06－R003

ビジュアルインタフェースの
研究開発報告書

平 成 7 年 3 月

JIPDEC

財団法人 日本情報処理開発協会

この報告書は、日本自転車振興会から競輪収益の一部である機械工業振興資金の補助を受けて平成６年度に実施した「ビジュアルインタフェースの研究開発」の成果の一部をとりまとめたものであります。

KEIRIN この報告書は、競輪の補助金を受けて作成したものです。

はじめに

コンピュータシステムの利用の拡大とともに、コンピュータを一層使い易くするために必要となるユーザインタフェースの大幅な改善が求められております。

従来のユーザインタフェースは、コマンドなどの論理記号を用いるほか、グラフィカルユーザインタフェース（GUI）に見られるような2次元図形表示を用いた方法が主流でした。最近は、人間がより理解し易いインタフェースとして、人間とコンピュータとの複雑な相互作用を視覚化したビジュアルインタフェースが提案されており、3次元グラフィックス、イメージ、アニメーション、センサー等の高度な技術を用いたインタフェースの研究が進められています。

こうした背景のもとに、当協会では、平成5年度より2年間にわたって、ビジュアルインタフェースに関する最新技術および応用についての研究動向を調査するとともに、3次元視覚化技術を用いた情報アクセスシステムのプロトタイプを開発し、効果、技術的課題の検討などを行いました。

実施に当たりましては、大学、企業などの研究者、専門家からなる「ビジュアルインタフェースに関する調査研究ワーキンググループ（主査　田中二郎　筑波大学電子・情報工学系助教授）」を設け、視覚を中心とした新しいユーザインタフェース、マルチモーダルインタフェース、視覚的ソフトウェア開発などの技術動向、事例、問題点について調査検討を行うほか、平成6年8月には国際会議に海外調査員を派遣し、最新技術情報の収集を図りました。

本報告書は、平成6年度における成果を取りまとめたもので全体は5章で構成されています。第1章は視覚的プログラミング、第2章はマルチモーダルインタフェース、第3章はソフトウェア開発における視覚化、第4章は情報の視覚化とアクセス、第5章は実用化動向となっております。

最後に、本開発研究に当たって、ご指導ご協力を頂いた、田中主査を始め各委員の方々及び関係各位に深甚なる謝意を表する次第です。

平成7年3月

財団法人日本情報処理開発協会

「ビジュアルインタフェースに関する調査研究ワーキンググループ」メンバー名簿（敬称略）

主　査	田中　二郎	筑波大学　電子・情報工学系　助教授
委　員	朝日　宣雄	三菱電機（株）　パーソナル情報機器開発研究所 携帯情報システムプロジェクトグループ　総合企画グループ
委　員	岡田　幹夫	東京電力（株）　システム研究所　制御研究室　主管研究員
委　員	神場　友成	日本電気（株）　C&C研究所　主任
委　員	小島　啓二	（株）日立製作所　システム開発研究所 第5部503研究ユニット　主任研究員
委　員	斉藤　康己	日本電信電話（株）　NTT基礎研究所　主幹研究員
委　員	竹内　彰一	（株）ソニーコンピュータサイエンス研究所　シニアリサーチャー
委　員	垂水　浩幸	日本電気（株）　関西C&C研究所　主任
委　員	西川　博昭	筑波大学　電子・情報工学系　助教授
委　員	萩谷　昌巳	東京大学　理学部　情報科学科　助教授
委　員	服部　桂	朝日新聞社　出版局　編集委員
委　員	平川　正人	広島大学　工学部　第二類（電気系）　情報システム　助教授
委　員	増井　俊之	シャープ（株）　技術本部ソフトウェア研究所　係長
委　員	松岡　聡	東京大学　工学部　計数工学科　講師
委　員	安村　通晃	慶應義塾大学　環境情報学部　教授
委　員	暦本　純一	（株）ソニーコンピュータサイエンス研究所　アソシエートリサーチャー
委　員	向山　博	（財）日本情報処理開発協会　開発研究室　主任研究員
事務局	妙泉　正隆	（財）日本情報処理開発協会　開発研究室　主任部員
ｵﾌﾞｻﾞｰﾊﾞ	Jean-paul Smets-Solanes	（財）日本情報処理開発協会　開発研究室　客員研究員

執筆者一覧

節番号	タイトル	執筆者名	所属
1．1	視覚的プログラミングの歴史	平川　正人	広島大学 工学部 第二類 助教授
1．2	視覚的プログラミングの分類	田中　二郎	筑波大学 電子・情報工学系 助教授
1．3	GUIベースのプログラミング	増井　俊之	シャープ（株） 技術本部 ソフトウェア研究所 係長
1．4	例示によるGUIプログラミング	松岡　聡	東京大学 工学部 計数工学科 講師
2．1	マルチモーダルインタフェースの現状と展望	安村　通晃	慶應義塾大学 環境情報学部 教授
2．2	マルチモーダルインタラクションと社会性	竹内　彰一	（株）ソニーコンピュータサイエンス研究所 シニアリサーチャー
2．3	マルチモーダルにおける音声の役割	竹林　洋一	（株）東芝 研究開発センター 情報・通信システム研究所 第三研究所 主任研究員
2．4	マルチモーダルインタフェースの将来性	斉藤　康己	日本電信電話（株） NTT基礎研究所 主幹研究員
3．1	ソフトウェアの静的視覚化	向山　博	（財）日本情報処理開発協会 開発研究室 主任研究員
3．2	ソフトウェアの動的視覚化	大西　淳	立命館大学 理工学部 教授
3．3	視覚的開発環境	垂水　浩幸	日本電気（株） 関西C&C研究所 主任
4．1	3次元情報視覚化	小池　英樹	電気通信大学 大学院 情報システム学研究科 助教授
		暦本　純一	（株）ソニーコンピュータサイエンス研究所 アソシエートリサーチャー
4．2	3次元を用いた情報アクセス	向山　博	（財）日本情報処理開発協会 開発研究室 主任研究員
		妙泉　正隆	（財）日本情報処理開発協会 開発研究室 主任部員
4．3	テキストビジュアリゼーション	斉藤　康己	日本電信電話（株） NTT基礎研究所 主幹研究員
5．1	PDAに関する動向	朝日　宣雄	三菱電機（株） パーソナル情報機器 開発研究所
5．2	今後の動向	小島　啓二	（株）日立製作所 システム開発研究所 主任研究員

目　次

1．　視覚化プログラミング

1. 視覚的プログラミング

1.1 視覚的プログラミングの歴史

コンピュータは人間の知的活動を支援する道具であり、それを使う人間との係わりの上で評価されるべきである。したがって、コンピュータを中心とするのではなく、ユーザ（人間）を中心とした設計が必要になるということを改めて力説する必要はないであろう。本章で述べる視覚的プログラミングの存在意義は、この流れの上に立って理解することができる。ユーザからの要求の記述が視覚的に行えるようなプログラミングの形態を視覚的プログラミングと呼ぶ[Hirakawa94]。以下では、まず視覚的プログラミングの開発を支持する背景について述べることにする。

人間は処理手順をある種の概念として頭の中で整理し、言語を用いてそれを我々の目に見える形に表現する。ところで、「一般的なもの」や「概念に似たもの」はアメーバのような単細胞動物にも存在することは心理学の成果として確認されている。その一方で、通常我々が認識するような「概念」はアメーバには存在し得ないことも知られている。さらに、人間の概念形成にあたっては、社会で用いられている「言語」が重要な要因として関係しているとも言われている。実際、思考が言語と不可分の関係にあり、言語の形式が論理的思考の導く結論に影響を及ぼすということは、哲学の世界ではよく知られた事実である。プログラミングの世界においても、この法則は成り立っているといえよう。

ところで、従来のプログラミングの世界では、人間自身の頭の中のできごとをコンピュータに伝える手段として許されていたのは文字だけであった。ところが、人間どうしが互いに意志を交換する場面では、文字の他にも、音や声、身振り、絵などの種々の表記法が用いられている。したがって、「人間の側に立った」設計を推し進めるにあたっては、文字以外の表現手段を積極的に導入することが必須である。

誤解がないように一言付け加えておくならば、先に言及した心理学の分野における概念と言語の話題における「言語」とは、記号学者や哲学者が呼ぶところの「記号」として広義に捉えられるべきである [加藤83]。ただし、記号といっても、単に丸とか三角といったような、我々エンジニアが連想する狭い意味での記号を指し示しているわけではない。記号学者のピエール・ギローが、「すべてが記号であり、しかもたっぷりふくらんだ記号なのだ。樹木も雲も顔もコーヒー挽きも .... みな、多様な解釈によって何重にもくるまれている。多様な解釈が意味のパイをこね、いくえにも層を重ねる」と自著の中で述べているように、人間に何らかの意味を喚起するものはすべて記号として捉える。話し言葉や書き言葉を始め、交通信号や電話のベルのような信号、星空を見て明日の快晴を想起したりする指標、目の充血から疲労を読み取る徴候、さらには宗教や芸術なども含まれる。さらに、ドイツの哲学者カッシーラは、「人間は、言語的形式、芸術的イメージ、神話的シンボル、宗教的儀式の中に完全に自己を包含してしまったので、人為的な媒介物を介入させないでは、なにものをも見たり聴い

たりできない。」とまで言い切っている。

なお、もう一つ付け加えておくと、オグデンとリチャーズによると、記号はそれが指し示そうとしている指向対象と直接に関係するのではなく、指向的意味を介して間接的に関係すると言われている[加藤83]。なお、指向的意味とは一言で言えば「概念」に相当するが、普遍的な概念だけでなく、記号を解釈しようとしている人の個人的な興味、気分、背景などの主観的な要素も含む。

コンピュータという記号を例にとって考えるならば（その表現形式は文字に限られず、例えば音声であってもよい)、販売店に並べられている個々の具体的な製品が指示対象となる。指示的意味には、例えば普遍的概念としての「計算処理を実行したり、情報を蓄えておくもの」といった意味の他に、ある人には「絶対に間違いを起こさないが、逆に融通が利かない」といった意味付けが、またコンピュータ関連株で儲けた人にとっては「金のなる木」といった意味付けが与えられるであろう。

コンピュータ技術との係わりの上で重要なことは、上記のモデルを使って陰喩（比喩）が説明できるということである。すなわち、陰喩はコンピュータ環境における視覚化技法として、最も一般的なアイコンのイメージ設計を行うにあたっての手助けとなる。

ところで、絵シンボル（アイコン）を情報伝達手段として利用するという考えがコンピュータの世界で現実のものとなったのはほんの20年ほど前からであるが、一般の人間社会では、言うまでもなく太古の昔から行われている。例えば、およそ2万年前のスペインのアルタミラの洞窟壁画を始めとして、紀元前5千年頃にはメソポタミア地方でシュメル文字が使われていたし、古代エジプトでも同じように象形文字が使われていた。我々が馴れ親しんでいる漢字も、かなり抽象化が進んではいるけれども、元を辿れば古代中国の象形文字に源をおいたものであり、文字の一つ一つは基本的にはものの形状に基づいて組み立てられている。このように、古代言語はアイコンと非常に近い関係にあるといえる。コンピュータ世界での視覚情報利用について言及する前に、それ以外の分野での視覚化の試みについて概観しておくことにする。

まず、グラフィックアートの分野ではピクトグラムと呼ばれる考え方がある。文献 [太田87] によると、ピクトグラムには「グラフィックシンボルの典型であって、意味するものの形状を使ってその意味概念を理解させる記号」という定義が与えられている。これは、コンピュータエンジニアが思い浮かべるアイコンの定義と基本的には同じである。ただし、予め学習しておかないと理解できないようなものはピクトグラムには含まれない。ピクトグラムとなり得るぎりぎりの境界線上にあるものの例として、感嘆符（！）や疑問符（？）がある。逆に、ピクトグラムには含まれないが、それにかなり近い記号の例としては駐車場のPなどがある。

さて、アイコンという用語はギリシャ語の「像」を意味するエイコンに語源を発している。記号論の世界では、パースは記号を①類像、②指標、③シンボルの三つに区分することを提唱したが、ここでいう類像がまさにiconである。ただし、この分野ではそれは通常フランス語読みでイコンと呼ばれ

る。宗教美術の世界においてもイコンと呼ばれる範疇のものがある。これは、教会や家庭内に置かれて礼拝の対象とされていた板絵のことであり、ビザンチン世界において聖画像として一般化された。そこでは一定の図像学的な規則に従って聖母子、聖者、使徒らが配置される。これは主に製作技法上の特質からきているが、様式が類型化され、模倣されることも多く見受けられた。以下では、この流れを汲んだイコノグラフィーという考え方についても簡単に紹介しておく [加藤83]。

イコノグラフィーは美術史の世界で用いられている用語であるが、様々な、特に宗教美術に現われる像の記述や分類などを支援する枠組みとして位置付けられる。例えば、リーパは、抽象的な概念を視覚化するための表現手段として女性像を用いた寓意化を提示し、衣装、装飾品、あるいは仕草などによる意味の分節化の可能性を説いた。

一方、イコノグラフィーを踏まえて、新たにイコノロジーという考え方も提唱されている [加藤83]。これは、単に絵画制作上の手助けとしての知識体系というよりも、むしろ作品解釈、つまり形象のうちに表現された象徴的、教義的、神話的な意味の理解のための方法論として捉えられる。言い換えれば、イコノグラフィーはモチーフであるとか物語、寓意などの、かなり表面的な意味を取り扱っているが、これに対してイコノロジーは深いレベルに秘められている基本的原理まで捉えようとしている。ただ、基本的原理を理解しようとすると、作品が生まれた時代背景や作者の生活環境などを含めた広い知見が要求されることは言うまでもない。

さて上に見たように、コンピュータ以外の世界でも積極的に研究が進められてきているアイコン（あるいはイコン）であるが、コンピュータの世界においては、直接操作という対話技法と相俟って、人々の生活スタイルに近い非常に親しみやすい操作環境を提供している。いわゆるグラフィカルユーザインタフェースである。

アイコンを用いることの最大の利点は、一見してその内容が把握できるというところにある。ガテーニョの言葉を借りると、「視覚はのみこみが速く、分析的・総合的で、その働きは光の速さに匹敵すると同時に、ほとんどエネルギーを要せず、我々の精神は数分の一秒にして無限の情緒を受容し、把握する」という特性を持っている。さらに、ケペッシュは「発言や語彙、文法の限界はない。文盲の人も、そうでない人も同じように感じる。そして、より広く、深い範囲で真理と思想を伝達することができる」と特徴付けている [吉岡83]。

このように、視覚情報を活用することで、現実社会に見られる文字文化の溝を埋めることができると期待されている。これは取りもなおさず、コンピュータ専門家と非専門家との溝を埋めることができるということを意味しており、コンピュータ利用者層が急速に拡大している今日にあっては、欠くべからざる項目となってきている。ところが、これまでに開発されてきているシステムの多くは、Macintosh で提案されたアイデアの枠にほぼ留まっているといえる。具体的に言えば、コマンドインタフェースの可視化といった範囲にあるものがほとんどである。したがって、定型作業であっても、

ユーザはその都度処理記述を行わなければならず、余計な手間をユーザに強要することになる。

そこで、まとまった一連の作業を（視覚的に）登録定義することができるような機能が意味を持ってくることになる。これが視覚的プログラミングである。なお、この場合はプログラムを記述する言語自体が視覚的であり、この意味で、ソフトウェアの理解を助ける目的で、従来の文字ベースのプログラミング言語で書かれているプログラム構造（広くはソフトウェア構造）を視覚的に表示する「ソフトウェアの視覚化」のアプローチと異なる。ソフトウェアの視覚化については3章を参照されたい。ただし注意しておくが、言語構造を基本的に文字に頼っている場合は、例えGUI環境で動作していても、それらは視覚的プログラミングの範疇には含めない。

参考までに、視覚的プログラミングに相当する考え方がグラフィックアートの分野でも見受けられる。例としては、太田によって提案されているロコスがある[太田87]。そこでは、対象あるいは動作などを表す絵シンボルが用意されており、それらを組み合わせることによって文章を視覚的に記述することができるようになっている。

さて、コンピュータ技術者の間で視覚的プログラミングという用語が受け入れられるようになったのは80年代の初めである。それ以前にも、現時点で捉えれば視覚的プログラミングの範疇に含めてよいシステムも存在するが、その時点では個々の応用分野で個別に開発が進められていた。例えばCADツールがそれに相当する。さらに、視覚表現形態としてはアイコンが最もよく用いられているが、その他にも表やフォームなどといった種々のものがある。個々の具体的なシステム事例の紹介並びに分類は次節に委ねることにし、以下では視覚的プログラミング開発に係わるいくつかの注目すべき点について述べる。

