3次元ソフトウェア視覚化の枠組と実例による有効性の評価†

小池英樹†† 石井威望†††

3次元コンピュータ・グラフィクスのソフトウェア視覚化への応用がいくつか提案されているが，いずれも3次元利用の必然性に関する説得力に欠ける．そこで本論文は，3次元視覚化表示をソフトウェア開発ツールのユーザ・インタフェースとして利用する具体的枠組を提案すると共に，視覚化システムを試作し，実際のソフトウェア視覚化への適用を通じて，その有効性を示した．まず，2次元図形ツールの問題点を考察し，本論文が提案する3次元視覚化について述べた．次に，試作した3次元視覚化システム VOGUE について述べた．さらに，VOGUE を2種類の異なるソフトウェア視覚化に適用した．第1の適用例はプロセス・モニタである．3次元の概念を導入することで，プロセス間関連とタイミング・チャートを同時に視覚的に支援できた．これを用いて電力制御用ソフトウェアにおける障害事例の視覚化と，自律分散型マニピュレータの協調動作の視覚化を行い，3次元の有効性を示した．第2の適用例はオブジェクト指向言語のクラス・ライブラリである．3次元の概念を導入することでクラス階層とメソッド継承を同時に視覚的に支援することが可能となった．被験者実験によりメソッド探索における2次元ツールに対する優位性を示した．

1. はじめに

本論文は，3次元コンピュータ・グラフィクス(CG)による視覚化表示を，ソフトウェア開発ツールのユーザ・インタフェースとして利用する枠組を提案すると共に，視覚化システムを試作し，実際のソフトウェア視覚化への適用を通じて，その有効性を示す．

従来から様々な図がソフトウェア開発に利用されてきたが，ビットマップ・ディスプレイ等の開発に伴い，計算機ディスプレイ上にソフトウェア情報を視覚化表示する研究が，ソフトウェア工学の1分野として重要視され始めている[1),2)]．しかし，紙上に記述していた図のディスプレイ上への単純な移植は，計算能力の有効利用とは言えず，実用性にも乏しい．一方，機械系 CAD，分子グラフィクス等における3次元 CG 導入成功の理由は，高度なグラフィクス能力を利用した動きの表現，半透明表示等の仮想表示，空間的物体の空間的表現等が挙げられる[3),4)]．この3次元 CG をソフトウェア視覚化へ応用する試みはいくつか存在するが[5)~7)]，いずれの場合にも3次元導入の根拠が薄弱である．したがって本論文の目的は，ソフトウェア視覚化における3次元視覚化の意義を，具体例を示して明らかにすることである．

以下では，まず2次元視覚化ツールの問題点を述べ，著者が提案する3次元視覚化について述べる．次に，試作した視覚化システム VOGUE について述べ，2つの適用例を示し，3次元視覚化の有効性を実証する．第1の適用例はプロセス・モニタであり，第2の適用例はオブジェクト指向言語のクラス・ライブラリである．両者とも今後のソフトウェア工学における重要なテーマの1つであり，例題として適当なものであると考える．

2. 3次元視覚化

2.1 ソフトウェア開発に利用される図の分類

ソフトウェア開発においては，その局面に応じていくつかの異なる視点からソフトウェアを解析する必要性が生じ，視点に応じた図が用いられている．出原[8)]によれば，一般に図形言語は連結系，領域系，座標系，配列系に分類できるが，ソフトウェア開発に利用される各図形を，この分類にあてはめると**表1**のようになる．これらの分類を基にした考察から以下のことが言える．

- 連結系としてのグラフ表現が一般に用いられる
- 領域系は描画の困難さから使用が限定され，連結系で代用されることが多い
- 座標系としては1軸を時間にとったものが多用される
- 配列系は特殊な場合に限られる

以上から明らかなように，ソフトウェア開発において利用される図は，各ソフトウェア・オブジェクトを

† A Framework for Three-Dimensional Software Visualization and Evaluation of Its Effectiveness by HIDEKI KOIKE (Department of Communication and Systems, The University of Electro-Communications) and TAKEMOCHI ISHII (Faculty of Environmental Information, Keio University).

†† 電気通信大学電子情報学科
††† 慶応義塾大学環境情報学部

表 1　ソフトウェア開発に利用される図の分類
Table 1　Classification of types of visual tools used in Software development.

連結系	状態遷移図，ペトリネット，データフローグラフ，フローチャート，階層図，PART 図
領域系	
座標系	タイミング・チャート，工程管理図
配列系	スプレッド・シート

ノード，オブジェクト間の関係をリンクとして表すグラフ表現と，ソフトウェア・オブジェクトの時間的変化を表す時間関係図がその多くを占める．現在の視覚化ツールは，基本的にはこうした図を2次元平面上に実現したものと言える．本論文での議論は，これら2種類の図を対象として行われるが，上での分類から，著者らの提案する枠組は，ソフトウェア視覚化の広い範囲を包含できると考える．

2.2　2次元視覚化の問題点

一般に，人間の認知能力の特徴の1つとして，その並列性が挙げられる．人間は，多くの情報が記述された図から，必要な情報を迅速に獲得することができる．この並列性の有効利用を考えた場合，1つの図には多くの情報が記述されることが望ましい．しかし，情報量が認知許容限界を越えてしまうと，図は逆に人間の認知を妨げる．特に，ソフトウェアに関する情報は多すぎるため，1つの図にすべての情報を記述すると図の複雑さが増し，本来ユーザを支援すべき図が逆にユーザの理解を妨げることになる．したがって，ユーザの各視点は各々別な図として表され，特にマルチ・ウィンドウ環境内では，視点に対応する図が各々別なウィンドウ上に視覚化される．しかし，ユーザの視点が異なれば，同一の対象が異なる形で表現されることがしばしば起こる．図はユーザのメンタル・モデル形成を支援するのみならず，それを強制する効果をも持つため，ユーザは各図から各々のメンタル・モデルを獲得する．結果として，ソフトウェア全体を理解するためには，各図から得たメンタル・モデルの整合性を保った，1つのメンタル・モデルを再構成する負荷を強いられる（図 1）．

以上の議論から，ソフトウェア視覚化に対し，互いに矛盾する2つの要求仕様が存在することがわかる．つまり，

- 1つの図で可能な限り多くの関係を見ること
- 他の関係を消すことなく，各関係に焦点を移すこと

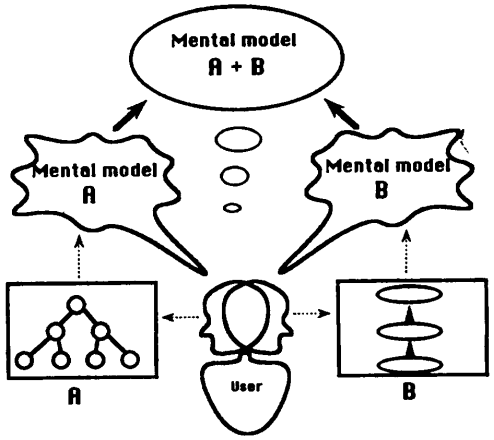


図 1　メンタル・モデルの再構成
Fig. 1　Reconstruction of a mental model.

