ichikawa, I. et al.: Program DEsign Visualization System for Object-Oriented Programs, *SIGPLAN NOTICES*, Vol. 24, No. 4 (*Proceedings of the ACM SIGPLAN Workshop on Object-Based Concurrent Programming*, San Diego, Sept., pp. 20-27, 1988), ed. Agha, G., Wegner, P. and Yonezawa, A., pp. 181-183 (Apr. 1989).
8) Ohara, S. et al.: A Prototype Software Simulator for FGHC, Logic Programming '86 (Proceedings of the 5th Conference Tokyo, Japan, June 1986), ed. Wada, E., *Lecture Notes in Computer Science 264*, pp. 46-57, Springer-Verlag, Berlin (1987).

(平成元年8月31日受付)
(平成2年9月11日採録)

市川　至（正会員）
昭和33年生．昭和62年東京工業大学博士後期課程修了(情報工学)．工学博士．昭和61年富士通(株)に入社．JUNET など UNIX ネットワークの発展に微力ながらも協力する．現在，(株)富士通研究所にて，ソフトウェア工学，特にソフト開発における視覚化の研究に従事．電子情報通信学会会員．

小野　越夫（正会員）
昭和26年生．昭和50年横浜国立大学工学部電気工学科卒業．同年(株)富士通研究所に入社．ソフトウェア工学系の研究開発に従事．



毛利　友治（正会員）
昭和25年生．昭和48年九州大学工学部電子工学科卒業．昭和50年同大学院工学研究科(電子工学専攻)修士課程修了．同年(株)富士通研究所入社．以来，ソフトウェア工学，人工知能の研究に従事．現在同研究所企画調査室主任研究員．ソフトウェア開発支援，ロジック・プログラミング，オブジェクト指向，視覚的プログラミングに興味をもつ．IEEE 会員．