まずは視覚表現の解釈に関してであるが、視覚的プログラミングシステムの利用者は、与えられた視覚表現を通して、視覚表現設計者が設計時に頭に描いた概念を理解しようとする。ただしここで、視覚表現は見る側の判断に任される部分が多く、必ずしも両者の解釈が一致するとは限らない。そのような意味の不一致が起こると、システムは実用から遠い存在になってしまう。これまで視覚的プログラミングが期待通りには普及していない原因の一つとして、そのような点に十分な配慮がなされていなかったことが挙げられよう。

一般に、現実に存在する「もの」に対応する視覚表現を設計することは比較的簡単であるが、機能やファンクションを視覚化しようとする場合はかなりの困難を伴う。こういった問題を回避するために、コンピュータの世界では、適用範囲を限定するといったことが通常行われる。そうすることによって、設計者と利用者の意識の違いを最小限に押さえ込もうとする。しかし逆に、「視覚情報が備えている多様性を素直に受け入れ、むしろ積極的にそれを利用しよう」と考えるアプローチもある。そのような例として、HI-VISUALシステムを簡単に紹介する[Hirakawa94, Hirakawa91]。

HI-VISUALは、アイコンを言語要素として持つ視覚的プログラミングシステムである。図1.1-1にイ

ンタフェース画面例を示す。書類、筆記用具、書棚、人間など、画面上に現われているものはすべてアイコンとして管理されている。これまでの一般のシステムでは、個々のアイコンにはただ一つの意味しか許されていなかったのに対して、HI-VISUAL では、アイコンは複数の異なった意味を持つことができる。しかも、データとファンクションも区別していない。

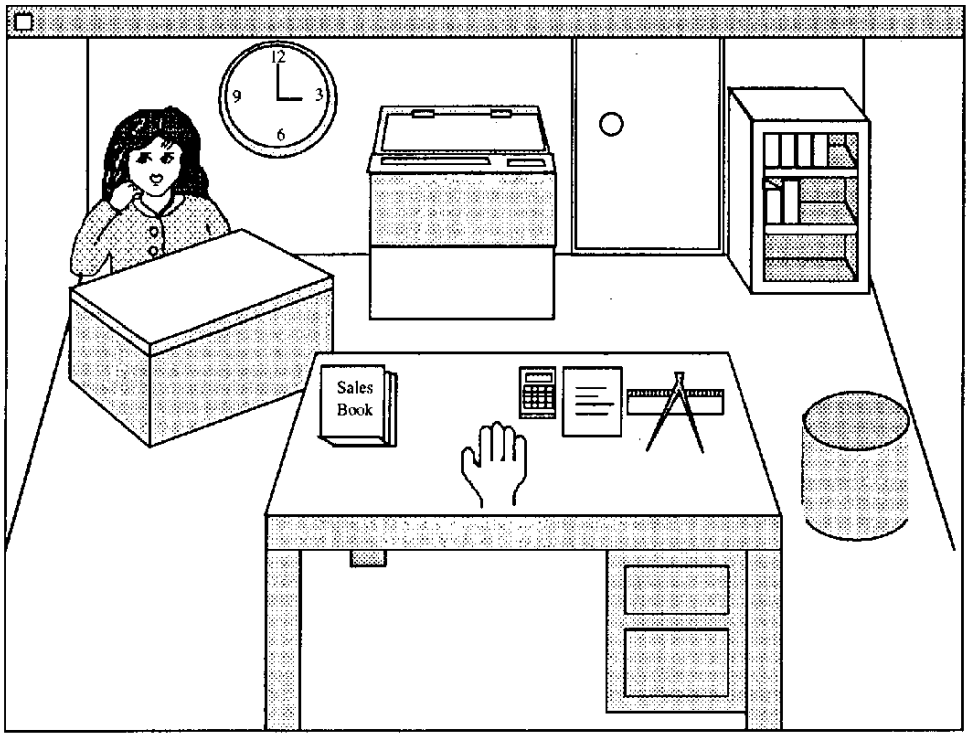


図1.1-1　HI-VISUALのインタフェース画面例

例として、書類というアイコンを考える。これには、文章を記録するものという、いわゆる受動的な性質だけでなく、「包む」であるとか「転写する」などといった能動的な性質も併せて記述される。逆にいえば、個々のアイコンの意味は、それ自身では一意的には決まらない。HI-VISUAL では、アイコンを画面上で重ね合わせることによってプログラミングを行うが、そのアイコンの組合せに応じて、それぞれのアイコンの働きが決定されると考える。

例えば、書類に鉛筆が重ね合わせられたとする。システムは、この組合せに対して、例えば「書く」という機能を想起する。この場合、鉛筆が能動的な働きをして、書類は受動的な役割を担うわけである。ただし、ここで想起される機能はただ一つに限られない。先ほどは鉛筆が書類に対して能動的に機能したが、今度は逆に書類が能動的な働きを担い、その結果として「包む」という機能が想起されることもあり得る。ただ、ある時点でのアイコン組合せに対して、ユーザが意図している機能はただ一つである。複数の機能候補中から所望の一つを選び出すにあたり、プロトタイプシステムでは、ユ

ーザがメニューから選択するようになっている。知識ベースの支援の下に、状況に応じてシステムが自動的に最適な機能を選び出すような研究も併せて行われている[Hirakawa91]。

なお、先に述べたように、画面上の個々のアイコンはそれぞれ具体的なデータ値を持っている。アイコンの組合せを通して、ある機能が選択・実行されると、それに関連したデータ値も併せて変わる。もちろん、その実行結果の値を見ることもできるようになっており、ユーザは動作を確認しながら具体的にプログラミングを進めていくことができる。また、処理によっては新たなアイコンが生成されたり、逆に不必要になったアイコンが消滅するといったことも生じる。例えば、書類をコピー機に重ねた結果として「コピーする」という処理が起動されると、コピーされた書類のアイコンが新たに画面上に現われることになる。

上のようにして記述された一連の処理の記述、いわゆるプログラムは「秘書」アイコンに登録される。例えば、「売上げ帳と電卓を重ね合わせて月別の売上げ集計結果が得られ、それに鉛筆とコンパスを重ね合わせることで集計結果のグラフが得られた」とする。この一連の操作を秘書アイコンに対応付けておくと、以降では「秘書に売上げ帳を手渡す（重ね合わせる）」だけで先ほどの処理が自動的に実行される。

なお、複数の秘書アイコンに個別の独立した分野の仕事を割り当てておくこともできる。例えば、秘書Aには経理に係わる仕事を担当させ、秘書Bには在庫管理を受け持たせることで、ユーザのプログラム管理の支援を行っている。

ところで、HI-VISUAL では、個々のアイコンはそれぞれ具体的な値を持っており、ユーザは具体的な値を持ったアイコンに順次機能を実行させながら処理を記述していく。言い換えれば、それぞれの段階の処理の効果を具体例で確認しながら「処理の記述を例示的に進めている」と言える。これに対して、処理の全体像を把握し、抽象化してからプログラムを書き始めるといった従来のプログラム開発手法では、ソフトウェア開発者に論理的思考能力が要求されていた。我々人間の作業修得パターンを考えると、そこでは具体的な例を通して順次知識を獲得していく。例示によってプログラムを積み上げていくことができれば、一般の人々にとっても馴染みやすいものだと言える。こういった手法は例示的プログラミング（example-based programming）と呼ばれ、プログラム開発の作業形態を質的に大きく変える起爆剤になり得るものとして期待されている。事実、ユーザインタフェースの世界では最近特に注目されているテーマであり、数多くの研究報告がなされている [Cypher93, 萩谷91]。本分野の研究動向については1.4節に詳述される。

初期の視覚的プログラミング言語／システムは既存の文字ベースの言語（例えばPascalやLisp）に単に視覚的なインタフェースを被せたものであった。今後この分野の研究がより一般的に受け入れられるには、例示処理モデルなどといったような視覚的プログラミングにとって有効なモデルの上に立ったシステム開発が欠かせない。

一方で、マルチメディア時代を反映して、視覚情報に限らず、もっと多彩な形態の情報をプログラミングに活用しようという動きが活発化しつつある。マルチメディア／マルチモーダルプログラミングと呼ぶに相応しい研究領域であり、本報告書の2章で関連技術動向がサーベイされている。なお、言語（処理系）やシステムだけでなく、ソフトウェア開発方法論までも含めて、これからのマルチメディア時代を意識したプログラム（ソフトウェア）開発形態を模索する動きとして、マルチメディアソフトウェア工学という新しい提案もある[平川94]。

【参考文献】

[Cypher93] A. Cypher (ed.), Watch What I Do: Programming by Demonstration, MIT Press, 1993.

[萩谷91] 萩谷, 視覚的プログラミングと自動プログラミング, コンピュータソフトウェア, 8, 2, pp.27-39, 1991年3月.

[Hirakawa90] M. Hirakawa, M. Tanaka, and T. Ichikawa, An Iconic Programming System, HI-VISUAL, IEEE Transactions on Software Engineering, Vol. 16, No. 10, pp. 1178-1184, October 1990.

[Hirakawa91] M. Hirakawa, Y. Nishimura, M. Kado, and T. Ichikawa, Interpretation of Icon Overlapping in Iconic Programming, Proc., IEEE Workshop on Visual Languages, pp.254-259, 1991.

[Hirakawa94] M. Hirakawa and T. Ichikawa, Visual Language Studies - A Perspective, Journal of Software - Concepts and Tools, 15, pp.61-67, 1994.

[平川94] 平川, マルチメディア・ソフトウェア工学, 情報処理学会変革期のソフトウェア工学シンポジウム論文集, pp.27-35, 1994年9月.

[加藤83] 加藤、持田、谷川、中川、芸術の記号論、勁草書房、1983年.

[太田87] 太田幸夫, ピクトグラム[絵文字]デザイン, 柏書房, 1987年.

[吉岡83] 吉岡徹, 基礎デザイン, 光生館, 1983年.

1.2 視覚的プログラミングの分類

1.2.1 視覚的プログラミングの背景

視覚的プログラミングとは、複雑なことがらをシンボル化し、コンピュータにより直接操作できるものに置き換える技術であり、従来の文字列や記号を中心としたインタフェースに代わって、アイコン、図形、アニメーションという人間がより理解しやすいビジュアルな表現を用いて人間とコンピュータの相互作用を行う技術である[Gilnert90a, Gilnert90b, Shu88]。

このような視覚的プログラミングに関する研究が、最近注目を浴びるようになった背景的要因としては、以下のような理由が挙げられよう。

① 1980年代に入ると、マイクロプロセッサの高性能化に伴い、従来の汎用コンピュータに代わりワークステーションやパーソナルコンピュータが出現し、個人ベースで豊かなプログラミング環境を専有できる状況が生まれた。

② 1980年代にパーソナルコンピュータとして、デスクトップメタファに基づくMacintoshが発売された。これにより、従来のコマンドベースのインタフェースと比較し、よりヒューマンフレンドリなインタフェースが求められるようになった。また、1990年代にパーソナルコンピュータの高機能化が進み、そのGUIとしてWINDOWSが広く普及した。

③ ワークステーション上でのUNIXの普及が始まり、その上で汎用のウィンドウシステムの開発が行われた。また、UNIX上のクライアントサーバ型のX Window SystemがMITで開発され世界中に無料で配布された。さらには、X Window Systemに基づき、Motifなどのグラフィカルユーザインタフェース（GUI）が開発された。

④ 従来は、ハイエンドの特殊装置を必要とした3次元グラフィクスなどもマイクロプロセッサの発達に伴い、通常のワークステーション、パーソナルコンピュータ、ゲーム機などで一般に用いられるようになった。

これらの理由により、視覚的プログラミングは、今後きわめて「手軽」で「実用的」な技術となっていくことが予想される。また、グラフィカルユーザインタフェース（GUI）という言葉に象徴されるように、単なるグラフィクスによる表示だけでなく、ユーザとのインタフェース、あるいはインタラクションが、今後よりいっそう重要になっていくと考えられる。

1.2.2 視覚的プログラミングのメリット

通常、プログラムはテキスト形式で記述されることが多い。しかし、このような形式でプログラムを理解することはあまり直感的ではない。そのため、人間は図、表、絵などを用いてその動作原理を表現することをよく行う。

絵やアニメーションなどを用いてプログラムを表現したりその実行の様子を表示することができれ

ば、見る人の持つ直感を視覚的に刺激することができる。さらに絵などによって得られる情報はテキストによって得られる情報よりも遥かに豊富である。つまりテキストとしてではなく絵として情報を表示することにより、実際の情報量としては同等のものを表示しても、はるかに豊かな情報をユーザに獲得させることができる[田中95]。

「視覚的プログラミングのメリット」は、以下の3点にまとめることができよう。

① 見栄えのよさ

ソフトウェアの「見栄え」の点でGUIを用いたシステムは従来のシステムに比べ優れている。すなわち、入出力にGUIを用いて、各種のボタン、メニュー、スクロールバーなどを操作できるようにすれば、画面に計器板が並んでいるような感覚で操作を行うことができる。

② 初心者用

実際にプログラミングやオペレーティングシステムの使用経験のないエンドユーザには、従来のようなコマンドベースの入出力に抵抗感がある。しかしながら、視覚的環境をマウスを用いて操作することにより、抵抗なくシステムを操作することができる。視覚的システムは、CAIなどの教育用、テレビゲームなどの娯楽用の用途にも適していると考えられる。

③ ヒューマンフレンドリ

初心者を対象とするだけでなく、熟練者にも視覚的プログラミングは有用と思われる。視覚的環境は人間に対して自然であり、視覚的環境とマウスによるインタフェース／インタラクションをうまく用いれば、「疲労度」や「誤り」の少ないヒューマンフレンドリなインタフェースが実現すると考えられる。

1.2.3 広義 v.s. 狭義の視覚的プログラミング

市川は、「視覚的プログラミング」を以下の3つの分野に分類している[Grafton85, 市川88, 紫合88]。

① ソフトウェアライフサイクルのあらゆるフェーズを、ソフトウェアの状態を目で確かめられるようにしようとするソフトウェアの視覚化。

② データのフローやコントロールのフローを可視化するアルゴリズムアニメーション。

③ プログラム作りに高次元のグラフィックなシンボルを用いようとするグラフィカルプログラミングあるいはビジュアルプログラミング。

「ソフトウェアの視覚化」とは、その言葉のとおり、ソフトウェアをビジュアルに表示することである。ソフトウェアライフサイクルとしては、要求解析、ソフトウェア設計、コーディング、テスト、運用、廃棄などのフェーズが考えられるが、このそれぞれのフェーズに対応して視覚化が考えられる(これらは、いわゆるCASEツールと呼ばれるものを含む)。「アルゴリズムアニメーション」は、プログラムやテキストに記述されたアルゴリズムについて、アニメーションを用いて「抽象度の高い」視

覚化を行う。「ビジュアルプログラミング」は、最初にプログラミングを行う時から、テキストではなく図形を組み合わせる形でプログラミングを行おうとするものである。

なお、これらには含まれていないが、最近、流行しているものとして

④　インタフェースの視覚化

がある。これは、アプリケーションプログラムのインタフェース環境をユーザが視覚的に構築できるようにするもので、いわゆるVisual Basic、Visual C++、インタフェースビルダなどはこれに含まれると考えられる。

「視覚的プログラミング」という言葉は、様々な文脈の中で使用され、その意味するところも様々である。「視覚的プログラミング」を広義に捉えれば、プログラミングが何らかの意味で視覚的要素を含むならば「視覚的プログラミング」であり、上記の①～④すべてが「視覚的プログラミング」ということになる。一方、「視覚的プログラミング」について、この言葉をもっと限定的に使用する立場も存在する（狭義の立場）。この場合、「視覚的プログラミング」は、上記の③だけを指す言葉として使用されることが多い。

ここでは、視覚的プログラミングについて広義の立場をとり、その分類について論じる。

1.2.4　Myersの視覚的プログラミングの分類

視覚的プログラミングの分類で、特によく引用されるのはMyersの分類[Myers90]である。

Myersは、まず、従来においてVisual ProgrammingとProgram Visualizationが十分に区別されてこなかった状況を分析し、視覚的プログラミングという言葉の代わりに視覚的言語（Visual Languages）という言葉を使うことを提唱している。