例えば，前述のグラフ表現の例として，is-a 関係と has-a 関係の2つを持つ知識ベースの表示を考えてみよう．本来，この知識ベースの構造を理解するためには，2関係の分離記述は望ましくない．しかし，知識ベースが大規模になると，その図の複雑さは著しく増大する．同様のことは，オブジェクト指向言語のクラス・メタクラス関係の表示でも起こる．ここにはクラス・インスタンス関係と上位・下位クラス関係が存在し，実際には両関係とも各々階層構造をなしているが，2次元では両者を同時に明確に記述するのは困難である．さらに，SemNet[5] で指摘された，リンクの交差によるグラフの視認性の低下も考慮されねばならない．

次に，時間関係図の場合，縦軸にはソフトウェア・オブジェクト（例えば，後述するプロセス・モニタの場合は各プロセス，プロジェクト行程図の場合は各サブ・プロジェクト等）を配置し，横軸に時間軸をとる．しかし，このオブジェクト同士が階層構造等，何らかの関係を持つ場合，縦軸の1次元でこれを表現することは困難である．したがって，こうした関係は別な図として提示されることにならざるをえない．さらに，オブジェクト数が増加した場合，現在のように1軸に記述するのでは限界があるのは明らかである．

2.3　3次元視覚化による解決

本論文で提案する3次元視覚化は，前述の2つの要求仕様を同時に満たし，かつ他の問題点をも解決することができる☆．

☆ ただし，ここで言う3次元とは，xyz 座標によって定義される3次元空間の意味で用い，要素記号の形態的差異による自由度のことではない．色・形等の次元要素は差異により分類された部分集合間の順序関係を示す能力が弱い．これらの次元要素は上述した3次元表現を補助する目的に使用されるべきであると考える．

一般に，1つの2次元表現は1視点しか提供できないが，3次元表現は複数を提供することができる．各オブジェクトが3次元空間内に適当に配置されると，ユーザは複数の関係を同時に把握しつつ，視点を変更することで各関係に焦点を移動できる．この場合，ユーザに対して提示される図は，あくまでも1つであるから，メンタル・モデル再構成の負荷は生じない☆．

グラフ表現の場合，各関係の分離記述が可能となるとともに，立体的グラフ表現によって，2次元グラフにおけるリンク交差の問題も解決する．時間関係図においては，次元の増加によりオブジェクト間関係の記述が可能となるとともに，描画領域が増加し，多数オブジェクトにも対応できる．実際の視覚化例については，各々4章および5章で示されるが，ここで提案した3次元視覚化は紙上では実現不可能であり，3次元CG を利用する意義がそこにある．

3. 試作システム VOGUE

3.1 システム設計の基本方針

提案した3次元視覚化の有効性を実証するために，試作システム VOGUE (Visualization Oriented Generic User-interface Environment) を実現した．試作にあたっては以下のような設計指針を念頭においている．

● 視覚化に主眼をおく

将来的には VOGUE で示された枠組を用いて，3次元 CG と直接対話する環境の実現が望まれるが，現段階では3次元入力インタフェース技術が確立されていない．ただし，ここで言う3次元入力インタフェースとは以下の事項をさす．

— 3次元対象のポインティング

— 3次元空間での視点の移動

よって，図の直接操作によるデータ変更は行わず，視覚化 (Visualization) に主眼をおく．

● 汎用性の重視

3次元視覚化の概念の有効性実証には，特定のソフトウェアにだけ適用できる視覚化ツールの開発では不十分であり，異種ソフトウェアに対してもツールの基本部分を変更せずに対応できることが望ましい．したがって実現にあたっては汎用性を重視し，実現手法としてはノード・リンクを構成要素とするグラフ表現を採用する．また，グラフ表現は容易に時間関係図へも適用できる．

● 3次元立体視のハードウェア・サポート

3次元 CG による視覚表現は，そのままではただの2次元投影図にすぎず，リンク交差の問題は解決されない．後述する視点の高速な移動により，立体感を得ることもできるが，図形を静止させて見る場合には，この立体感の問題のソフトウェアだけによる解決は困難であると考えられる．VOGUE ではこうした点を補助する目的で，立体視眼鏡を用いたハードウェアによる3次元立体視の支援をも行う．

● 視点の高速かつ連続的切り替え

ユーザの着目点移動に迅速に対応するために，高速な視点切り替えを支援する必要がある．また Shepard らの実験[10]によれば，異なる角度から見た3次元物体の認識には，角度に比例した時間が必要となる．したがって，連続的視点移動が重要である．

3.2 システム構成

設計指針に基づき試作した VOGUE のハードウェア構成図を図 2，ソフトウェア構成図を図 3 に各々示す．SUN Microsystems 社製 SPARC Station 1 と Hewlett Packard 社製 HP 9000/350 SRX がイーサネットを介して接続され，HP 上のグラフィクス画面は液晶シャッタ眼鏡を用いて3次元立体視を行うことが可能である．SUN 側には CLOS (Common Lisp Object System)[11] を拡張して実現したオブジェクト

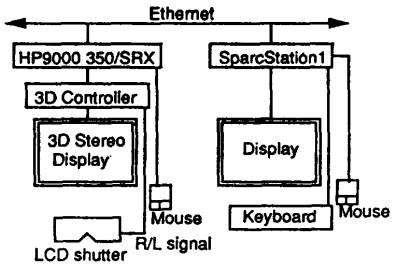


図 2 VOGUE のハードウェア構成
Fig. 2 Hardware of VOGUE.

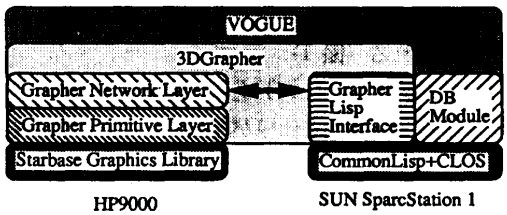


図 3 VOGUE のソフトウェア構成
Fig. 3 Software of VOGUE.

☆ この点に関し，宮崎[9] は Apollonius の円錐曲線を例に，複数の図を関連づける1つのモデルが，人間の理解に及ぼす影響の重要性を指摘している．

指向データベース(以後DBと略記)と3次元Grapherの Lisp インタフェース部があり，HP 側には3次元 Grapher 本体が実現されている．Grapher 本体は Starbase Graphics Library を用いて記述され，グラフィクス・ライブラリに依存する下層と，Lisp からのメッセージを解釈・実行する上層とからなる．このような2層構造の採用によって，他の3次元グラフィクス・ライブラリへの移植性が高まっている．

VOGUE による視覚化は，まず視覚化対象を DB 上でモデル化し，次にこれらインスタンスへのポインタを持つ概念的ノード・リンクを生成する．そして，この概念的ノード・リンクへ描画命令を送ると，それらは HP へ送られ3次元グラフが生成される，以後，グラフィクス属性の変更等は，各概念的ノード・リンクへのメッセージで実現される．

ユーザ・インタフェースとしては，ディスプレイ上に数個のボタンとスクロールバーを用意し，ユーザは並進，回転，ズームイン・アウトの視点移動，およびノードの選択といった基本操作をマウスで実行できる．

4. 適用例 1: プロセス・モニタ

本章では，まず時間関係図の3次元視覚化例として，プロセス・モニタをとりあげる．

4.1 3次元プロセス・モニタの概念

計算機システムの大規模分散並列化により，今後並列（あるいは並行）プログラミングの必要性は増すばかりである．McDowell らは，実際に有効だと思われる並列プログラムのデバッグ手法の1つとして，制御の流れや分散されたデータの視覚化を挙げ，複雑な並列システムに理想的なデバッガは，ある瞬間のシステムの状態と一定時間に渡った動作のパターンの両方を，マルチ・ウィンドウ等で支援するものであると述べている[12]．しかし，前述のとおりマルチ・ウィンドウの乱用は，ユーザに対する負荷を高める危険性を

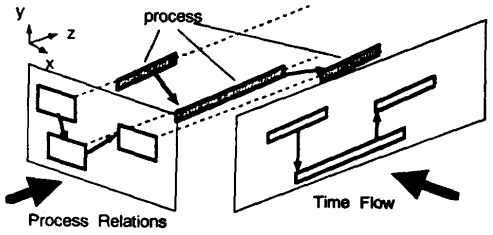


図 4　3次元プロセスモニタの概念
Fig. 4　3D representation of process monitor.

持つ．

この両者は3次元の概念を採用することにより，図4 のように同時に視覚化することが可能である．xy 平面にプロセスを配置し，z 軸方向に時間軸をとることにより，xy 平面に垂直な方向から見るとプロセス間の関連を表し，これに垂直な方向から見ると従来のタイミング・チャートを表すことができる．

以下ではこの3次元プロセス・モニタの有効性を示すために，2種類のシステムの視覚化を行う．まず，電力制御用計算機の実機を稼働して得られたトレースデータの視覚化を行い，プロセス・モニタ3次元化の意義を議論する．ただし，現段階で複数台計算機の協調動作のトレースデータを取得することは困難であるため，本例は1台の計算機のトレースデータである．次に，自律分散型マニピュレータのトレースデータの視覚化を行い，将来的な分散並列システムへの適用を議論する．

4.2 電力制御用ソフトウェアの視覚化

4.2.1 電力制御用ソフトウェア

現在，電力制御システムのデバッギング・ツールとして，専用のプロセス・モニタが開発・実用化され，実機を稼働することで図5のようなトレースデータを取得することができる．このトレースデータからは多くの情報を入手することが可能であるが，ここではプロセスの状態と時間に着目し，視覚化を行った．具体的には，各タスクに対応するノードを z 軸方向に細

NO.	INF. TYPE	TIME	PID	PROCESS NAME	STATE	PRI	M	SYSTEM EVENT	PC	PFC
1	WAIT	22:18:18.54	200A9	_TTA7:	HIB	25	U		7FFEDF8A	584
2	CTXST	22:18:18.54	10080	MMSKLT	CUR	24	U		D28D	
3	EVENT	22:18:18.58	100AA	DBBOBA	LEF	1 30		AST	7FFEDE09	
4	CTXED	22:18:18.58	10080	MMSKLT	CUR	24	K		17F80	0
5	CTXST	22:18:18.58	100AA	DBBOBA	CUR	30	E		7FFEDE09	
6	EVENT	22:18:18.58	100AA	DBBOBA	CUR	30		EVENT	7FFEDE09	
7	EVENT	22:18:18.58	200AB	ERRFMT	LEF	27 27		AST	7FFEE128	
8	WAIT	22:18:18.58	100AA	DBBOBA	HIB	30	K		7FFEDF8A	0
9	CTXST	22:18:18.58	200AB	ERRFMT	CUR	27	U		7FFEE128	
10	EVENT	22:18:18.58	1006E	FILAIP	HIB	24		EVENT	7FFEDF8A	

図 5　電力制御用ソフトウェアのトレースデータ
Fig. 5　Trace file of electric-power control software.

表 2 プロセス状態と対応する色
Table 2 Process states and colors assigned to each process.

状 態	色
PFW	マゼンタ
LEF	緑
HIB	黄色
COM	シアン
CUR	赤
その他	青

長い柱状にし，xy 平面上にサブシステムと呼ばれる関連するタスク集合ごとに配置する．次にトレースデータを1行ずつ読み込み，対応するタスク・ノードの x，y 座標，および時間に対応する z 座標に，プロセスに対応するノードを配置する．プロセスは全部で14種類の状態をとるので，各状態にそれぞれ表 2 に示した色をあてはめた．またプロセス・ノードだけを表示するとプロセス間の時間関係の把握が困難になるため，直前まで CUR 状態だったノードから現在 CUR 状態のノードに対してリンクをはることとした．

4.2.2 障害事例の視覚化

実際の障害時のトレースデータを得るのは現実には困難であるので，制御用ソフトウェアにパッチをあてることで障害を疑似的に再現した．

まず，普通各プロセスが資源を確保する順番は決められているが，この順番を通常とは逆にすることでデッドロック状態を仮作成した．図 6 は正常時，および図 7 は障害時のトレースデータの3次元視覚化である．視認性を考慮して，着目しているタスク以外のタスクに対応する柱は消去してある．前半部分はほとんど同じ形状であるが，中間部分から差がでているのがわかる．障害時にはプロセスの状態が緑，つまり LEF 状態で止ったまま沈黙に入っている．一方，正常時にはその後何度かプロセスが走り，最終的に黄色，つまり HIB 状態で終了している．プロセスが資源待ちになると，状態は LEF になって止るので，このプロセスは資源待ちのままデッドロックに入ったことが視覚的にわかる．

次に，普段は線形リスト構造をした電力監視データを，意図的にループ状にすることで，無限ループを仮作成した．図 8 は正常時と無限ループ時のトレースデータを同時に3次元視覚化し，これを時間軸に垂直な方向から見たものである．正常時（図8上部）には，制御がシステムの状態変化を解析するプロセス（状態解析プロセス）から電流電圧監視プロセスに移り，監視が終了すると再び制御が状態解析プロセスに移っていくが，障害時（図8下部）にはこの電流電圧監視プロセスがいつまでたっても終了しないのが視覚的に明らかである．