すなわち、視覚的言語（Visual Languages）とは、視覚的なアクティビティの総称であり、その中に、Visual ProgrammingとProgram Visualizationの二つの大きなカテゴリがある。Myersは、このそれぞれについて分類を行っている。

なお、前項の市川の分類とMyersの分類との関係であるが、市川の分類は、ソフトウェアの視覚化までを含む広いものであるのに比べ、Myersの分類はプログラミングに焦点を当てたものになっている。強いて両者を対応させれば、市川の分類の①、②を合わせたものがMyersの分類のProgram Visualization、市川の分類の③がMyersの分類のVisual Programmingに相当する。

1.2.5　MyersによるVisual Programmingの分類

Myersは、Visual Programmingの分類基準として、

・例示によるシステムか？

・インタプリティブかコンパイラベースか？

の二つを挙げている。

Visual Programmingのシステムには例示を用いるもの（Example-based programming）が多い。Example-based programmingは、例の用い方により、Programming by ExampleとProgramming with Exampleとに分類される。

Programming by Exampleでは、例えば、プログラムを実行した時に期待される入出力の対、あるいは期待される実行トレース例を与えることにより、システムが自動的に推論を行ってプログラムを合成する。一方、Programming with Exampleは、具体例を用いながらプログラムを行っていく。Programming by ExampleとProgramming with Exampleの違いは、前者が推論の機構を持つのに比べ、後者は推論の機能を持たないことである。後者の場合、使用される例によりプログラムが合成されることはないが、プログラマは例を使いながらプログラミングを行うことができる。

二つ目の、インタプリティブかコンパイラベースか？という視点は、対象となる言語の実装がインタプリティブに行われるのか、それともコンパイラベースで行われるのかという視点である。インタプリティブなほうがインタラクティブで柔軟であるが効率に欠ける。反面、コンパイラベースの場合には効率的であるが柔軟性に劣る。

例えば、PIGS、 Pict、 PROGRAPHなどのシステムは、例示に基づかないインタプリティブなシステムとして、また、FORMAL、 OPALなどは、例示に基づかないコンパイラベースのシステムとして、Rehersal World、 HI-VISUALなどは、例示に基づいたインタプリティブなシステムとして分類される。

1.2.6 BurnettによるVisual Programmingの分類

MyersのようなVisual Programmingの分類ではなく、Visual Programmingの文献を整理するための基準として、BurnettはACMのComputing Reviewsの分類システムを拡張する形式で、以下のような分類を提案している[Burnett94]。

VPL：Visual Programming Languages （VPL）

VPL-I： VPLのためのツール、環境

VPL-II：言語の種類

VPL-III：言語の特徴

VPL-IV： 言語の実装法

VPL-V：言語の目的

VPL-VI： VPLの理論

「VPL-II：言語の種類」では、さらにA：パラダイム、B：視覚的表示法に分類し、「A：パラダイム」としては、

1. 並列言語

2. 制約言語
3. データフロー言語
4. フォームベース、スプレッドシートベース言語
5. 関数型言語
6. 既存の手続き型言語
7. 論理型言語
8. マルチパラダイム言語
9. オブジェクト指向言語
10. Programming-by-demonstration言語
11. ルールベース言語

などに分類し、「B：視覚的表示法の分類」としては、

1. ダイヤグラム言語
2. アイコン言語
3. 静的な絵の列に基づく言語

などを挙げている。この分類で分かるようにVisual Programming言語としては、既存の手続き型言語ではなく、宣言型言語、あるいは高水準言語が用いられることが多い。これは、Visualな世界が、手続き的でなく宣言的であることに由来すると思われる。

また、「VPL-V：言語の目的」では、Visual Programming言語を

A：汎用言語
B：データベース言語
C：画像処理言語
D： 科学計算視覚化言語
E：ユーザインタフェース生成用言語

などに分類している。

最後にBurnettは、「VPL-VI：VPLの理論」として、

A：VPLの形式的定義
B：アイコン言語
C：言語設計の諸問題

などを挙げている。

1.2.7 Program Visualization

Program Visualization（プログラムの視覚化）とは、記述されたプログラム、データや、その実行

の様子を視覚的に表現することにより、ユーザに分かりやすく提供しようとする試みを指す。Program Visualizationとは、既に書かれたプログラムを視覚化するシステムであり、プログラミングそのものを変化させるわけではない（この点がVisual Programmingとの相違点である。Visual Programmingの場合には、最初から図形的にプログラムを作成する）。

通常、視覚的プログラミングの試みはProgram Visualizationから始まるが、やがて、既存の言語を視覚化するだけでは物足りなくなり、Visual Programming、すなわち、プログラム要素が2次元（あるいは3次元）の広がりを持つ、新たな言語の提案へと結び付いていくのが常である。例えば、Macintosh上で稼働するPrographにしても、当初はPrologを可視化したLOGRAPHという言語の提案から始まり、やがて、図形がテキストから独立し、図形だけで直接実行やデバッグができるようにしたものである。

図形によるプログラム可読性向上の試みは、決して新しいものではない。FortranやPascalのプログラミングの際に書かれるフローチャートは、図形プログラミング言語の元祖である。二村によれば、最初の流れ図が、計算機関連の論文の中に現れたのは1947年である[二村84]。世界最初の計算機ABCマシンが開発されたのが1939年、ENIACが完成したのが1946年であるので、それこそ流れ図は、計算機が誕生した時からあることになる。

60年代の終りにDijkstra等により、構造化プログラミングが提唱されるようになると、流れ図におけるプログラム構造の見にくさなどの欠点も明らかになり、70年代にはそれらの欠点を解消し、プログラムの表現にチャートやダイヤグラムを用いた多数の「構造化プログラム図式」が提案された。PADもその一つである。当初、構造化プログラム図式は、既存のテキストを図式化することから出発したが、次第にテキストと図形の相互変換が図られるようになった。また構造化プログラム図式の普及と計算機のグラフィック処理能力の向上につれ、様々なソフトウェアツールが開発されるようになってきた。そうしたツールとしては、「図式を入力、変更、清書する図形エディタ」、「図式をテキストプログラムに変換するコンパイラ」、「テキストプログラムを図式に変換する自動作図プログラム」などが挙げられる。

このように、静的なコードの視覚化から始まったProgram Visualizationも、現在では、要求解析、ソフトウェア設計、コーディング、テスト、運用、廃棄などのソフトウェアライフサイクルのすべてに渡るようになっている。また、視覚化の関心も静的な視覚化から動的なプログラム実行の視覚化に移っている。

1.2.8 MyersによるProgram Visualizationの分類

MyersはProgram Visualizationの分類基準として、以下の基準を提案している[Myers90]。

すなわち、視覚化の対象について、

・コード

・データ

・アルゴリズム

の三つを挙げている。

また視覚化の性質について、

・視覚化が静的か

・視覚化が動的か

などを挙げている。

例えば、通常のフローチャートやPegasysなどのシステムは、静的なコードの視覚化の例である。また、動的なコードの視覚化としてはBALSAの例があり、実行中のコードがハイライト表示される。

また、アルゴリズムの視覚化としては、動的、アニメーション的にアルゴリズムを表示するもの（すなわちアルゴリズムアニメーション）が多い。アルゴリズムアニメーションはProgram Visualizationの一分野であるが、近年、積極的に様々な研究が行われている。

1.2.9 RomanによるProgram Visualizationの分類

MyersによるProgram Visualizationの分類と比べ、より詳しい分類尺度を提供しているのが、Romanの分類[Roman93]である。Romanは、Program Visualizationの分類尺度として以下の5つを提案している。

1.	視覚化対象	コード、データ、制御、振舞い
2.	抽象度	直接表現、構造表現、総合表現
3.	仕様記述法	組込み、注記、宣言、例示
4.	表示法	解析的、説明的、編成的
5.	インタフェース	図形語彙、インタラクション

視覚化対象であるが、これは何を視覚化の対象とするかということである。コード、データ、制御などが視覚化の対象として挙がっている。また、振舞いというのは、プログラムの状態の変化を時系列に沿って並べたものである。この振舞いを視覚化することにより、より高度の視覚化効果が得られる。

抽象度に関しては、直接表現、構造表現、総合表現の順に抽象度が高くなる。直接表現では、プログラムやデータを直接的に表現するだけであるが、構造表現の場合にはプログラムやデータの一部に焦点を当てたり、不必要な部分を省略したりして抽象度を上げている。また、総合表現とは構造表現をより進めたもので、アニメーションのためプログラムに直接現れない情報をも付加して表現しようというものである。

また、仕様の記述に関しては、システムに直接組み込んでしまう方法、プログラムにアノテーションの形式で注記を行う方法、プログラムに宣言の形で含める方法、例示による方法などに分類している。例示による方法とは、例えばサンプルとなるデータに対してジェスチャの形でお手本を見せると、後はそれを真似してくれるという方法である。

表示の方法には、データを数値的、解析的に表示する場合と、視覚事象を積極的に用い説明的に表示する方法がある。編成的表示とはもっと複雑な場合で、同じプログラムに複数の入力を与え同時に表示したり、同じ情報に基づき複数の表示形式を同時に表示するような方法である。

また、インタフェースは、図形語彙とインタラクションに大別される。図形語彙とは、視覚化の道具としてどのような描画要素が用意されているかということであり、点、線、円、多角形、テキストなどの単純図形、それらを組み合わせた複合図形、また、時間的に離散的した事象を連続的に変化するように見せるアニメーションのための視覚的事象のサポートなどが含まれる。インタラクションは、システムの「処理の制御」及び「表示の制御」の目的で使われる。

Program Visualizationの最近の動向であるが、この分類尺度でも分かるように、振舞い、総合表現、例示、編成的などを重視したより複雑なメカニズムが次々に開発され、インタフェースの図形語彙についても一部では3次元的な表示の使用が始まっている。

また、並列プログラムが一般的になるにつれ、並列プログラムの実行の様子を様々な視点から解析的に表示するParaGraphのようなシステムも現れている。

1.2.10 Visual ProgrammingについてのFAQ

インターネット上のニュースグループの一つにcomp.lang.visualというニュースグループがあり、このニュースグループに投稿される記事を読むことにより、Visual Programmingに関する一般の人たちの関心の動向を読むことができる。

そこで、毎週投稿されている記事に、Visual ProgrammingについてのFrequently Asked Questions（FAQ）がある。このFAQには、

1. Visual Programming Languageとは何か？
2. Visual BasicやVisual C++は、Visual Programming Languageと呼べるか？
3. Visual Programming Languageの種類（研究用、商品）
4. 参考資料（本、雑誌、会議録、ネットワーク上の情報）
5. Visual Programmingに関する論文の分類プロジェクト
6. Visual Programmingについての懐疑的意見
7. Visual Programming Language設計のためのツールキット一覧

などのFAQ及びそれに関する模範回答がまとめられている。このニュースグループでは、「視覚的プ

ログラミング」に関して狭義の解釈をとっており、Visual BasicやVisual C++は、Visual Programming Languageの中に含めていない。

1.2.11 視覚的プログラミングの課題

Myersは視覚的プログラミングの課題として、以下のような点を挙げている[Myers90]。また、田中は既存のVisual Programmingの問題点を挙げている[田中92]。これらを整理すると以下のようにまとめられる。

(1) 大規模プログラムへの対応の欠如

一般にVisual Programmingのシステムは試験的に作られることが多く、機能も基本機能しかサポートされない。プログラムもほとんどがトイプログラムであり、大規模な実用プログラムが書かれることも少ない。したがって、Visual Programmingのシステムはいつまで経ってもデモ用のシステムのままである。一般に、プログラムは図で記述すると量が増加するので、スクリーン上に収まらない図形を如何にして表示するかという問題を解決する必要がある。スクロールやモジュール化、抽象化のメカニズムに関して研究の必要がある。

(2) 描画アルゴリズム、自動レイアウトの必要性

図形やアイコンを組み合わせてプログラムを作るというのは一見簡単そうであるが、大規模プログラムの場合には手間がかかる。また、図形による描画についても、特に大規模プログラムの場合、現在のシステムでは、表現があまりにも貧弱であり、理解、編集、デバッグなどが困難になる。描画のアルゴリズムや例示による自動化の手法に関して、もっと研究を行う必要がある。

(3) Visual Languageの形式的な仕様がない

現在までのところVisual Languageの研究開発は経験的に行われており、言語の仕様を形式的に記述する方法が存在しない。Visual Languageの研究がHard Scienceであるためには、言語仕様を形式的に記述する方法として、例えばテキスト言語のBNFに相当するような方法を考える必要がある。

(4) 処理系や環境が困難、ポータビリティの欠如

Visual Languageでは図形を扱うので、エディタ、コンパイラ、デバッガなどの処理系や環境がテキストベースの場合と比べより複雑になる。また、ポータビリティの問題も生ずる。

1.2.12　視覚的プログラミングの最近の動向

視覚的プログラミングの最近の動向として、アニメーション技術の流行を挙げることができる。単に、静的に図形を描くだけでなく、アニメーション技術を取り入れることによりプログラムの実行の様子をより生き生きと示すことができる。従来、アニメーションは、Program Visualizationの一分野であるアルゴリズムの視覚化に使用されることが多かったが、Kahnはそれをプログラム実行の視覚化に適用した[Kahn92]。

KahnのPictorial Janusのプログラム例を図1.2-1に示す。このプログラムは二つのリストを結合するappendプログラムである。このシステムではプログラムの実行されていく様子がアニメーション映画のように連続的に示される。

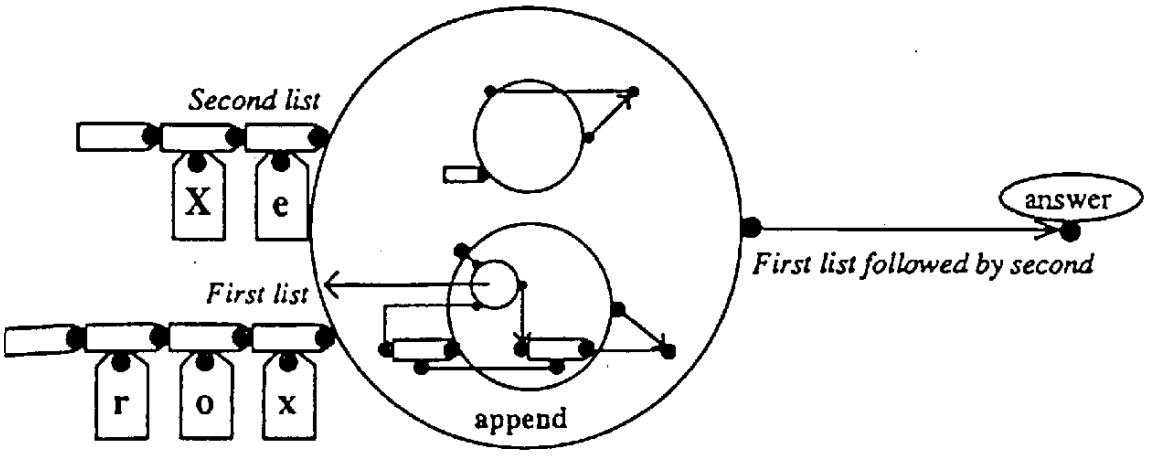


図1.2-1　Pictorial Janusによるappendプログラム

Kahnの所属したXerox PARCでは、子供のためのシステムに特別な意味を持たせることが多いが、このシステムにおいても、究極の目標とするところは子供が落書するような要領で絵を書けば、それがプログラムになり、動き出すようなシステムである。

なお、Kahnは、その後Xerox PARCを退職し、自分でAnimated Programsというベンチャー企業を設立し、ToonTalkというIBM PC上で稼働するビデオゲーム風のVisual Programming Systemの研究開発に従事している[Kahn94]。ToonTalkの意図するところは、子供がビジュアルプログラミングにより、いわゆるゲーム感覚でプログラミングを楽しむことである。

また、前項に視覚的プログラミングの課題を挙げたが、それらの問題の解決をVisual ProgrammingとProgram Visualizationの中間の方向、すなわち、両者のハイブリッドなシステムを目指したのが、田中らによるPictorial Programmingの研究である[Tanaka94, 田中94, 中野94]。ここでは、プログラムについてビジュアルな図形的な表現と通常のテキスト形式が併用されている。