4.2.3 電力制御ソフトウェアの視覚化に関する考察

現在のデバッグ手法は，開発者のノウハウによるところが大きく，トレースデータから可能性の高い部分を抽出し，詳細に調べるという作業が行われている．障害発生時のトレースデータを取得した段階では，障害の種別および発生箇所は明らかではないため，開発者は微妙な兆候の集合から障害を同定していく．この意味で現在のデバッグ作業はボトムアップ的である．一方，視覚的に表現した場合には，まずマクロな見地からトレースを見て，障害発生箇所および種類の同定を正常時の視覚表現と比較することで行える．つまり，ドップダウン的な解析が可能となる．例えば，デッドロックの視覚化の場合，開発者は正常状態のプロセス実行状態を視覚的パターンとして記憶しており，異常が発生した場合，まず図から障害を検知する．次に，全体のどのプロセスの挙動に異常があるかを同定した後，そのプロセスのソースのデバッグに入ることができる．勿論，図だけから障害を正確に特定することは困難であり，これは従来のデバッグ手法によって解決されるべきである．視覚化の意義は，障害の検知，障害箇所の推定をいち早く行える点である．

次に3次元化の意義を考察する．図8のような側面図は，現在の2次元平面図でのプロセス・モニタに対応しているが，グラフの全体像を把握するためには，視点の位置を図8のように十分後方にとらなくてはならない．この時，ノードの色を認識することは不可能である．またプロセスの詳細を調べるために接近すると，今度はグラフ全体としての形が把握できなくなってしまう．これに対して，図6のような角度からグラフを見ると，視点位置の近傍のノードは色まではっきりと認識でき，かつグラフ後方の状態をも同時に把握することが可能である．もちろん前者の場合，横方向の縮尺を変更して，より狭い範囲内に全グラフを表示することは可能であるが，この場合には1ノードの奥行きが極端に短くなり，微妙な色の変化を認識するのに障害が起こると考えられる．

第2に2次元表現の場合には，サブシステム間の関連が把握しにくい点が挙げられる．図8，つまり従来のプロセス・モニタに対応する視点では，サブシステム間の関係は不明瞭である．一方，図6の角度から見るとサブシステム間の関係が理解できる．本来，空間的広がりのあるサブシステム間の関係を，2次元プロセス・モニタでは1軸上で表現しなければならない．しかし，3次元視覚化を行うことで，これらを空間的パターンとして自然に表現することが可能になるのである．

4.3 自律分散型マニピュレータにおけるメッセージ送信の視覚化

4.3.1 自律分散型マニピュレータ

池井の自律分散型マニピュレータ[13]は，複数の回転型関節を直列に接続した多関節型ロボットアームであり，各関節ごとに各々計算機を組み込んで搭載している点が特徴である．各計算機は基本的に並列に動作し，メッセージ送信という形態で他の計算機と通信を行うという協調動作のもとで，与えられた作業を実行する．

今，上位制御計算機から肘上げの指令メッセージが

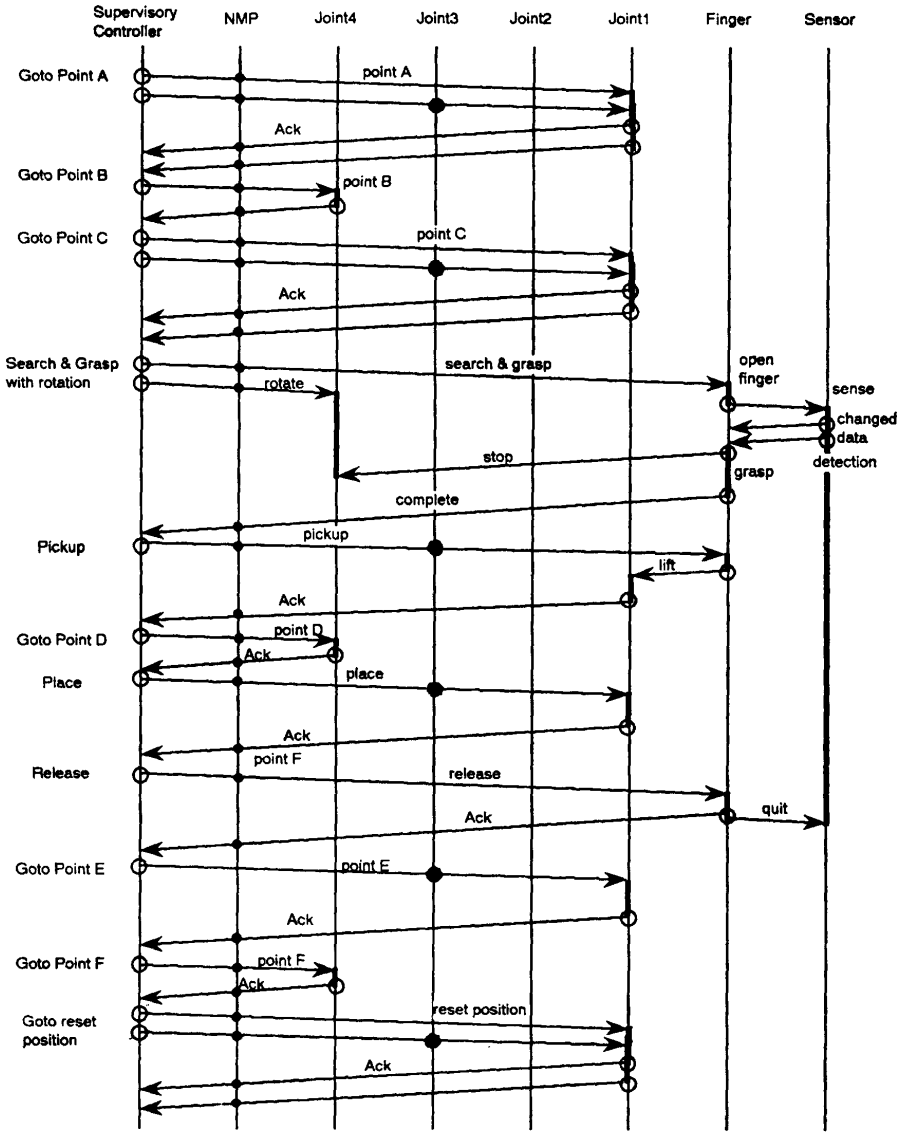


図 9　第3関節モータ故障時のメッセージ伝達
Fig. 9　Time chart while the third joint is damaged.

与えられたが，第3関節モータが故障している場合には，図9のように第3関節において，メッセージが第1関節に再送され，残りの関節を用いて不完全ながら代替動作が行われる．矢印に付加された中空黒丸はメッセージの送信計算機，黒丸は経由した計算機，第3関節部分の灰色丸はメッセージの再送を表している．2次元表現の場合には，メッセージを示す矢印が計算機を表す線を交差せざるを得ず，経由した計算機が把握しづらい．これに対して，xy 平面上に計算機に対応するノードを配置し，z 軸方向に時間軸をとることで3次元視覚化を実現した．メッセージの集中を考慮し，ネットワーク管理用計算機（NMP）を中心に，上位制御計算機をその左に配置し，第4～第1関節，指部，センサ部計算機をこの順番に NMP の上から時計回りに配置した．図 10，図 11 は VOGUE によるトレースの視覚化である．経由した計算機，全く使用されていない計算機等が明白である．

4.3.2 自律分散型マニピュレータの視覚化に関する考察

ここで取り上げた自律分散型マニピュレータは，小規模ながらも分散並列処理システムとしての本質を有する．並列計算機システムのデバッグにおける問題点の1つは，いかに迅速に障害を起こしている計算機を同定するかであるが，従来の2次元上でのメッセージ記述は，視認性が低くこうした要求を満たしてはいない．一方，3次元視覚化の場合，図を見ることにより，本来通信の行われる必要のない第3，第1関節間においてメッセージのやりとりがあることから，第3関節で行われるべき処理が，何らかの理由から不可能となり，第1関節へのメッセージ送信となっているとの推測が成り立つ．こうして開発者は，第3関節のハードウェア的あるいはソフトウェア的障害を同定することが可能となるのである．

こうしたメッセージの視認性の高さは，3次元化によるリンク交差の回避に基づく視認性の差ではなく，計算機の平面的広がり情報と時間情報の2つが同時に認識可能だという事実によるものであると考えられる．ここで図9のような2次元の図においても，例えば第1関節部と第3関節部に相当する線を入れ替えることで，メッセージ送信を示す矢印を直線でなくし，第3関節経由の事実をわかりやすく表示することは可能であるとの反論があるかもしれないが，それは本例における特殊な場合においてのみ有効な方法であって，一般のメッセージ送信には対応しきれない．一方，3次元図の場合には，勿論初期の配置にもよるが，一般的に言ってこれらのメッセージ送信順番を誤認する可能性は低い．

5. 適用例 2: オブジェクト指向言語のクラス・ライブラリ

オブジェクト指向言語には，クラス，メタクラス，メソッド，インスタンス等のオブジェクトと，各オブジェクト間の関係が存在する．本章では，オブジェクト間の複数の関係の同時記述の例として，クラス間の階層関係とクラス-メソッド間という2つの関係を持つクラス・ライブラリの視覚化をとりあげる．

5.1 オブジェクト指向言語のメソッド継承

オブジェクト指向言語におけるメソッド継承は，開発時にはコード再利用という点から非常に便利な機能であり，ソフトウェア危機に対する手段としてオブジェクト指向言語が注目を集めている主たる要因の1つである．一方で，あるインスタンスへ送られたメッセージは，必ずしもそのインスタンスが属するクラスに存在するとは限らない．この場合，クラス階層を下から順番にたどっていき，最初に現れたメソッドが起動されることになる．こうしたメソッド継承の複雑なメカニズムは，実行時にはシステムによって自動的に行われるため，意識する必要はあまりないが，デバッグ時にはプログラマが1つ1つ追跡する必要が生じる．

この時，視覚的支援として一般的に利用されるのはクラス階層図である．しかし，クラス階層図だけでは，あるクラスのインスタンスに対してメッセージを送った場合に，実際に起動されるメソッドの位置は明らかでない．したがって，同一名を持つメソッドがどのクラスに存在するか，という情報も必要である．

後述する Flavors のような，いわゆるデーモン・メソッドを持つ場合には問題がさらに複雑になる．両者が与えられた場合，対応をとりながら起動されるメソッドを同定することは可能だが，両者のもつ制約を満足するメンタル・モデルをユーザが形成する負荷は大きく，例えメンタル・モデルが形成されたとしても記憶するのは不可能に近い．

5.2 クラス・ライブラリの3次元視覚化の概念

一方，図12は提案する3次元表現の概念図である，クラス階層図を xy 平面に配置し，各メソッドはこの平面と垂直な方向，つまり z 軸方向に，所属するクラスに対応するノードと同一 x，y 座標を持ち，かつ同

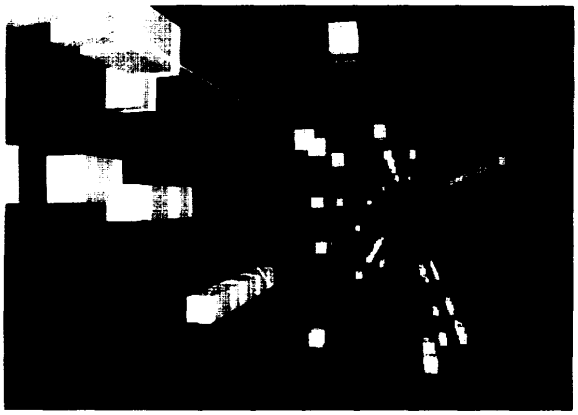


図 6 正常時のトレースの視覚化（正面図）
Fig. 6 Visualization of trace data during normal situation (front view).

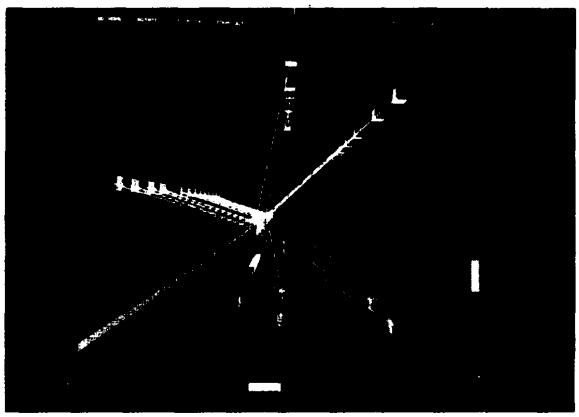


図 10 通常時のメッセージ伝達の3次元視覚化
Fig. 10 3D expression of time chart during normal situations.

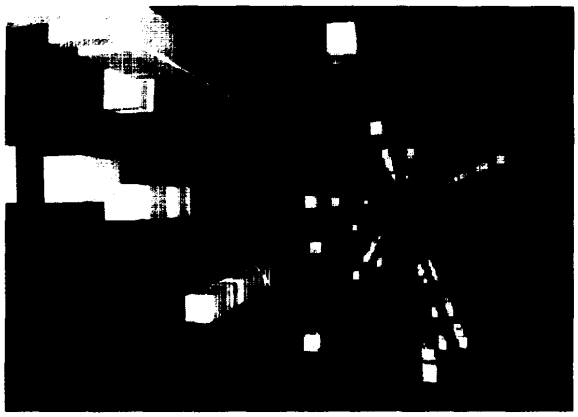


図 7 デッドロック時のトレースの視覚化（正面図）
Fig. 7 Visualization of trace data during deadlock (front view).

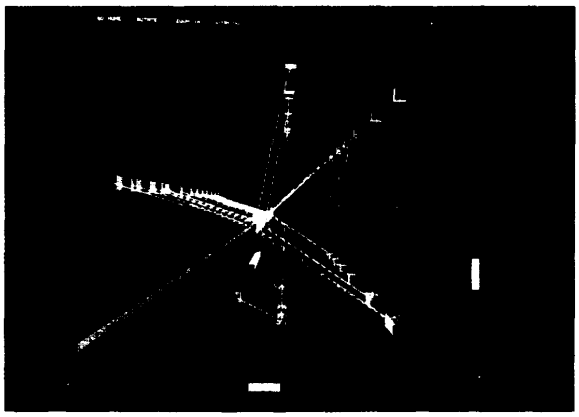


図 11 第3関節モータ故障時のメッセージ伝達の3次元視覚化
Fig. 11 3D expression of time chart while the third joint is damaged.