Pictorial Programmingによるappendプログラムの入力例を図1.2-2に示す。

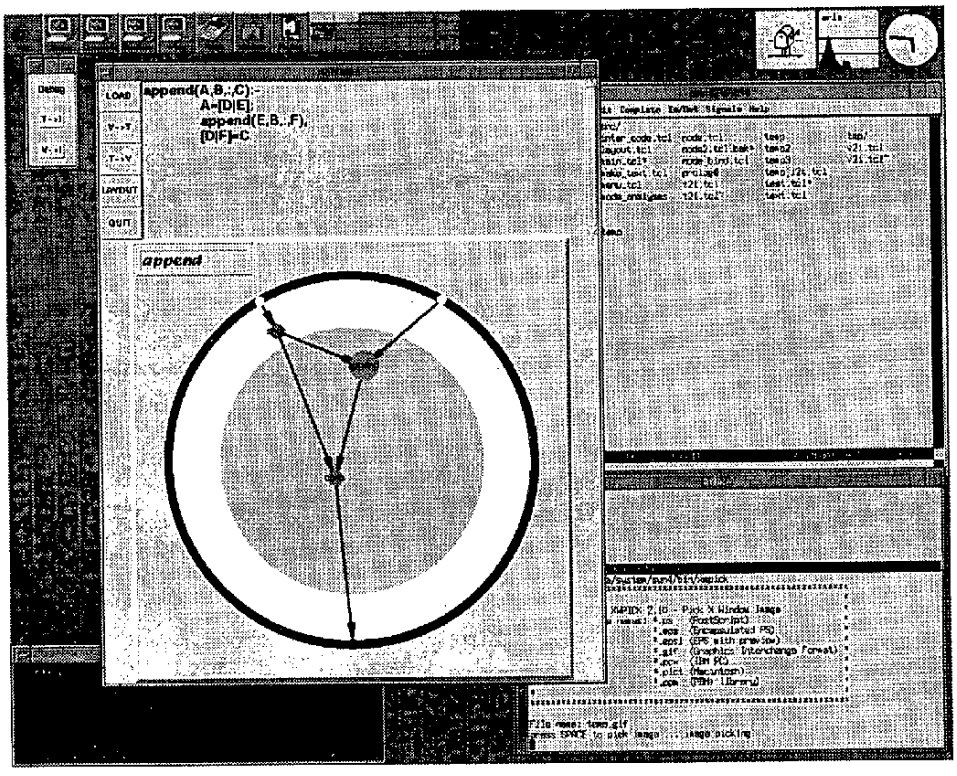


図1.2-2　Pictorial Programmingによるappendプログラムの入力例

本システムでは、通常のテキスト形式によるプログラムの表現と図形による表現の二通りの表現が常に存在するので、どちらの表現でプログラムの入力を行ってもよい。テキスト形式で入力が行われた場合には、描画アルゴリズムに従って自動的にレイアウトが定められる。

二通りの表現は互いに因果結合されており、どちらから入力をしても、それらはすぐに他方に反映される。Pictorial Programmingの特徴は、この両者の因果関係を極限まで追求したところにある。すなわち、一方の表現に変化があると、他方の表現に変換のトリガーがかかるが、その変換の粒度を、図形入力の一動作に相当するように非常に細かく設定してある。図形入力の一動作ごとに変換のトリガーがかかり、再計算が行われる様子、また逆にテキストから入力していくに従い、自動的に描画がなされていく様子は非常にアトラクティブである。

なお、ここで取り上げたPictorial JanusとPictorial Programmingは、いずれも並列論理型言語をベースとする視覚的プログラミング言語である。比較的高水準言語である並列論理型言語を用いることにより、静的な図形表現と実行時の図形表現のギャップがほとんどなく、双方とも同様な技法を用いて完全な視覚化（Complete Visualization）が実現されている。

【参考文献】

[Burnett94] Burnett, M.M. and Baker, M.J.: A Classification System for Visual Programming Languages, Journal of Visual Languages andComputing, Vol.5, No.3, pp.2877-300(1994).

[二村84]二村良彦: 構造化プログラム図式, コンピュータ・ソフトウェア, Vol.1, No.1, pp.64-71(1984).

[Gilnert90a] Gilnert, E.P.: Visual Programming Environments: Paradigms and Systems, IEEE Computer Society Press(1990).

[Gilnert90b] Gilnert, E.P.: Visual Programming Environments: Applications and Issues, IEEE Computer Society Press(1990).

[Grafton85] Grafton, R.B. and Ichikawa, T.: Visual Programming特集, Computer, Vol.18, No.8(1985).

[市川88] 市川忠男, 平川正人: ビジュアル・プログラミング, bit, Vol.20, No.4, pp.404-412(1988).

[Kahn92] Kahn, K.M.: Concurrent Constraint Programs to Parse and Animate Pictures of Concurrent Constraint Programs,Poc. FGCS'92, Tokyo, pp.943-950(1992).

[Kahn94] Kahn, K.M.: ToonTalk - An Animated Programmin Environment for Children(1994).

[Myers90] Myers, B.A.: Taxonomies of Visual Programming and Programming Visualization, Journal of Visual Languages and Computing,Vol.1, No.1, pp.97-123(1990).

[中野94] 中野勝次郎, 田中二郎:ビジュアルプログラミングシステムにおけるモデルの視覚化アルゴリズム, インタラクティブシステムとソフトウエアII, 日本ソフトウェア科学会 WISS'94, 近代科学社, pp.205-214(1994).

[Roman93] Roman, G.-C. and Cox, K.C.: A Taxonomy of Program Visualization Systems, Computer,Vol.26, No.12, pp.11-24(1993).

[紫合88]紫合治 他: 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, 情報処理, Vol.29, No.5, pp.485-504(1988).

[Shu88] Shu, N.C.: Visual Programming, Van Nostrand Reinhold(1988), (西川博昭訳,ビジュアル・プログラミング,日経BP社(1991)).

[田中92] 田中二郎: ビジュアル・プログラミングを目指して, ユーザインタフェース大作戦7, bit, Vol.24, No.7, pp.758-766(1992).

[Tanaka94] Tanaka, J.: Visual Programming System for Parallel Logic Langauges, Workshop on Parallel Logic Programming and its Program Environments, the University of Oregon, pp.175-186(1994).

[田中94] 田中二郎: 並列論理型言語GHCのビジュアル化の試み, インタラクティブシステムとソフトウエアI, 日本ソフトウェア科学会 WISS'93, 近代科学社, pp.265-272(1994).

[田中95]田中二郎, 神田陽治編: インタフェース大作戦: グループウェアとビジュアルインタフェース, 共立出版(1995).

1.3 GUIベースのプログラミング

1.3.1 GUIの歴史

現在グラフィックユーザインタフェース（GUI）という言葉は、ウィンドウ／アイコン／メニュー／ポインティングデバイスを利用する、いわゆる「WIMP」インタフェースを指すのが普通である。文字端末を使用する、いわゆる「スクリーンエディタ」なども2次元ディスプレイ画面を効果的に使用したインタフェースであるが、このようなものは通常GUIとは呼ばれないようである。WIMPインタフェースは、現在のパーソナルコンピュータやワークステーション上のインタフェースの主流となっており、携帯型端末でも同様のインタフェースを継承しているものが多い。

GUIが現在優れたインタフェース方式として広く使用されている理由として、直接操作（Direct Manipulation）が可能であることが挙げられる[Shneiderman83]。直接操作とは、マウスなどのポインティングデバイスを使用して、画面上に表示されたオブジェクトをあたかも現実の物体のように直接動かしたり拡大したりするインタフェース方式であり、ウィンドウを使用した多くのアプリケーションで採用されている。

GUIのもう一つの特徴として、キーボードをなるべく使わずポインティングデバイスのみで仕事を行うという点がある。WIMPインタフェースの出現前は、計算機の操作のほとんどはキーボードから指令されていたが、WIMPインタフェースではポインティングデバイスを使用してメニューやアイコンの選択により同等のことを行う。メニューやアイコンを使用する場合、ユーザは計算機に対する指令の形式を正確に覚えておく必要がないし、キーボードのタイピングが苦手な人でも簡単に指示を出すことができる。

GUIはキーボードのみを使用するインタフェースに比べ格段に操作性が良いことが多いため、近年非常に多くの計算機アプリケーションにおいてGUIが採用されている。

ウィンドウの操作からグラフィック描画まで、すべてのGUI処理についてアプリケーションプログラムで面倒をみることも可能であり、少し前のパーソナルコンピュータ上のアプリケーションではそのような形式のものも多かったが、現在は、ウィンドウなどの資源や入出力装置を管理したりグラフィック表示ライブラリを提供する「ウィンドウシステム」を使用し、その機能を使ってGUIアプリケーションを開発するのが普通になっている。本節では、まず現在GUIの主流となっているウィンドウシステムとそのプログラミングについて解説し、その後ウィンドウシステムの研究動向について述べる。

1.3.2 ウィンドウシステム

(1) GUI登場の背景

GUIが登場した背景には、以下のようなハードウェア／ソフトウェア技術の進歩が挙げられる。

・マウスの発明

・ウィンドウやアイコンの発明

・直接操作方式の発明

・メモリの大容量化

・グラフィクスやCPUの高速化

現在のGUIは、これらの微妙なバランスの上に成立している。ビットマップメモリを使用して矩形を画面上に重ねて並べるというウィンドウの方式や、候補リストを一時的に画面上に矩形で表示してマウスで項目を選択するというメニューの方式などは、使いやすさと実装しやすさの両者を追及した結果の妥協的着地点ということができるだろう。

(2) ウィンドウシステムの登場

GUIを実現する要素技術が得られても、それらを組み合わせてアプリケーションを構築するためには大規模なソフトウェアが必要となるので、GUIアプリケーションの作成を支援する各種のウィンドウシステムが開発されてきた。ウィンドウシステムは以下のような各種の機能を提供する。

・ウィンドウやアイコンなどの資源の管理

・グラフィック描画ライブラリ

・入出力装置の管理

・アプリケーションの協調

ウィンドウシステムの利用により、アプリケーション作成者はこれらをすべて扱う必要がなくなる。

GKSなどのコンピュータグラフィクスの規格においても、グラフィック描画と入出力装置の扱いが規定されているが、ウィンドウやアイコンなどの扱いは規定していないため、ウィンドウシステムの標準としては普及しなかった。

(3) ウィンドウシステムのアーキテクチャ

(a) 仕様

WIMP式GUIにも細かな仕様の違いがいろいろ考えられる。以下のような仕様の選択が従来議論の対象となっていた。

・ウィンドウは誰が管理するか。

・表示されるウィンドウの重なり合いを許すか否か（オーバーラップウィンドウv.s.タイリングウィンドウ）。

・ウィンドウを階層関係で管理するかどうか。

・ウィンドウへのグラフィック描画やテキスト描画をどのように扱うか。

・キーボードやマウスなどからの入力をどのように扱い、ウィンドウとどのように関連付けるか。
・描画データはどのような形式で誰が持っているのか。
・隠れていたウィンドウが表示されるとき、どのように再描画を行うか。

淘汰の結果、現在は以下のような仕様が一般的になっているようである。ワークステーションやPC上で現在広まっている各種のウィンドウシステムにおいて、これらの仕様は共通であり、ライブラリやインタフェースの違いはあるものの、外見的にもプログラミングにおいても、かなり似た構造になっているものが多い。

・ウィンドウはウィンドウシステムが管理する。
・ウィンドウは階層構造を持ち重なり合って表示される。子ウィンドウは親ウィンドウでクリッピングされる。
・グラフィックライブラリがウィンドウシステムに用意され、あらゆるウィンドウへの画面出力に使用される。
・キーボードやマウスなどからの入力や、システムからの通知は「イベント」として統一的に扱う。キーボードからの入力は階層構造の末端の（木構造の葉に相当する）ウィンドウに届く。
・描画データはビットマップ型式で持つ。
・再描画は場合に応じて、システム又はアプリケーションが行う。

(b) 役割の分担

上述のように、ウィンドウシステムはウィンドウ、アイコン、入出力装置など各種の資源を管理し調停する役割を持つ。これは、OSがファイルやメモリや計算機時間を管理し調停しなければならないのと同様である。あらゆる仕事を一つの巨大なOSが面倒をみる方式に対し、多数の小さなモジュールが別々の仕事を受け持つマイクロカーネル方式のOSが近年優勢であるのと同様に、昔は一つの巨大なウィンドウシステムがすべての面倒をみるものが多かったのに対し、最近のウィンドウシステムは機能ごとに小さなモジュールに分割されていることが多い。例えば、Sun Workstation上のSunViewウィンドウシステムとその上で動作する図形エディタでは、各種の機能は図1.3-1のようにUNIXカーネルとアプリケーションプロセスの二つに分かれて実装されるが、たくさんのライブラリがリンクされてしまうため、カーネルもアプリケーションも巨大となり扱いにくかった。

これに対し、X Window System上の図形エディタでは、図1.3-2のように処理が分担される。またNEXTSTEPでは、図1.3-3のように分担される。

このように、新しいウィンドウシステムほど多くのモジュールに役割が分担される傾向がある。

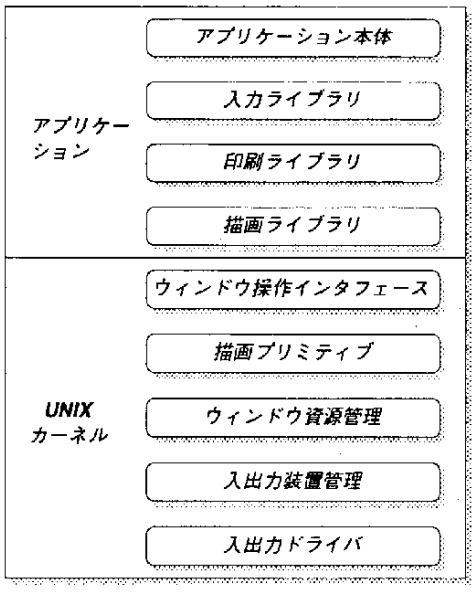


図1.3-1　SunView上のアプリケーションの役割分担

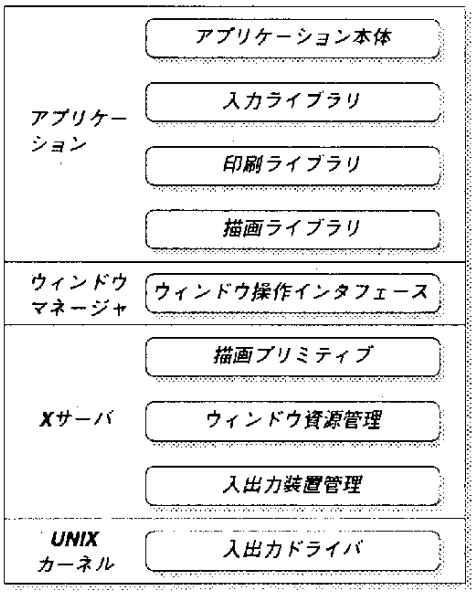


図1.3-2　X11上のアプリケーションの役割分担

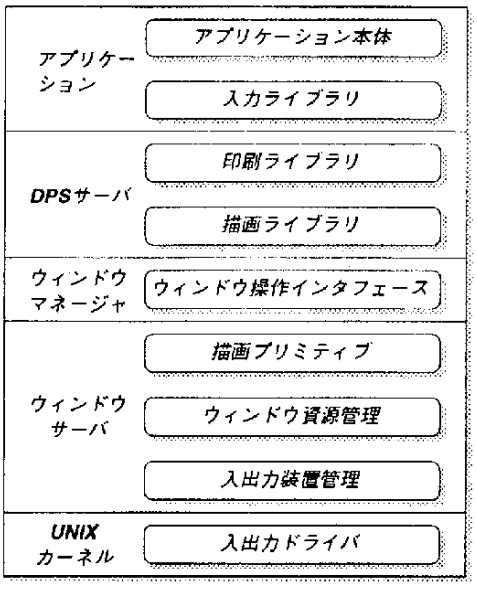


図1.3-3　NEXTSTEP上のアプリケーションの役割分担

(c)　実装

(i)　画面出力

ウィンドウへの出力は、ウィンドウシステムに固有のグラフィックライブラリが使用されるのが普通である。グラフィックライブラリとしては、コンピュータグラフィクスの分野で標準化が進められてきたGKSやCGIなどの規格が使用されたり、プリンタ用のページ記述言語PostScriptをウィンドウシステムで扱えるように拡張したものが使用されたり、独自のライブラリが使用されたり様々である。