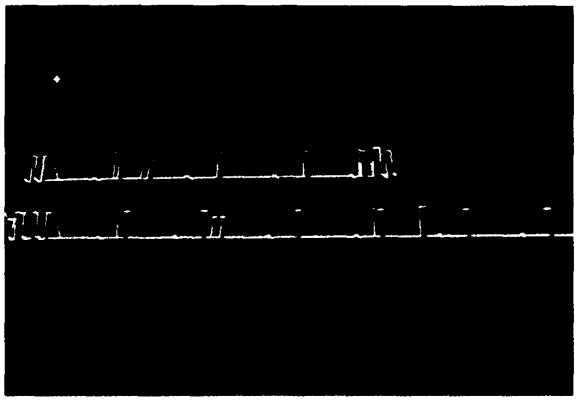


図 8 無限ループの視覚化
Fig. 8 Visualization of trace data during infinite loop (side view).

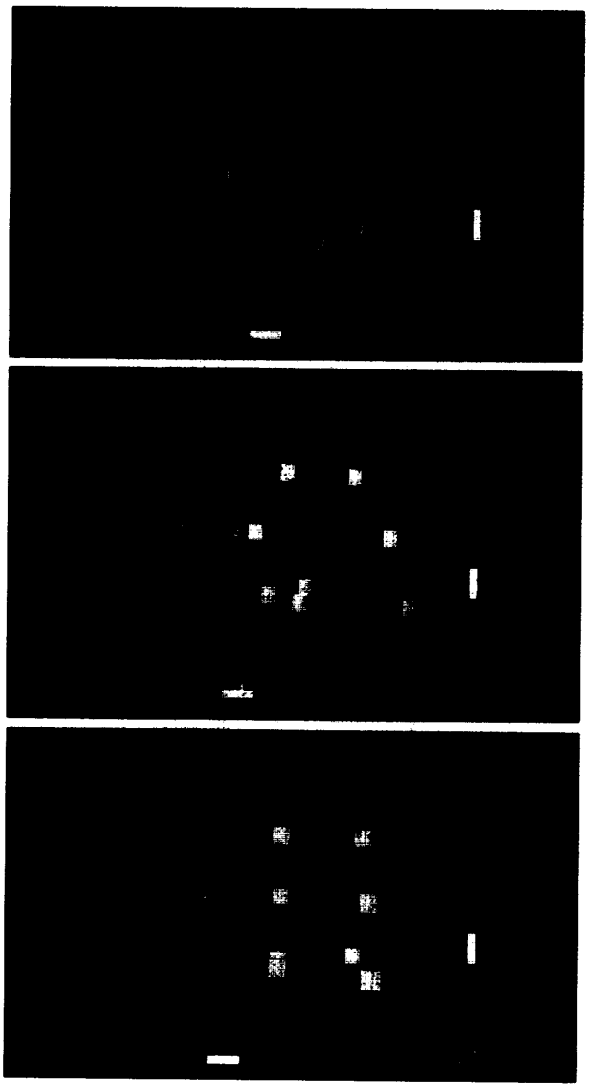


図 13 VOGUE で実現した3次元ブラウザ
Fig. 13 3D browser implemented in VOGUE.

```
(defclass class1 () ())
(defmethod test ((obj class1))
  1)
(defmethod foo ((obj class1))
  (test obj))

(defclass class2 (class1) ())
(defmethod bar ((obj class2))
  (test obj))
(defmethod test ((obj class2))
  2)

(defclass class3 (class1) ())
(defmethod bar ((obj class3))
  (let ((instance (make-instance 'class4)))
    (test instance)))

(defclass class4 (class2) ())
(defmethod test ((obj class4))
  3)

(defclass class5 (class2) ())
(defmethod foo ((obj class5))
  (bar obj))

(defclass class6 (class3) ())
(defmethod test ((obj class6))
  4)
(defmethod bar ((obj class6))
  (foo obj))

(defclass class7 (class3) ())
```

図 14 (Fig. 14)

```
(test (make-instance 'class3))
(test (make-instance 'class5))
(test (make-instance 'class7))
(foo (make-instance 'class1))
(foo (make-instance 'class2))
(foo (make-instance 'class4))
(foo (make-instance 'class5))
(foo (make-instance 'class6))
(bar (make-instance 'class2))
(bar (make-instance 'class3))
(bar (make-instance 'class4))
(bar (make-instance 'class6))
(bar (make-instance 'class7))
```

図 15 (Fig. 15)

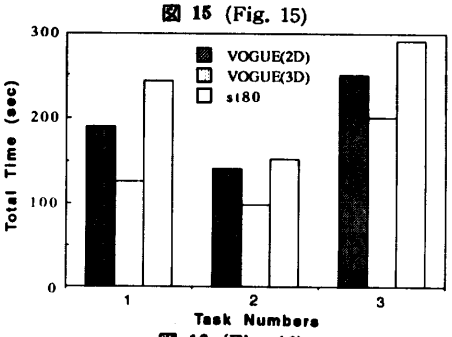


図 16 (Fig. 16)

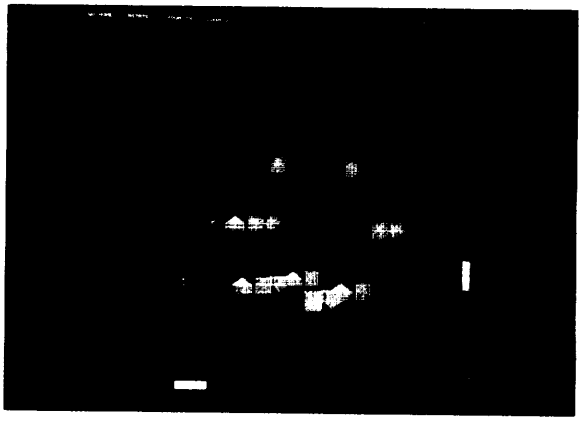


図 17 (Fig. 17)

図 14 視覚化したプログラム
Fig. 14 Sample program to visualize.

図 15 被験者に与えたタスク
Fig. 15 Tasks given to subjects.

図 16 実験結果のグラフ
Fig. 16 Experimental results.

図 17 Flavor に対応したブラウザ
Fig. 17 3D browser for Flavors.