(ii)　入力

ウィンドウシステム上のアプリケーションは、キーボードやマウスなど各種の入力装置や他アプリケーションからの要求を受け付ける必要があるが、Cのような一般のプログラミング言語では同時に複数の入力を効率良く待つことが難しいため、複数の装置からの要求をウィンドウシステムがひとまとめにして同じ形式に揃え、アプリケーションに対して「イベント」として統一的な形で送り込むのが普通になっている。各アプリケーションは常にウィンドウシステムからのイベントを待ち続け、イベントが送られたとき、その種類に対応する処理を行うようにすればよい。

例えば、NEXTSTEPではイベントは図1.3-4のような構造体で表現される。他のウィンドウシステムでも似たようなものである。

入力装置や他アプリケーションからの雑多な要求を、ウィンドウシステムで一度同じ形式のイベントに変換してからアプリケーションでまた分類するという方式は、問題点も多い。これについては、

```
typedef struct _NXEvent {
    int type;                 //イベントの種類
    NXPoint location;         //座標
    long time;                //イベント発生時間
    int flags;                //KB状態フラグ
    unsigned int window;      //ウィンドウ番号
    NXEventData data;         //イベント種別に依存するデータ
    DPSContext ctxt;          //DPSのコンテクスト
} NXEvent, *NXEventPtr;
```

図1.3-4　NEXTSTEPのイベント構造体

1.3.4(1)で詳しく述べる。

(iii)　ウィンドウマネージャ

アプリケーションのウィンドウ内で発生したイベントはそのアプリケーションが処理するが、アプリケーションのウィンドウの移動などの操作をする場合、誰がその面倒をみるかについては以下のようないろいろな方式が考えられる。

方式1：アプリケーションが自力で面倒をみる

方式2：システムが全イベントを管理する

方式3：ウィンドウに「枠」を付けてシステムは枠のみを管理する

方式1は、ウィンドウの移動や拡大／縮小などの各種の処理について、すべてアプリケーションが実行しなければならないのが大変であることに加え、アプリケーションごとにそれらのインタフェースが異なってしまう可能性があるという問題がある。方式2は、システムがユーザからの全イベントを捕捉して、ウィンドウ操作以外のイベントのみをアプリケーションに渡す方式である。方式3は、アプリケーションの各ウィンドウの周囲に枠をシステムが付加し、ユーザが枠の上でマウスを操作した場合にウィンドウ操作を行う方式である。現在は方式3が一般的であり、図1.3-2、図1.3-3のように特別なアプリケーション「ウィンドウマネージャ」を用意してウィンドウの操作を受け持たせることが多い。

(iv)　並列性の扱い

ウィンドウシステムでは、複数のアプリケーションが並行して動作する必要がある。UNIXのようなプリエンプティブなマルチタスクOSでは、各アプリケーションを別プロセスとして動作させることができるが、PC上のウィンドウシステムでは、各アプリケーションが自発的に別のプロセスへの切換えを指示する必要がある。

ウィンドウシステムのアプリケーションでは、一つのアプリケーション内の各部品も並行して動作

するとさらに便利である。例えば、図形エディタにおいてマウスで線を描きながらキーボードで線の太さを変化させるといったことができるとよい。部品を並行に動作させるためには、各部品にプロセスやスレッドを割り当てるか、擬似並行動作をさせればよい。

(d) ウィンドウシステムを使用したアプリケーションのプログラミング例

ウィンドウシステム上の単純なアプリケーションは、図1.3-5のような構造になる。

```
main()
{
    ウィンドウを作成;
    初期化;
    while(1){
        イベント取得;
        if(マウスイベント)
            アプリケーション処理;
        if(再描画イベント)
            画面を書き直す;
        システムに制御を戻す;  <-- プリエンプティブでないOSの場合
    }
}
```

図1.3-5 典型的なアプリケーションの構造

例えばX Window Systemで、ウィンドウを開いてマウスがクリックされるごとに、その位置にアスタリスク（*）を表示するプログラムは図1.3-6のようになる。

図1.3-6のようなプログラミングスタイルは、ウィンドウシステムに特有のものではない。例えば、文字端末を使用するスクリーンエディタviの基本構造は、図1.3-5と同じである。

単純なアプリケーションでは、このようなプログラミングスタイルでも問題ないが、複雑なインタラクションを行わなければならない場合は、イベントを条件により分類したり、イベントの発生位置により動作を変えたりする必要があるため、そのような処理を行う「イベントディスパッチャ」の条件判断や状態の数が増えて混乱が起こる。このような問題に対処するため、従来からプログラムの状態遷移を容易に記述するために、[Wasserman85]など各種のユーザインタフェース記述言語が提案されてきた。しかしウィンドウシステムでは、状態遷移記述を支援するツールよりも、見栄えやインタラクション手法全体をライブラリ化した「ツールキット」が広く使用されている。

(4) ツールキット

前述したように、ウィンドウシステムを使用する場合でもGUIのプログラミングには面倒な要素が

```
#include <X11/Xlib.h>

main()
{
    Display *disp;
    Window win;
    GC gc;
    XEvent e;
    Font font;

    disp = XOpenDisplay(NULL);
    font = XLoadFont(disp, "a14");
    win = XCreateSimpleWindow(disp,
        RootWindow(disp,0), 0, 0, 400, 400, 2,
        BlackPixel(disp, 0), WhitePixel(disp, 0));

    XSelectInput(disp, win, ButtonPressMask);
    XMapWindow(disp, win);
    gc = XCreateGC(disp, win, 0, 0);
    XSetFont(disp, gc, font);

    while(1){
        XNextEvent(disp, &e);
        if(e.type == ButtonPress){
            XDrawString(disp, win, gc,
                e.xbutton.x, e.xbutton.y, "*", 1);
        }
    }
}
```

図1.3-6　X Window Systemのプログラム例

多いため、各種のインタフェースツールキット（または単にツールキット）と呼ばれるライブラリ群がインタフェースプログラムの作成に広く使用されている。ツールキットとは、テキスト入力枠やボタンなどのグラフィックインタフェース部品の外見及び動作をライブラリとして定義したもので、インタフェース部がプログラムのメインルーチンとなって動作し、各部品に対しマウスやキーボードなどから操作を加えると、部品に対し定義された「コールバック」関数が呼び出されるようになっている。ツールキットでは、入力装置からの要求をシステムがとりまとめてツールキットに渡し、ツールキット側においてその要求がどの部品に対応するのか判断して、各部品にイベントを配送するという構造になっているものが多い。

図1.3-7にNEXTSTEPのツールキットAppkitを使用したプログラムの例を示す。図1.3-5のプログラムと異なり、プログラムのメインループを構成するイベントディスパッチャはツールキットライブラリ（この場合runメソッド）の中にあり、あらゆるメニューやツールを初期化してコールバック関数を設定した後でこれを呼び出す。

```
#import <appkit/appkit.h>

main()
{
    id  window, menu;
    NXRect rect = {{0,0}, {100,100}};

    [Application new];      //イベントディスパッチなどを管理する
                            //アプリケーションオブジェクトを作成し
                            //大域変数NXAppに格納

    window = [[Window alloc]  //ウィンドウ作成
        initContent:&rect
        style:NX_TITLEDSTYLE
        backing:NX_BUFFERED
        buttonMask:NX_MINIATURIZEBUTTONMASK
        defer:NO];

    menu = [[Menu alloc]        //メニューの作成
        initTitle:"Menu"];
    [menu addItem:"Hide"        //メニュー項目とコールバックの設定
        action:@selector(hide:)
        keyEquivalent:'h'];
    [menu addItem:"Quit"
        action:@selector(terminate:)
        keyEquivalent:'q'];
    [menu sizeToFit];           //メニューの大きさ決定
    [NXApp setMainMenu:menu]; //アプリケーションにメニューを登録

    [window center];            //ウィンドウを画面の中心に配置
    [window display];           //ウィンドウを表示
    [window orderFront:nil];  //最も前面に移動

    [NXApp run];                //ツールキットメインループ
                                //メニューで"Quit"を選択すると抜ける
    [NXApp free];               //ツールキットの後始末
}
```

図1.3-7　Appkitのプログラム例

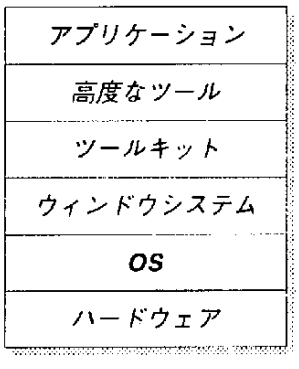


図1.3-8　ユーザインタフェースツールの階層（出典：[Myers94]）

現在、ほとんどのGUIアプリケーションにおいて何らかのツールキットが使用されているので、アプリケーションは図1.3-8のような階層構造で構築されていることが多い[Myers94]。

(5) インタフェースビルダ

図1.3-8において、ツールキットの上位に位置する「高度なツール」として、対話的にグラフィック設計を行ういわゆる「インタフェースビルダ」が近年広く用いられるようになっている。インタフェースビルダという名前は、NeXT社のワークステーションに登載されている「Interface Builder」のようなシステムという意味で一般に用いられており、ウィンドウシステムのツールキット部品を直接操作によりウィンドウ画面に配置しながらアプリケーションの外見を設計し、かつツールを操作したときのアプリケーションの動作記述を支援するシステムのことを指す。

インタフェースビルダでは、直接操作インタフェースによるグラフィック画面設計とテキスト編集によるプログラミングを同時進行させながらアプリケーションを作成する。ツールキットの使用が前提となっており、ボタンやスライダのようなインタフェース部品それぞれに対し、それらの部品が操作されたときに呼ばれるコールバックルーチンを対応付ける。コールバックルーチンの雛型やインタフェース部品の初期化のためなどのコードは自動的に生成され、コールバックルーチンをきちんと書かなくてもアプリケーションの見かけの動作をテストすることができる。またアプリケーション作成のプロジェクト全体を管理する機能を持つものもある。

インタフェースビルダは、最も成功したビジュアルプログラミングシステムの一つということができる。マウスなどでツール部品を画面に配置することはテキストで指定するのに比べ容易なので、画面の配置などはビジュアルに行い、プログラム本体はテキストエディタを使って書くという具合に、ビジュアルな部分とテキスト編集部分を相補的に使用することができる。またテスト機能によりラピッドプロトタイピングを支援しているし、統合化プログラミング環境としても優れたものが多い。

1.3.3 ウィンドウシステムとツールキットの製品例

Alto、Star、Smalltalkなどの研究的要素の強いウィンドウシステムに始まり、数々の商用ウィンドウシステムが提案されてきた。1994年現在、PCではMicrosoft WindowsとMacintoshのウィンドウシステム、ワークステーションではX11とNEXTSTEPが製品として普及している。本項では、商用で成功したいくつかのウィンドウシステムの特徴を解説する。

(1) SunView

SunViewは、Sun Mycrosystems社のUNIXワークステーションに標準で搭載されていたウィンドウシステムとツールキットである。(2)で述べるX Window Systemが普及する以前から製品化され、GUI

を活用したdbxデバッガのような優れたアプリケーションも搭載されていたため、Sun Mycrosystems社のワークステーションが広く使用されるようになった原動力の一つとなった。

SunViewツールキットはC言語のライブラリであるが、Cのプリプロセッサ、構造体と関数ポインタ、可変個引数などを最大限に活用してオブジェクト指向風の使いやすいライブラリが構築されていた。

SunViewでは図1.3-1に示したように、ウィンドウシステムとウィンドウマネージャがすべてOSの内部に構築されているため柔軟性に乏しく、またツールキットライブラリがすべてアプリケーションにリンクされる必要があったため、アプリケーションプログラムが巨大になるという欠点があった。

SunViewウィンドウシステムは、X Window Systemの普及に伴い標準として提供されなくなったが、ツールキットは現在もX上のXViewライブラリとして生き残っている。

(2) X Window System

X Window Systemは、UNIXワークステーション上で現在最も多く使用されているウィンドウシステムである。X Windowは、元々MITのマルチメディアプロジェクトProject AthenaにおいてRobert Scheiflerらにより開発されたものであり、パブリックドメインソフトウェアとして提供されたことにより全世界に急速に広まった。

Xのアーキテクチャにおいて最も画期的なことは、1.3.2(3)で述べたように、ウィンドウシステムにおけるウィンドウやハードウェアの管理部とインタフェースを扱うウィンドウマネージャを分離したことと、TCP/IP上のプロトコルを使用することにより、UNIXマシンに限らずネットワーク上の任意のマシンでアプリケーションやウィンドウマネージャを動かすことができるようにした点であろう。その他、無償で配付されたため大学など多くの機関で拡張が行われたこと、特定のハードウェアやOSに依存しない工夫がされていたこと、それ以前に広く使われていたウィンドウシステムが存在していなかったことなどの理由で普及が促進された。

Xは以上のような優れた特長を持ってはいたが、汎用を目指したために仕様が過度に複雑になっていたり、ライブラリXlibのグラフィクス関数がその他のどんなグラフィックライブラリとも整合性の悪い奇妙な仕様になっていたりという問題があったり、互換性のない各種のツールキットがあちこちから発表されて混乱を招いたりしていたが、広く普及したおかげで各種の優れた高度なツールが開発されることにより有効性が高まりつつある。

X上のインタフェースツールとして近年特に注目されているのは、John Ousterhoutの開発したTk/Tclシステム[Ousterhout]である。TclはUNIXのシェルに似た構文を採用したインタプリタ型コマンド言語であり、TkはTclから使用できるツールキットである。Tk/Tclの発表以前にもX上には各種のツールキットが存在し、Common Lispなどのインタプリタ言語でそれらを操作するためのライブラ

リも存在したが、Tkツールキットの出来が良かったことや、Common Lispよりもコマンド型インタプリタ言語の方が受け入れられやすかったこと、作者が積極的に宣伝活動を行ったことなどの理由により現在広まりつつある。Tk/Tclシステムを使用すると、Xの複雑な仕様を知らなくても簡単にGUIのプログラミングができること、インタフェースビルダが付属していること、インタプリタ言語でありラピッドプロトタイピングに向いているといった特長もある。

Tk/Tclシステムの問題点は、Tcl言語の機能が低いことである。Tcl言語は一行ごとの処理を基本としており、改行文字が特別の意味を持つためフリーフォーマットの記述ができないし、Cでは"i++"と書くようなちょっとした計算や代入を行う場合でも、"set i [expr $i+1]"のような不自然な記述を行わなければならない。このため、SchemeからTkを使用するSTKのように、Tclを使わずにTkを使う工夫も行われている。

TclとTkのプログラムの例を図1.3-9に示す。counterというラベルとstart、stopという二つのボタンを作成してストップウォッチを実現している。これらのラベルやボタンは自動的に配置される。

```
label .counter -text 0.00 -relief raised -width 10
button .start -text Start -command "set stop 0; tick"
button .stop -text Stop -command {set stop 1}
pack append . .counter {bot fill} .start {left expand fill}\
        .stop {right expand fill}

set seconds 0
set hundredths 0
set stop 0

proc tick {}{
    global seconds hundredths stop
    if $stop return
    after 20 tick
    set hundredths [expr $hundredths+2]
    if {$hundredths >= 100}{
        set hundredths 0
        set seconds [expr $seconds+1]
    }
    .counter config -text [format "%d.%2d" $seconds $hundredths]
}

bind . <Control-c> {destroy .}
bind . <Control-q> {destroy .}
focus .
```

図1.3-9　Tcl/Tkのプログラム例

(3)　NeWS

Sun Microsystems社はSunViewの後継ウィンドウシステムとして、James Goslingの開発した拡張PostScriptに基づくNeWSウィンドウシステムを開発した。NeWSでは、描画などすべての入出力やウィンドウ操作にPostScriptが使用され、またPostScriptによるツールキットntk（NeWS ToolKit）も用意されていた。

NeWSは先進的なウィンドウシステムであったが、発表時、既にX Window Systemが実質上の標準として普及していたことや、PostScriptを使用するメリットが大きくなかったり、Cプログラミングとの整合性が悪かったりしたためか、普及しないままに終ってしまった。

(4) NEXTSTEP

NeXT社は1988年、自社製のUNIX（MACH）ワークステーションで動作するウィンドウシステムNEXTSTEPと、その上のツールキットAppkit及びインタフェースビルダInterface Builderを発表した。1994年現在、NeXT社はワークステーションハードウェアの生産を中止しているが、上述のソフトウェアを継続して販売している。NEXTSTEPはMACHの機能及びDisplay PostScript（DPS）、Objective-Cの長所を統合した先進的なウィンドウシステムである。またInterface Builderは最初に開発されたインタフェースビルダであり、数度の機能拡張を経た現在においても、最も優れたインタフェースビルダの一つである。Appkitは、ウィンドウや各種のインタフェースツールがすべて階層的なオブジェクトとして表現されたオブジェクト指向型ツールキットである。

Display PostScriptは、Adobe社がPostScriptを拡張したもので、NeWSの拡張PostScriptと同様にPostScriptをウィンドウシステムでも使用できるようにしたものであるが、C言語と組み合わせて使用することについてよく考慮されており、Appkitの描画ライブラリ／イベント操作ライブラリとして活用されている。

上述のシステムやツールは完成度が高く、これらを使用するとGUIアプリケーションを容易に構築することが可能であるが、NeXT社のOSでしか使用することができないため、これまでにあまり普及が進んでおらず、またPOSIX、X11、Open GLといった標準化の主流からはずれているので、今後の発展は不透明なようである。

(5) シリコングラフィクス社のウィンドウシステム

グラフィックワークステーションにおいて、現在圧倒的なシェアを誇るシリコングラフィクス社（SGI）は、高速グラフィクスハードウェアを備えたワークステーション及び独自のウィンドウシステム上の3次元グラフィックライブラリGraphic Library（GL）により高い評価を得てきた。GLは高度な3次元グラフィクスツールのための基本ライブラリとして有用であるが、GLを直接を使っても比較的簡単にウィンドウシステムのプログラミングを行うことができることが特徴である。例えばウィンドウを開いて矩形を描くプログラムは、図1.3-10に示すように書くことができる。

GLは3次元描画ライブラリの実質的標準となるに伴って、さらなる標準化が行われている。独自のウィンドウシステムは現在はX11とマージされており、通常のX11のアプリケーションもGLのアプリケーションも動作するようになっている。またGL自体もOpen GLとして標準化されており、Open GL

が使えるSGI社以外のシステムが増えつつある。

GLは3次元グラフィクスの表現に非常に強力であるが、2次元描画能力はDisplay PostScriptに及ばず、またインタフェースビルダのようなツールも標準的に提供されていないため、GUIアプリケーション作成の効率はNeXTに及ばないようである。しかし、SGIのシステムはPOSIXやOpen GLのような今後普及するであろう標準に準拠している上に、Display PostScriptも使用することができるので、GUIアプリケーション開発環境の整備に期待したいところである。

```
#include <gl/gl.h>

main()
{
    int win;
    win = winopen("GL Example");
    color(RED);
    clear();
    color(YELLOW);
    rectf(10, 10, 50, 50);
    sleep(10);
}
```

図1.3-10　GLのプログラム例

(6)　PC上のウィンドウシステム

PCで普及しているウィンドウシステムは、1.3.2(3)で述べたように、アプリケーションにおいてプロセス切換えを明示的に指示しなければならなかったり、開発環境が不十分だったりするため、アプリケーションの開発は困難を伴うことが多かったが、開発環境や言語の充実により状況が変わりつつある。

近年特に注目される開発環境／言語にVisual Basicがある。Visual Basicは、Microsoft Windows用のプログラミング言語／アプリケーション開発環境である。Basicと名うってはいるが、従来のパソコンBASICとは似ても似つかぬ構造的インタプリタ言語である。Visual Basicは特定のツールキットに対応して後から開発されたインタフェースビルダではなく、最初から統合的プログラミング環境として意図されたシステムである。ボタンなどの部品は、ウィンドウの上でClick & Dragによる直接操作により作成／配置し、属性をメニューで設定することができる。Visual Basicはプログラミング言語としては洗練されたものではないが、インタプリタ言語であることからインタフェースの作成／テストのサイクルを迅速に行うことができるし、OLE、DDEのような機能を使ってExcelやWordのような既存のアプリケーションと組み合わせて使うこともできるし、統合的プログラミング環境としてよくできているため、今後着実にユーザ数が増大すると予想される。

1.3.4 ウィンドウシステムの研究動向

Macintosh、Microsoft Windows、X Window Systemなどのウィンドウシステム及びそのアプリケーションが世間に広まって数多く使われているため、1.3.2(3)で述べたような各種の仕様の選択に関しては現在は議論の対象とはなっていない。ウィンドウシステム自体のアーキテクチャを大きく変更する必然性は少なく、実装手法は既に研究者の興味の対象とはなっていない。

複雑なアプリケーションのプログラミングは容易ではないが、ツールキットやインタフェースビルダのような高度なツールが普及してきたため以前よりは状況が良くなっており、また多少苦労すれば一応誰でもアプリケーションを作成できるため、「ウィンドウシステムのプログラミングはそういうものである」といった認識が広まっている。また、既存のシステムに慣れたプログラマが多数存在するので、わざわざ新たに特別なシステムを導入するほどのことはないと思われているようである。

しかし、プログラミングにおける問題点は多くのプログラマの間で共通に認識されてきているため、既存の枠組から大きくはずれずに問題点をうまく回避するための各種のしかけが研究されてきている。コンパイラの作成は昔は困難であったが、構文／字句解析ツールの普及により開発が非常に楽になったのと同様に、ウィンドウシステム上のユーザインタフェースの開発も各種のツールの開発により徐々に容易になりつつある。

(1) ユーザインタフェースソフトウェア構築上の問題点

近年のインタラクティブシステムにおいては、ソフトウェア全体のうちユーザインタフェース部の占める割合は、50%から80%にものぼることが普通である[Myers92b]。ユーザインタフェースのソフトウェアは以下のような点において作成が困難であると言われている[Myers92b]。

(a) 対話的設計

インタフェースは実際に使用し評価してみなければ善し悪しが分からないことが多いため、よいインタフェースを作成するためには何度も繰返し改良を行うことが必要である。このためには、プログラムの修正とテストを数多く繰り返す必要があるので、システムの開発に手間と時間がかかってしまう。

(b) グラフィック設計

インタフェースを使いやすくするためには、ソフトウェアの見栄えや部品の配置が重要な意味を持つ。人手で試行錯誤的に配置を決めるのは手間がかかるし、自動的に見栄えのよい配置を行うことは難しい。

(c)　非同期入力のサポート

システムがどのような状態で動作していても、ユーザがシステムを制御する要求を出す可能性があるので、システムは常にユーザの非同期入力に注意している必要がある。

(d)　並列処理

マウスとキーボードのように独立した複数の入出力装置を扱う場合は、それぞれに別々のプロセスを割り当てる必要があるし、テキスト入力枠やボタンなどのインタフェース部品が並んだグラフィックインタフェース画面においては、ユーザが任意の順番で部品を操作可能にするためには、それぞれのインタフェース部品を並列に動作させなければならない。またユーザをアプリケーションと独立に動くプロセスと考えると都合が良い場合もあるし、アプリケーション部とユーザインタフェース部の分離も有効である。このように高度なユーザインタフェースソフトウェアでは並行処理機能が必須であるが、並列処理を扱うプログラミングはそうでないものに比べ複雑になりがちである。

(e)　効率

使いやすいシステムは、常にユーザの操作に高速に反応する必要があるので、入力に対する応答時間や描画などの出力に要する時間が小さくなるように、常に効率的な実装が要求される。

(f)　エラー処理

人間の操作には間違いがつきものであるが、ユーザがどのような間違った操作を行った場合でもどのような間違いが起こったのかをユーザに知らせたり、正しい状態に回復したりする機能が必要である。ユーザのエラーに対するきめ細かい処理を作成することは、機械的な処理に対するエラー処理に比べはるかに複雑である。

(g)　例外処理、undo 機能

ユーザが間違った操作を繰り返した場合でも、元の状態に復帰できるようにするため、すべての操作履歴を記憶しておいたり、すべての操作に対して逆操作 (undo) が可能なようにするなどの対応が必要である。

(2)　プログラミング言語による支援

(1)で述べたようなユーザインタフェースソフトウェアの作成にまつわる問題点は、アプリケーション作成に使用される汎用言語がユーザインタフェース作成に向いていないことに起因するものが多い。

1．視覚的プログラミング

ユーザインタフェースの作成においては各種の機能が言語に要求されるが、あらゆる機能をすべて一つの言語に盛り込むことは不可能だし有効なことではない。このことから、上のような問題点は、以下のような方式で解決するのがよいと思われる[Masui92]。

・問題に向いた各種言語を併用する。例えば状態遷移の記述には状態遷移記述言語を用い、インタラクションのプロトタイプ作成にはインタプリタ言語を使う。
・言語に共通な並列動作プリミティブを用意し、各種言語で記述されたモジュールを互いに通信させる。

(3) プログラミング環境の支援

(2)で述べたように、ユーザインタフェース作成に適したプログラミング言語を使用することでGUI 作成に関する問題点はある程度解決するが、ユーザインタフェースを表現するにはテキストベースのプログラミング言語では不十分な場合が多い。1.3.2(5)で述べたインタフェースビルダは配置をビジュアルに行いつつ、テキストベースのプログラミングを行う統合プログラミング環境として有用であるが、すべてのプログラミングをビジュアルに行おうとするビジュアルプログラミングシステム（1.2節参照） や、操作手順のようにテキストで表現しにくいものについて、例示と汎化によりプログラムを作成しようとする例示プログラミングシステム（1.4節参照）も有用である。

(4) 配置の支援

GUIの設計においては、ウィンドウや部品を使いやすいよう配置するのに手間がかかる。またツールや表示対象の配置をしばしば動的に決める必要があるため、配置支援システムが重要である。

1.3.2(5)で述べたインタフェースビルダは、インタフェースツールの静的な配置の設計には有用であるが、動的な配置への対応は十分ではない。動的に変化する部品やデータを有効に配置／表示するための研究が行われている。

(a) 制約解決システム

GUI部品などの配置は、単純な幾何制約で表現できることが多い。例えば、「3個の文字入力枠を等間隔で左揃えで並べる」とか「ボタンをウィンドウの中心に配置する」といった具合である。このような制約のみを予めシステムに与えておけば、自動的に制約を満たす値を計算してくれるものが制約解決システムである。例えば、システムに「ボタンの位置 = ウィンドウの左端位置 + ウィンドウの幅 / 2」といった制約を与えておけば、ウィンドウの位置や大きさを変更したときボタンの位置が自動的に計算され配置される。

(b) 自動配置システム

データ構造を自動的に2次元画面に配置する手法については古くから数多くの研究がなされており、特に木構造や有向グラフの配置に関して多くの研究が行われている。従来の自動配置システムのほとんどは、ある審美的基準に基づいて手続き的アルゴリズムにより配置を求めるものであったが、近年は遺伝的アルゴリズムやシミュレーティッドアニーリングを用いた自動配置手法がいろいろ提案されている。

(5) 新しいインタフェース部品

一般のツールキットでは、標準的な限られたインタラクション手法しかサポートされていない。スライダやメニューなどのインタフェース部品は各種のツールキットにおいて標準的にサポートされているが、目的の処理を実行するために最善の部品というわけではなく改良の余地を含んでいる。メリーランド大学のShneidermanらは既存のツールキット部品に様々な改良を行ったものを多数提案している。

(a) 微調整スライダAlphaSlider

GUIにおいて連続的な値を設定したり項目を選択したりする場合、スライダがよく使用されるが、普通のツールキットのスライダでは設定する値を微調整したり大量のデータから一つの項目を選択することは難しい。Alphaslider[Ahlberg94]は、スライダのノブを粗調整部と微調整部に分けることにより、細かな値の調整ができるようにしたものである。

(b) SplitMenu

GUIにおいてリストの中から項目を選択するためにプルダウンメニューがよく使用されるが、多数の項目の中で実際によく使用するものは限られていることが多い。SplitMenu[Sears94]は、頻繁に選択した項目をメニューの最上部に自動的に移動させることにより、よく使われる項目を選択しやすくしたものである。

このような新しい部品を含むツールキットが1.3.4(8)で紹介するGalaxyツールキットに実装され、Maryland Toolkitとして販売されている。

(6) ウィンドウシステムの拡大

ウィンドウシステムは、複数の重なり合うウィンドウを使うことにより多数の仕事を同時に行うための枠組であるが、必要なウィンドウの種類は仕事の種類によって異なるし、ウィンドウの大きさや

配置も仕事によって変えた方がよいかもしれない。普通のウィンドウシステムでは、別の種類の仕事に移る場合は不要なウィンドウをすべて消したり、必要なウィンドウを新たに開いたりする操作が必要であるが、D. Hendersonらの提案するRoomsシステム[Henderson86]では、各々の仕事に対して「部屋」を用意して仕事に最適なアプリケーションとウィンドウを配置し、部屋を移動することにより環境を切り替えることができる。例えば、「メールの部屋」では文書編集ウィンドウが大きな領域を占めているが、「プログラミングの部屋」ではデバッガなどが大きな領域を占めているという具合になる。

(7) CSCWへの拡張

通常のウィンドウシステムは一人で使用することを前提としているが、CSCWに使用するためには同じウィンドウやデータを複数の人間で共有する枠組が必要である。一つのウィンドウを複数のディスプレイに表示するための最も簡単な方法は、アプリケーションから同じ描画要求をネットワーク上の複数のディスプレイに送出することであるが、描画要求は一般に量が多いので多数のデータを転送しなければならないことが問題になる。これを避けるため、複数のマシンで同じアプリケーションを動作させ、これらすべてに同じ操作要求を送ることにより同じ画面が得られることを期待するという方式も提案されているが、異なるマシン上で同じ状態や画面が得られる保証がないという問題がある。データを共有する複数のアプリケーションが、独自に入力を扱ったり描画を行いながら協調して一つのCSCWシステムを構成する方式がより望ましいと考えられるので、これを可能にするための[増井93]のような基本的枠組の確立が望まれる。

(8) プラットフォーム独立のツールキット

各種のツールキットの違いを隠し、異なる機械やOS上で同じアプリケーションを動作させるための仮想ツールキット（Virtual Toolkit）が近年各種発表されている。仮想ツールキットは、図1.3-11の

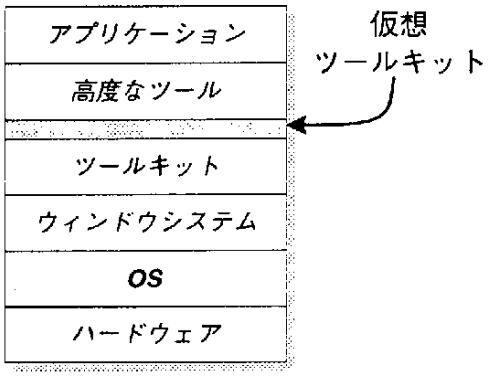


図1.3-11 仮想ツールキット

ように既存のツールキットの上位に位置してツールキットごとの違いを吸収するものである。

XVT CorporationのXVTなど多くの仮想ツールキットは既存のツールキットを使用しているが、Visix SoftwareのGalaxyのように、Mac／X／PC上に独自のツールキットを構築しているものもある。

【参考文献】

[Ahlberg94] Ahlberg, C. and Shneiderman, B. AlphaSlider: A Compact and Rapid Selector. in: Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'94). Addison-Wesley, 1994, pp. 365-371.

[Henderson86] Henderson, D. A. and Card, S. K. Rooms: the use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. ACM Transactions on Graphics, vol. 5 (1986), pp. 211-243.

[Masui92] Masui, T. User Interface Programming with Cooperative Processes. in: Languages for Developing User Interfaces, by T. Masui, edited by B. A. Myers. Jones and Bartlett, Boston, MA, 1992, pp. 261-277.

[増井93] 増井俊之, 花田恵太郎, 音川英之. 共有空間通信を利用したグループワークシステムの構築. 情報処理学会プログラミング-言語・基礎・実践-研究会研究報告92-PRG-10, vol. 93 (1993), pp. 49-56.

[Myers92a] Languages for Developing User Interfaces. edited by B. A. Myers. Jones and Bartlett, Boston, MA, 1992.

[Myers94] Myers, B. A. and Olsen, D. R. User Interface Tools (CHI'94 Tutorial). 1994.

[Myers92b] Myers, B. A. and Rosson, M. B. Survey on User Interface Programming. in: Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'92). Addison-Wesley, 1992, pp. 195-202.

[Ousterhout] Ousterhout, J. Tcl and the Tk Toolkit. Addison-Wesley.

[Sears94] Sears, A. and Shneiderman, B. Split Menus: Effectively Using Selection Frequency to Organize Menus. ACM Transactions on Computer-Human Interaction, vol. 1 (1994), pp. 27-51.

[Shneiderman83] Shneiderman, B. Direct Manipulation: A Step Beyound Programming Languages. IEEE Computer, vol. 16 (1983), pp. 57-69.