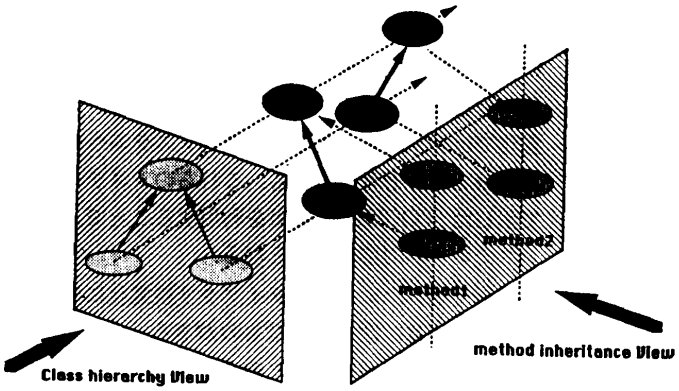


図 12　クラスライブラリの3次元視覚化の概念
Fig. 12　3D representation of class library.

じ名前を持つメソッドは同一の z 座標を持つよう配置する．xy 平面に垂直な方向から見るとクラス階層図を表し，これに垂直な方向から見るとメソッド・リストを表すことができる．

この視覚表現は Smalltalk-80, Flavors, C++ 等，メッセージ送信式の第1引数をメッセージのレシーバとする，一般のオブジェクト指向言語には適用可能であるが，CLOS のようにメッセージのレシーバを特定せず，引数すべてを対等に扱う場合には適用不可能である．ただし，引数が1個以下の場合には，便宜上その引数をレシーバとみなすことによって，Flavors と同様の取り扱いが可能である．以後の実験では便宜上引数を1つ以下に限定することで CLOS のプログラムを取り扱うが，これによってブラウザの本質を損ねることはないと考える．

図 13 が VOGUE で実現した3次元視覚化である．メソッドに対応するノードをマウス・クリックするとノードはハイライトし，対応するソースコードが SUN 側に表示される．

5.3 被験者実験

3次元視覚化表現の有効性を検証するために，2次元図形ツールとの比較実験を行った．ただし，2次元と3次元との本質的な違いに無関係な点での条件を統一するために，Smalltalk-80 上に実現されているクラス階層ブラウザ[14]の本質を損なわない範囲で，VOGUE 上にクラス階層ブラウザをシミュレートしたツールを作成した．クラス階層図が xy 平面上の木として表現され，任意のクラスに対応するノードをマウス・クリックすると，そのクラスに属するメソッドが垂直下に出現する．さらにメソッドに対応するノードをクリックすると，メソッドの定義が SUN 側のウィンドウ上に表示される．別なクラスを選択すると，それまで表示されていたメソッドのノードは消え，新たなメソッドがポップアップする．以後，これを 2D ブラウザと呼び，3次元に実現されたものを 3D ブラウザと呼ぶことにする．

実験は被験者8人を用いた個人実験で以下のように行った．まず，簡単な例題でツールの使用法に関する説明を行い，被験者をツールに習熟させた後，3D ブラウザに**図 14** に示すようなクラス・メソッドの組合せを表示する．その状態で，**図 15** に示すメッセージ送信式を，この順番で1つずつ記述したカードを被験者に与える．被験者は，ブラウザを用いてメソッド定義を参照しながら，メッセージ送信式が最終的に返す値を次々に紙に書いていく．ここでカードを用いたのは前出の答えを後出の問題で再利用しないよう配慮したためである．このような課題を3種類与えた．さらに1日後，今度は 2D ブラウザで同様の実験を行った．

本実験では条件統一のためシミュレート・ツールを用いたが，代表的な Smalltalk-80 のシステム・ブラウザで同様のタスク完遂に要する時間を把握しておくことは意味がある．そこで，あくまでも参考実験として以下のような実験も行った．つまり，Smalltalk-80 のシステム・ブラウザを用い，インスタンス変数，クラス変数等，実験に不必要な部分を極力削除し，コメント形式で図 14 の CLOS のクラス関係，メソッド関係を定義した．メソッド・カテゴリとしては private のみを作成し，すべてのメソッドがこれに属することとした．この場合重要なのはツールとしてのブラウザの機能であって，コードが CLOS か Smalltalk-80 かは本質的でない．各被験者の課題達成までに要した時間から，各課題ごとに最短と最長データを除外した6データの平均をグラフにしたのが**図 16** である．すべての課題に対して 3D ブラウザの方が成績が良く，2D ブラウザに対し 66～80% の時間で課題は達成された．

5.4 Flavors への対応

Flavors 系のオブジェクト指向言語にはデーモン・メソッドと呼ばれる特殊なメソッドが存在する．こうしたデーモンを用いたプログラミングは，効果的に使用すればコードの再利用がさらに促進されるが，逆に

メソッドのトレースはさらに複雑になる．一般のオブジェクト指向言語におけるメソッド探索では，実際に起動されるメソッドを1つ同定しさえすればよかったが，Flavors の場合には関連するデーモンはすべて起動されるので，クラス階層とメソッド・リストの視覚的同時支援がよりいっそう重要である．

図 17 は 5.2 節で実現した 3D ブラウザにおいて，before デーモンを基本メソッドに対して z 軸の負方向，after デーモンを基本メソッドに対して正方向にずらした位置に配置することで実現した 3D ブラウザである．被験者5人を用いて同様の実験を行い解析した結果，2D ブラウザに対し 62% の時間で課題は達成された．

6. 考　　察

6.1 従来の研究との比較

3次元を利用した視覚化ツールとしては SemNet[5] が有名である．SemNet は Prolog 知識ベースの作り出すネットワーク構造を3次元的に視覚化することで，大規模知識ベースの保守を容易にすることを目的とした実験システムであり，3次元利用の主たる目的はグラフの視認性の向上である．SemNet はリンクのセマンティクスを一切無視し，シンタックスだけによるアプローチをとっているが，結果として，複雑な2次元ネットワークを3次元に移植したにすぎず，この意味において3次元 CG の持つ能力を有効に利用したとは言えない．また，Lieberman[7] はプログラムの木構造を3次元的に表し，実行状態を動的に視覚化しているが，3次元 CG 導入の主たる目的はその動画能力にあり，2次元システムとの本質的な差はない．一方，本システムでの3次元導入は，3つの次元の積極的利用が目的である．つまり，セマンティクスを利用し，ノードを3次元空間に適当に配置することで，従来の視覚化では困難だった視点の提供を目的としている．

4章で述べたプロセス・モニタに関連する研究としては，次のようなものがある．BALSA[15] は Macintosh 上に実現されたアルゴリズム・アニメーション・システムであり，種々のソート・アルゴリズム等をバーチャートや木を用いて視覚的に表現する．PIE[16] は Mach OS のプロセス状態をバーチャートで表現するプロセス・モニタである．これらのシステムが採用した視覚化アプローチは，教育あるいは並列プログラムのデバッグのために有効な手法であり，かつ実現が容易な点から，今後多くのシステムが採用する手法であると考えられる．しかし，本文でも述べているように，描画領域の問題，要素間の関係記述の問題，そして要素間の通信記述の問題等から，2次元視覚化では将来的な大規模分散並列システムへの対応が不十分であると考えられる．これらのシステムは4章で示した VOGUE の枠組を利用することで，大規模システムへの対応が可能となる．