[Wasserman85] Wasserman, A. I. Extending State Transition Diagrams for the Specification of Human Computer Interaction. IEEE Transactions on Software Engineering, vol. 11 (August 1985), pp. 699-713.

1.4 例示によるGUIプログラミング

1.4.1 はじめに：例示によるGUIプログラミングの目的

GUIの構築が従来のインタフェースの構築と比較して非常に労力を要することは、久しく指摘されてきている。この問題を解決するためにUser Interface Management Systems（UIMS）や、widgetsの開発が盛んに行われているが、それと同時に従来のテキストベースのプログラミングとは全く異なったプログラミングスタイルが模索されてきている。その中の一つの大きな流れは、「例示によるプログラミング」の応用である「例示によるGUIプログラミング」である。これらのシステム（PBE-GUIシステム）は、GUI構築のためのプログラミング支援と、エンドユーザの処理の自動化のためのマクロ（又は、シェル）プログラムの生成に大きく分けられる。以下にまず、なぜ例示によるGUIプログラミングの研究が注目されているのかを述べる。

従来のテキストベースのプログラミング言語を用いてGUIの構築を行おうとしたときには、以下のようなGUIの特質がプログラミングを非常に繁雑にしている。

[問題1] 多くのGUIはイベント駆動なので処理の流れが分かりづらく、関数呼出しなどの関係が複雑になりがちである。

[問題2] GUIではグラフィカルな属性（例えばフォントのサイズ）などが重要になってくるがそれらは具体的な絵を見ないと分かりづらい。

widgetは[問題1]を解決するために、基本的なGUIの部品を予めパラメータ化して用意している。プログラマは部品の細かい挙動をプログラミングする必要がなくなり、プログラミング効率の向上が期待される。また多くのUIMSが備えているプロトタイピングシステムは、視覚的にwidgetの配置などをするためのツールで、[問題2]をある程度解決している。しかし、これらの手法はともに非常に定型的なプログラミングの範囲での支援しか提供せず、プログラミングの最も重要な部分、つまりアプリケーションに特化したGUIの挙動や外見に関する要求仕様から、そのようなGUIを実現するような抽象化を伴うプログラミングへの変換に対するサポートが欠けている。視覚的プログラミング言語を用いる手法も、いわゆるインタフェースビルダのようなwidgetの配置や、簡単なイベントの配送の指定をする以外は、必ずしも通常のプログラム言語と比較して有利であるとはいえない。つまり、生成されるインタフェースとは必ずしも関係がない視覚表現が導入されるため、インタフェース・デザイナにはやはりプログラミングの素養が要求されてしまう。

「例示によるプログラミングProgramming by Example（以下PBE)」は、この「アプリケーションに特化したGUIの視覚的挙動の抽象化」という部分の支援を目指したものである。つまり究極的にはプログラマはGUIの完成予想図を描き、それの挙動をデモンストレーションするだけで、システムが自動的にそのようなGUIを生成してくれるわけである。PBEを適用すると上記の二つの問題点は以下のようにして解決される。

[解決策1] プログラマはエンドユーザの立場からGUIの振舞いをデモンストレーションするだけでよいので、デモで示されたような挙動を実現するための複雑なプログラミングから開放される。

[解決策2] プログラマはGUIの外見を描画エディタを用いて直接指定することができるので、グラフィカルな属性の指定は容易である。

ここまで述べてきたのは、PBEによるGUI構築のプログラミングの支援であるが、一方PBEを利用してGUIのエンドユーザを支援しようという研究も多くなされている。GUIは初心者に分かりやすいインタフェースとして広く使われるようになったが、その最大の理由は、「具体性」にある。つまり、コンピュータ上の抽象的な概念、例えばファイル、ディレクトリなどをアイコンなどのすべて目に見える形で表現し、ポインティングデバイスを用いてそれらのアイコンなどを直接操作できるようにしたので、多くの人に受け入れられてきた。しかし、ひとたび「自動化」というコンピュータの最大の武器を利用しようとし始めると、この「具体性」が大きな障害となる。つまり、「自動化」=「マクロ（又は、シェル）のプログラミング」によって処理を行うためには、処理に対する汎用的な記述が必要になる。従来のテキストベースのインタフェースの上では、すべての抽象的なものに対して文字列で名前を付けていたために、テキストベースのプログラミングとの相性が非常によかった。しかし、GUI上での処理をプログラミングしようとすると、GUIの抽象性の欠如のため従来のテキストベースの記述では扱うことが非常に困難になる。

このように、エンドユーザの視点から、GUIと相性のよいマクロのプログラミングスタイルを模索した結果、たどり着いたのがPBEである。PBEを用いたGUIプログラミングでは、ユーザはテキストベースの記述について考える必要はなく、通常のGUI上での操作をデモンストレーションするだけでよい。結果として、GUIの具体的な世界とマクロのプログラムの抽象世界との間の橋渡しは、プログラマではなくPBEシステムが行うことになる。

1.4.2 PBE-GUIシステムの研究課題

技術的な観点からすると、PBE-GUIシステムを特徴付ける大きな部分は、以下の二つの点である。

① ユーザの直接操作による指示を、インタフェース上のオブジェクトやアプリケーションの内部の状態変更に結び付けるための（帰納）推論エンジン

② それらの状態変更を、一般的なデータを扱えるプログラムにする汎化（Generalization）のための汎化推論エンジン

これらは、完全に独立した推論ではなく、例えば汎化を行う際に内部状態との対応関係の推論に影響が及ぶことも考えられる。実際、従来のPBE-GUIシステムは、これらが渾然一体となっているシステムもある。しかし、直接操作をより抽象度の高い操作にまとめる場合や、直接操作をどのようなア

プリケーションのデータに対する抽象操作に対応させるか、などの場合は、汎化とは関係なく、周りの状況から①の推論のみを行う場合もある。例えば、ユーザが直線の端点をつかみ、別な直線の端点の近傍に置いた場合、それが直線の単なる変更を意味しているか、あるいは直線どうしの連接を意味しているのか、といった場合である。あるいは、複数の図形オブジェクトを描画したとき、その間に成り立つ幾何制約を推論するPeridotやTRIP3のようなシステムもある。この場合、複数の幾何制約が複数のオブジェクト間で成り立つ場合、どれがユーザが本質的に意図したものかを適切に帰納的に推論する必要がある。

また、PBEシステムが汎化を行うか行わないかで、生成されたプログラムの一般的な適用性に本質的な差が生まれる。汎化を行わない場合、ユーザが明示的にPBEの起動を指示し、かつPBEによって得られたプログラムは、単純な、繰り返しや条件分岐がなく、起動時にはユーザが明示的に引数を指定するような、いわば従来のマクロが生成される。これに対し、汎化を行うことによって、ユーザ操作の繰り返しを自動的に検出したり、定数のパラメタ化などにより、より一般的なプログラムの生成が可能となる。Myersは、[Myers88]で、これらをProgramming-by-exampleのシステムと、Programming-with-exampleのシステムとの二つに分類しているが、近年では一緒に論ずることが多い。

エンドユーザ向けのシステムでは、ユーザのアクション列に対して汎化が可能になるとき、ユーザにそれを知らせて、かつ実際に汎化を行ってよいか否か確認するものがある。さらに一般にPBE-GUIシステムは、ユーザに対して推論や汎化の可能性が多様で、結果が曖昧になるとき、システムがユーザに対してどの推論結果を選択すべきなのか質問するシステムが多い。これらの場合、①どのようなタイミングでユーザに問いを出すのか、②どのように選択肢を分かりやすく見せるか、及び③どのように選択肢を限定するか、などが研究の大きな課題である。古くは、Peridotでは自然言語の質問のテンプレートを用意し、可能性を一つづつ確かめていたが、最近のシステムではダイアログボックスで複数の候補から選ぶシステムが見られる。これは、前者のエンドユーザ向けのシステムも同様で、汎化が可能なときにアイコン表示が変化したり、ダイアログが表示されたり、あるいはMondrianのように音声によってユーザに通知したりする、などの試みが行われている。さらに、最近では複数の推論や汎化の可能性を、実際のアプリケーションに対する具体的な操作の例示として表すシステムも登場している。これらは、実際の例を少量の説明で補佐することによって、ユーザが分かりやすい形で選択肢を提示できる手法として注目されている。さらに、IMAGEシステムのようにこれらの具体例示に変更を施すことによって、より正確にシステムの意図を伝達しようとするシステムも試みられている。

さらに推論系に関して、よく試みられているアプローチとしては、GUIに関する領域特有の知識やヒューリスティクスを推論系に導入することである。例えば、ある図形オブジェクト全体の明度を大幅に変化させた場合は、そのオブジェクトを選択したという例示だとみなす、といった具合である。

これを突き進めるとMaquiseのようなシステムになり、予め領域に特化して作られた図形オブジェクトをパラメータ化するのが例示の主眼となる。この場合、例示数が少なくても、有効な推論が可能となるが、逆に汎用性が失われ、ユーザが領域知識外の例示を示すことが難しくなる。一方、それに対するアプローチとして、汎用性のある推論エンジンを用いて、それを領域知識でパラメータ化するようなInference Bear、IMAGE、ProDeGE+のようなシステムも開発されている。これらは、基本的には推論や汎化が特定領域によらないので、適用範囲が広くなり、拡張も容易になる反面、自動的に推論できるプログラムの範囲が減少し、例示数の必要性が多くなって、ユーザの負担は増える傾向にある。

もう一つの大きな課題は、例を示すインタラクション時に、どのようにシステムが推論のために獲得する情報を増やせるか、といった問題を、クラシックな解決法、つまり、同じインタラクションからより多くの情報を得ようとする、いわば人工知能的な手法ではなく、むしろユーザとのインタラクションの中で、より獲得しやすいGUIインタラクションの技法を開発しよう、といった動きである。より具体的に述べると、後の[Myers88]や[萩谷91]でサーベイされたシステムに比較すると、最近のシステムは単に汎化機能をより強力にするだけではなく、むしろシステムが限定された情報から推論するということの限界を見極め、むしろ推論を正確に行うための情報を増やすような工夫の研究が多い。「情報を増やす」というのは、ユーザがGUIシステムとのインタラクションを行う際に、「自然な形」、つまり、極力ユーザの通常のインタラクションの行為を乱さない形で、推論エンジンがより多くの情報を得るようにすることである。これらは、①先にも挙げたユーザに対する問合せの工夫による情報獲得と、②直接操作においてはユーザ操作のコンテクストをユーザに自然に指定させることによって、より多くの情報を得るなどの手法が開発されている。例えば、②に関しては、Chimera、Mondrianなどのシステムで、適用可能なスクリプトの起動条件に他のオブジェクトをcontextualな情報としてスクリプト内に残し、そのスクリプトが起動されるための状況の検出に役立っている。

1.4.3 例示によるGUIプログラミングの最近の各研究の分類・総括

初期のPBE-GUIシステムの研究例として、Rehearsl WorldやPeridotなどがある[Myers90]。これらは[萩谷91]のサーベイに詳しいので、そちらを参照されたい。また、[Cypher93]は、最近のPBE-GUIの研究を数多く集めた論文集であり、この分野の必読書といえ、本節で紹介する幾つかの研究も収録されている。

本項では、最近の研究に焦点を絞ることにする。まず、PBEの利用に関する最近の研究について総観し、共通する評価項目を列挙する（表1.4-1参照）。

対象ユーザ： 先の項でも述べたとおり、ここに収録した研究は大きく分けてGUIを構築するシステムプログラマの支援、GUI利用者がGUI上での処理をプログラミングする際の支援の二つに分類される。

表1.4-1　例示によるGUIプログラミングシステムの機能一覧

	対象ユーザ	推論					結果確認	プログラム変更
		変数化	条件分岐	繰返し汎化	直接対話	幾何制約		
DemoII	GUI構築	ユーザ指示	自動推論	N/A	自動推論	自動推論	テキスト	追加例示
Marquise	GUI構築	ユーザ指示	N/A	N/A	自動推論	自動推論	テキスト	直接操作+テキスト
Layouts by Example	GUI構築	ユーザ指示	N/A	自動判断	N/A	自動推論	例示	追加例示
Multiple Snapshots	GUI構築	ユーザ指示	N/A	N/A	制約生成	自動推論	例示	追加例示
TRIP3	GUI構築	自動推論	自動推論	ヒント	制約生成	自動推論	テキスト	テキスト
IMAGE	GUI構築	自動推論	自動推論	ヒント	制約生成	自動推論	例示	追加例示
Inference Bear	GUI構築	自動推論	N/A	N/A	制約生成	自動推論	テキスト	テキスト+追加例示
MIKE	エンドユーザ	ユーザ指示	ユーザ指示	ユーザ指示	マクロ生成	N/A	テキスト	テキスト+追加例示
SmallStar	エンドユーザ	ユーザ指示	ユーザ指示	ユーザ指示	マクロ生成	N/A	テキスト	テキスト
AIDE	エンドユーザ	ユーザ指示	N/A	自動推論	マクロ生成	N/A	テキスト	追加例示
Eager	エンドユーザ	指示の先取	指示の先取	指示の先取	マクロ生成	N/A	実行のみ	追加例示
Pursuit	エンドユーザ	ユーザ指示	ダイアログ確認	ダイアログ確認	マクロ生成	N/A	グラフィカル	グラフィカル
Chimera	エンドユーザ	ユーザ指示	N/A	N/A	マクロ生成	自動推論	グラフィカル	グラフィカル
Mondrian	エンドユーザ	ユーザ指示	N/A	N/A	マクロ生成	自動推論	グラフィカル	グラフィカル
Metamouse	エンドユーザ	テキスト確認	テキスト確認	テキスト確認	マクロ生成	グラフィカル確認	実行のみ	追加例示
ProDe-GE+	エンドユーザ	テキスト確認	ユーザ指示	ユーザ指示	マクロ生成	テキスト確認	グラフィカル	グラフィカル
GA Graph Layout	エンドユーザ	ユーザ指示	N/A	N/A	N/A	自動推論	テキスト	追加例示

推論：　PBEの実用性はその推論メカニズムに依存する。表1.4-1では以下の5項目についてまとめた。

[変数化]　例の中の具体的な要素の中でどの部分を抽象化するか。

[条件分岐]　実用的なプログラムには条件分岐が欠かせない。

[数の汎化]　一種の条件分岐だが、数の繰り返しはプログラムの基本なので別項目とした。

[直接対話]　GUI構築の支援の場合、ユーザとの直接対話の挙動をプログラミングする。

[幾何制約]　画面上に配置したオブジェクトの間の位置関係に関する人間の意図を読みとる。

プログラムの表現形式：　システムが生成したプログラムをどのように人間に示すか。従来はテキストベースの記述が多く用いられてきたが、GUIを扱うためのシステムということで、グラフィカルに表示したり、挙動を具体的な例を用いて人間に示す試みも行われてきている。

プログラムの変更：　システムが生成したプログラムは、必ずしも人間の意図したとおりのものであるとは限らない。その場合に、どのようにしてプログラムを変更するのかという点で三つの選択肢がある。

① テキストベースの記述をそのまま扱うのか。

② グラフィカルに表現されたプログラムを直接操作を用いて編集するのか。

③ 複数の例を与えることでシステムに意図を間接的に伝えるのか。

これによって、全体的に以下のような傾向が見られることが分かる。

(a)　システムによる自動推論とユーザによる直接指示

すべてのシステムで、列挙したすべての項目でシステムが推論によって完全自動でGUIプログラムを生成するのではなく、何らかのユーザからの直接的な指示も入っているのが普通である。これは、通常のGUIのインタラクションでは、システムが与えられた情報からプログラムを生成するには推論能力が不十分であることに起因している。

(b) 幾何制約の推論の有無

GUI構築のシステムでは、widgetやその他のインタフェース用の図形オブジェクトのレイアウトや動的挙動を制御するため、何らかの幾何制約推論を行っている。一方、エンドユーザのマクロプログラミングのシステムやユーザのシェルプログラミングを支援するシステムでは行っていない。しかし、図形エディタのマクロ生成にPBEを用いているシステムでは、当然のことながら、図形オブジェクトに対する編集操作の推論の一部として幾何推論を行っている。ここで、シェルプログラミングで、今後幾何推論が必要になるか否かは検討を要する。

(c) 条件分岐や繰り返しの推論の有無

多くのエンドユーザ向けのマクロ生成システムでは、ユーザの繰り返し操作を自動的、あるいは半自動的に検出し、ループを生成するという汎化推論を行っている。GUI構築では、様々なアプリケーションのデータや、ユーザとのインタラクションに対応するための条件分岐の推論をサポートしているシステムは多いが、ループはそれほどでもない。これは、生成されるプログラムの種類に起因していると思われる。