5章で述べたクラス・ライブラリ視覚化に関連する研究としては，次のようなものがある．クラス階層ブラウザ[14]は Smalltalk-80 のクラス階層を視覚的に支援するシステムである．Prograph[22] は Macintosh 上の視覚的プログラミング環境で，やはりクラス階層図を支援する．しかし，現実のプログラミングにおけるクラス，メソッドへのアクセスは，システム・ブラウザに代表されるメニュー形式ブラウザのほうが早く，これら図形ブラウザは実際にはあまり利用されない．一方，5章で提案した枠組は，クラス階層図とメソッド・リストを同時に視覚的に支援し，その結果として複雑なメソッド継承を視覚的に支援することに成功している．従来，クラス階層とメソッド・リストの2つの情報から，心理的にトレースするしかなかった複雑なメソッド継承を，視覚的に把握することができる．これは図を複雑化させることなく2関係の同時表示を行えたため実現できた．本視覚表現は，実際のプログラマへの支援のほかに，初心者へのメソッド継承の教育にも有効であろうと考えられる．

6.2 問題点，課題

VOGUE は実験システムであり，単独で利用される完成したツールというより，今後のインタフェースの枠組である．したがって，この枠組を基に実用ツールのインタフェースが設計されるべきであり，実用化のためには次のような問題点，課題が残されている．

- 3次元空間における視点の移動方法を検討する必要がある．2次元グラフィクスの場合には水平・垂直方向へのスクロールと前後方向への拡大・縮小で十分であるが，3次元の場合着目点に対する視点の回転を考慮する必要がある．同時に入力デバイスの検討も必要である．
- 図形ツールの問題点の1つは，図形要素の増加に伴う図の複雑化とシステムの画面更新スピードの劣化である．より洗練された図形数制御機能が必要である．

7. おわりに

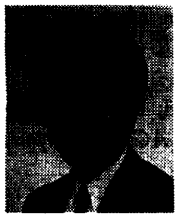
3次元CGを利用したソフトウェア視覚化について述べた．3次元視覚化の意義は2章で述べたが，そのほかに実例の視覚化により以下のことが明らかとなった．プロセス・モニタの例では，3次元透視図法を用いた局所情報と大局情報の把握，プロセス間通信の視認性の向上が示された．クラス・ライブラリの例では，2つの関係の同時提供がユーザの問題解決を早めることが確認された．今後は，本枠組を利用した実用ツールの開発を行うつもりである．

謝辞　マニピュレータに関する資料を提供していただいた大阪大学工学部の池井寧博士，ご支援いただいた東京電力，明電舎，そして，有益なコメントをいただいた査読者の方々に謝意を表す．

参考文献

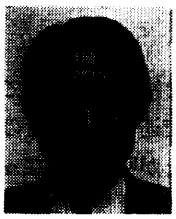
1) Shu, N. C.: *Visual Programming*, Van Nostrand Reinhold, New York (1988).
2) Baecker, R. M. and Marcus, A.: *Human Factors and Typography for More Readable Programs*, ACM Press (1990).
3) 小出昭夫：化学 CAD におけるコンピュータグラフィックス，情報処理，Vol. 29, No 10 (1988).
4) 国井利泰：コンピュータグラフィックス応用の発展動向，情報処理，Vol. 29, No. 10 (1988).
5) Fairchild, K. M., Poltrock, S. E. and Furnas, G. W.: SemNet: Three-dimensional Graphic Representation of Large Knowledge Bases, Guidon, R. ed., *Cognitive Science and Its Application for Human-Computer Interaction*, Lawrence Erlbaum Associates, New Jersey (1988).
6) Glinert, E. P.: Out of Flatland: Towards Three-dimensional Visual Programming, *1987 Fall Joint Computer Conference*, IEEE CS Press (1987).
7) Lieberman, H.: A Three-Dimensional Representation for Program Execution, *IEEE Proc. Workshop on Visual Languages*, pp. 111-116 (1989).
8) 出原栄一：人間と図形言語，吉川広之（編），コンピュータグラフィック論　第3章，pp. 63-117，日科技連（1977).
9) 宮崎清孝：理解と視点―概念理解の場合，佐伯（編），認知心理学講座 第3巻 推論と理解，東京大学出版会（1982).
10) Shepard, R. N. and Metzler, J.: Mental Rotation of Three-dimensional Objects, *Science*, Vol. 171 (1971).
11) 井田，元吉，大久保：Common Lisp オブジェクトシステム―CLOS とその周辺―，共立出版 (1989).
12) McDowell, C. E. and Helmbold, D. P.: Debugging Concurrent Programs, *ACM Comput. Surv.*, Vol. 21, No. 4 (1989).
13) 池井　寧：計算機組み込み型機械システムの構築手法に関する研究，東京大学大学院工学系研究科機械工学博士論文 (1988).
14) 青木　淳，藤田早苗：Smalltalk-80 Goodies Tools Applications，富士ゼロックス情報システムズ (1989).
15) Brown, M. H.: *Algorithm Animation*, MIT Press, Cambridge, MA (1988).
16) Lehr, T., Segall, Z., Vrsalovic, D. F., Caplan, E., Chung, A. L. and Fineman, C. E.: Visualizing Performance Debugging, *IEEE Comput.*, (1989).
17) Card, S. K., Mackinlay, D. and Robertson, G. G.: The Design Space of Input Devices, *CHI '90 Proceedings*, ACM (1990).
18) Myers, B. A.: Visual Programming, Programming by Example and Program Visualization: A Taxonomy, *Proc. CHI '86: Human Factors in Computing Systems*, pp. 59-66, ACM (1988).
19) Stroustrup, B.: *The C++ Programming Language*, Addison-Wesley, Reading, MA (1986).
20) Goldberg, A. and Robson, D.: Smalltalk-80: The Language and Its Implementation, Xerox (1983).
21) Atkinson, M. et al.: The Object-Oriented Database System Manifesto, *Pcoc. of the First International Conference on Deductive and Object-Oriented Databases* (1990).
22) TGS Systems: Prograph Reference Manual, TGS Systems (1990).

(平成3年7月19日受付)
(平成4年4月 9 日採録)



小池　英樹（正会員）

1961 年生．1986 年東京大学工学部舶用機械工学科卒業．1991 年同大学院工学系研究科情報工学専攻博士課程修了．工学博士．同年より電気通信大学電子情報学科助手．計算機ユーザ・インタフェースの研究に従事．特に，ソフトウェア視覚化，ソフトウェア開発環境，フラクタルに興味を持つ．ソフトウェア科学会，人工知能学会，ACM 各会員．



石井　威望（正会員）

1930 年生．1954 年東京大学医学部医学科卒業．1957 年同大学工学部機械工学科卒業．同年より 1 年間通産省．1963 年東京大学大学院博士課程修了．工学博士．同年同大学工学部専任講師，1964 年同助教授，1973 年同教授．1991 年より慶応大学環境情報学部教授．システム工学，医用工学の研究に従事．1972 年 IEEE 論文賞受賞．