(d) プログラム表現法・表現法の進化

過去のPBE-GUIシステムでは、ユーザが意図したプログラムが生成されているか否かを判断するには、テキストベースで表示されたものをユーザが解読するか、あるいは単なる内部表現のみが存在し、ユーザが実際に実行してみるしかなかった。しかし、前者では直接操作でインタラクションを行うという意図に反することになり、後者では判断が困難である。最近の研究では、この問題の解決法は、二つの傾向がある。

① 結果のプログラムの実行結果を例示するというもの。

② 視覚的言語を準備し、グラフィカルな結果表示や直接操作によるプログラム修正を可能にするもので、エンドユーザ向けのシステムに見受けられる。

(e) プログラム修正法の進化

プログラム修正法としては、①追加の例示を与える方法と、②先の視覚言語の直接操作による編集

があり、それぞれ一長一短である。前者は、インタフェースとしての簡潔性・一貫性があるが、ユーザが「正しい」追加例示を必ずしも与えられるわけではなく、汎化推論エンジンの限界により、どんな追加例示を行っても意図した修正が行えない可能性がある。また、下手に間違った追加例示を与えたりすると、その効果の取り消しが厄介である。一方、後者は実質的には直接的なプログラミングなので、誤りの余地は少なく、その言語で表現可能ないかなる修正も施せるが、有効な言語設計自身が難しく、冒頭に述べたように、テキストのプログラミングと実質的に変わらなくなってしまう恐れがある。近年では、これらの欠点の解決のために、前者ではシステムがイニシアティブを取って自身から例を生成したり、また、後者では視覚的なプログラミング中でも汎化エンジンの支援を提供することによって、プログラミングの負担を軽減していたりする。

1.4.4 個々のPBE-GUIシステムの特徴(1)：GUI構築システム

ここでは、個々のPBE-GUIシステムの特徴を述べる。まずシステムのGUI構築を支援するシステムを挙げる。

(1) DEMOII

DEMOII[Fischer92]は、PBEによってGUIのグラフィカルな挙動をプログラミングするシステムである（図1.4-1、注：本節の図はすべて本節末に掲載）。その最大の特徴は、複数の例を与えることによって徐々にプログラムを修正していくことができる点である。また、複数の例を用いることによって条件分岐も実現している。図1.4-1では例として、X-Windowsのxeyesと同じ挙動を示すGUIオブジェクトを生成している。

(2) Marquise

Marquise[Myers93]は、MacDrawのような図形エディタのインタフェースの構築やカスタマイズをPBEによって実現するシステムである（図1.4-2）。プリミティブな図形オブジェクトに対するマウスのクリック選択やドラッグなどのインタラクションを例示し、さらにそれらの複数の図形オブジェクト間をまたぐ操作を例示することによって、メニューやパレットを構築したり、その他の図形エディタの挙動を例示のみにより構築できる。図形エディタの挙動は予め汎用性のある形でパラメータ化してあるので、PBEの役割はそのパラメータ値を例の中から抽出することに限定されている。よって、推論の能力は際だったものではないが、実際にMacDrawの直接操作をほぼすべて例示によりプログラミング可能であるなど、非常に実用性の高いシステムに仕上がっている。

(3) Layouts by Example

Layouts by Example[Hudson93]は、複雑な内部構造を持ったデータ（例えば、家系図やプロセスネットワークのトポロジの図）を視覚化する際の描画ルーチンをPBEを用いて作成することを試みている（図1.4-3）。このシステムの特徴は描画ルーチンの訂正を行う手法にある。ユーザは、システムが提示した複数の視覚化の例から選択したり、汎化のやり直しを指示することによって、間接的に描画ルーチンの訂正を行うことができる（図1.4-4）。具体的なシステムの挙動としては、全順序が定義された図形オブジェクトの列を、いかに二次元上のパスにレイアウトするか、といった比較的単純な問題を解くものである。本研究の重要な点は、従来の複数の例を用いたPBEでは、ユーザが複数の例を提示する必要があったので、対話が煩雑になりがちであった。このシステムはこの問題に対して新しい方向性を示したといえる。

(4) Inferring Constraints from Multiple Snapshots

Inferring Constraints from Multiple Snapshots[Kurlander91]は、複数の絵から幾何制約を推論するシステムである（図1.4-5）。ただしこの場合、幾何制約とは静的な図を配置するための幾何制約ではなく、ユーザの直接操作に反応して変化する絵の構造を記述するための幾何制約である。このシステムの特筆すべき点は、作成したGUIが直接操作によって意図したとおりの挙動をしない時、正しい挙動をその場でデモンストレーションするだけでGUIを修正できるという点である。例として、複雑なリンクによって構成された机のランプにおいて、リンクを様々な角度に曲げた複数の例からリンクの自由度を検出し、必要な幾何制約を抽出している。

(5) TRIP3

TRIP3[宮下92]は、TRIP2システム[松岡92]で提唱されたBi-directional Translation Model（双方向変換モデル）上に構築されたPBE-GUIのシステムである（図1.4-6）。TRIP2の双方向変換モデルは、アプリケーションのデータとその視覚化された図形オブジェクトとの対応関係を宣言的な規則の集合のみで記述し、その上での「視覚化写像」及び「逆視覚化写像」を一種の制約解消問題として捉える。これにより、少量の明確で宣言的な規則のみで、広範なアプリケーションに対するDMを容易に実現することが可能となったことが報告されている。しかし、TRIP2では規則の指定がテキスト形式であり、GUIの視覚的な表示・操作との直感的な対応がとりにくいといった欠点も判明した。そこで、TRIP3においては、GUIプログラマは対応関係の規則を直接与えるのではなく、アプリケーションのデータと、対応する視覚化の具体的な例をGUIを使って与えるのみで、それらの例から計算機が規則を自動生成する。具体的には、GUIデザイナが、サンプルとなるデータとそれに対する視覚化の具体例を図形エディタを用いて描画する。システムはその図的構造、つまり、図形の各要素に成り立つ幾何制約とアプリケーションデータとの対応関係を推論する。さらにユーザが規則生成のコマンドを与

えると、システムはその対応関係を汎化して一般的な双方向の変換規則を生成する。生成された変換規則により、TRIP2の双方向変換エンジンがエンドユーザに対して、MacDrawのようなフリーハンドのエディタを提供し、ユーザはデータないしはその視覚的表現のどちらを編集しても、対応する相方にその変更が反映される。TRIP3はNeXTStation上で動作し、組織図、家系図や、データベースのスキーマエディタのGUIの自動生成などが可能になっている。

(6) IMAGE

IMAGE[宮下94]はTRIP3の後継システムであり、TRIP3の以下のような欠点を解消している（図1.4-7)。PBE-GUIで問題となる点は、①多くのPBEシステムは、生成されるプログラムをテキストとして表現し、折角のビジュアルなインタフェースの側面を生かし切っていない、②PBEシステムでは、完全に人間の意図を読み切ることは難しく、生成されたプログラムに対して何らかの直接的な変更法を提供するが、その方法が例えば単なるプログラム編集だったりすると、本来の直接操作のみによるインタフェース構築の目的を外れてしまう、③最も重要な点は、多くの場合、人間はどのような例を与えればよいのか、必ずしも明確に分かっていないことが多い、ことである。例えば、複数例を与えるPBEシステムの場合、果たしてどの時点でプログラムの「完全性」が満たされているかどうか確信が持ちにくいといった問題や、さらにはシステムの汎化能力に限界があるため、ユーザが意図した汎化を得るためには、システムの汎化アルゴリズムに精通していなければならないといった状況が起きる。IMAGEでは、この解決のために「複数例の対話的な修正によるプログラミング」という手法を用いている。従来の手法ではユーザが複数の例を一度にシステムに与え、システムはそれらの例を比較することにより一般的な規則を見つけ出していたが、それに対して、IMAGEではシステムがその時点での推論の結果を例を通じてユーザに示し、その例を訂正することによってユーザは新しい例をシステムに示す。つまり、システムが与える例示に対して主導権を持ち、ユーザはシステムが例示によって表している内部の推論状況に適切な修正を加えることにより、システムの推論を望ましい方向へ導くことができる。これにより、システムは汎化の範囲をうまく特定することができ、汎化の探索空間を大幅に削減できる。

(7) Inference Bear

Georgia Tech大学のInference Bear[Frank94]は、GUIのインタラクションに特化した汎化推論エンジンが主体のインタフェースビルダである（図1.4-8)。基本的には、GUIの状態を表す（図形エディタ上の図形オブジェクトなどの）オブジェクトのインタラクションの前後を比較し、その属性（つまり、フィールド値）の変化の相関を見て、そのような変化を起こすプログラムを生成する。推論モジュールは二つに分かれており、まず前処理用のcompactorモジュールにより、汎化の対象のフィール

ドの数を特定し、汎化推論モジュールに渡す。汎化推論モジュールはパラメタ化されており、汎化対象のオブジェクトのフィールドの型により、その型専用の汎化エンジンが呼び出される。各エンジンには、その領域固有であり、かつGUIに適した汎化戦略が内包されている。例えば、整数の汎化エンジンの場合は、複数の例示を与えたとき、それらを満たす変化の式を線形な方程式の集合とみなし、ガウスの消去法により変化の式を導出し、与えられた例示の集合をすべて満たすものの中で、最も簡単な解を探そうとする。通常のPBEと違い、ループの検出など、繰り返しに対する汎化は行わない。

1.4.5 個々のPBE-GUIシステムの特徴(2)：エンドユーザ支援システム

さらに、GUIのエンドユーザを支援するためのマクロ（シェル）プログラミングシステムを以下に挙げる。

(1) MIKE

MIKE[Olsen88]はマクロ機能をサポートした最初のUIMSであり、GUIにおいて既存のコマンドを組み合わせて新しくマクロを定義し、それを従来のコマンド体系に組み込む機能を提供する（図1.4-9)。しかし、このシステムはGUIといっても、基本的にはメニューによるコマンド起動のみを前提としているので、現在広まっている様々なインタラクションテクニックを用いたGUIと比較するとその古さは隠せない。しかし、この制限によってPBEによるユーザ定義マクロの作成を定型化している。MIKE UIMSにおいては、インタラクティブインタフェースは、データの型と各データ型に適応できる関数群として定義される。例えば、以下に簡単なドローイングプログラムの定義を示す。

1. データ型：

   Point, String, DrawPrimitive, Picture

2. 関数：

   NewPicture(Name: String) ... 新しい絵を生成する

   PictureName(Name: Strihg): Picture ... 絵を名前で探す

   IdentifyPicture(Location: Point): Picture ... 絵そのものを指すことで指定する

   OpenPicture(Pict: Picture) ... 既に存在する絵を起動する

   DeletePicture(Pict:Picture) ... 絵を消去する

   AddLine(P1, P2: Point) ... 絵に線プリミティブを加える

   AddText(At: Point; Txt:String) ... 絵に文字プリミティブを加える

   MovePrim(Prim: DrawPrimitive; To:Point) ... 選択されているプリミティブを移動する

   DeletePrim(Prim: DrawPrimitive) ... 選択されているプリミティブを消去する

   PickPrim(Where: Point): DrawPrimitive ... 指されているプリミティブを選択する

MIKEでは、一つの型に対してできるだけ多くの入力手段を提供している。例えば、整数の入力に対して、①キーボードでタイプする、②整数を返す関数の名前をタイプする、③整数を返す関数をメニューから選ぶ、④ボタンを押すことで関数を呼ぶ、といった手段である。これらの手段を予めMIKEが提供することで、入力手段に依存しない「値」を得ることができる。こうして与えられたデータ型、関数群から、MIKEは各関数の返す値の型ごとに別々のメニューを作る。ユーザは一つのコマンドに対する引数をMIKEを通じてすべて与えると、MIKEはその完全なコマンドをアプリケーションに送る。よって、アプリケーションにコマンドを送るまでは、入力は完全にMIKEの管理下にあるのでundo機能が実現できる。こうして自動生成されたメニューは、当然ユーザにとって必ずしも使いやすいものではない。そこで、MIKEでは以下のようなGUI修正機能を備えている（これらの機能はWYSIWYGスタイルで実現されている。）。

・メニューを木状に構成する。

・アイコンを生成して、そこにコマンドや部分評価した値を割り当てる。

・ファンクションキーなどにコマンドを割り当てる。

・メッセージ（エコー、プロンプト）を修正する。

・ウィンドウにコマンドを割り当てる。

マクロの定義は以下のような手順で行う。まず、「Create(Name: String)」コマンドによって、新しいマクロを宣言し、「NewParameter(Name: String)」コマンドによって、マクロに対するパラメータの名前を設定する。この段階では、各変数に型はない。そして、「StartRecord」コマンドによって、マクロの記録が始まり、マクロの中身を表示するウインドウが現れる。ユーザは、普段どおりにコマンドの起動を行う。通常と異なる点は、型の決っていないマクロパラメータの名前がどのメニューにも現れるということである（注：メニューは関数の返す型ごとにある）。メニューからその項目を選ぶと、そのマクロパラメータの型として、現在求められていた引数の型が割り当てられる。デモンストレーションを続けるためには、この選択したマクロパラメータに対するサンプルの値が必要であるから、値が通常と同じように入力される。しかし、マクロのための記録中には、このサンプルの値そのものではなく、マクロパラメータへの参照が記録される。こうしてナイーブな手法ではあるが、パラメータ化が実現されている。得られたマクロを専用のエディタで編集することにより、条件分岐、ループを加えることができる。ユーザがメニューからIF文を選択すると、システムはその引数としてTrue/Falseを返す関数を求めてくる。この関数の結果がTrueなら次のコマンドはTrueに対する選択肢、FalseならFalseに対する選択肢となる。もう一方の選択肢は、マクロをエディットするときに再度デモンストレーションする必要がある。WHILEループも同様にして生成できる。ただし、この場合ポインタをループの本体の中に持っていったとき、何回目の繰り返しに飛ぶべきなのかはっきりしない。

結局、MIKEはGUIとはいっても基本的には関数呼出しをメニューで行えるようにしただけで、

GUI生成で本当に労力のかかる部分、つまりユーザとのリアルタイム、イベントドリブンな対話をいかにして制御するかという問題には、解決策を与えていないといえる。

(2) SmallStar

SmallStar[Halbert93]は、オフィスでの定型的な処理（ファイル操作など）のマクロをPBEを用いて作成するシステムである。このシステムは、でき上がったマクロをプログラム言語風に表現し、ユーザが後からそれを修正して用いることを前提としている。よって強力な汎化機構などは備えていない（図1.4-10)。SmallStarはSmalltalk上で構築されたStarのサブセットであり、文書、フォルダ、ファイル用引出し、プリンタ、テキストの編集などをサポートしている。SmallStarで新しく加わった要素は以下の二つである。

① プログラムアイコン： プログラムを表す。

② プログラミング開始ボタン、終了ボタン： 開始ボタンを押した時から終了ボタンを押した時までのユーザの操作が記録されプログラムとなる。プログラムは英語風であるが、ユーザはこのプログラムを直接編集することはできない。

SmallStar上でマクロを定義している際に、あるオブジェクトの選択の意図としては以下のようないくつかの可能性がある。

① そのオブジェクトは定数である。

② そのオブジェクトは、プログラム中の直前の動作（例えばCOPY）で生成されたものである。

③ そのオブジェクトは、プログラムに対するパラメータとして呼び出す時に渡されている。

④ そのオブジェクトは、そのオブジェクトの持つ何らかの特徴に基づいて選択された。

ここで、SmallStarでは推論は用いず、システムは最も理にかなっていると思われるデフォルトの選択方針を用いてプログラムを生成し、ユーザは後でそのプログラムを変更する、といった方策を採っている（図1.4-11)。このオブジェクトの選択方針はここではdata descriptionと呼ばれ、これを表示するdata description sheetをユーザが編集することにより、一般的な状況に対応するプログラムが生成される。

(3) AIDE

AIDE[Perinot93]は、アプリケーションに依存しない汎用的なマクロ生成機能を備えたGUIフレームワークである（図1.4-12)。前出のMIKEはユーザのインタラクションをメニューによるコマンド起動に限定することで汎用的なPBEを実現していたのに対して、AIDEはアプリケーションプログラムをメッセージ駆動型に限定することによって汎用的なPBEを達成している。しかし多くのGUIは元々メッセージ駆動型なのでこのことは大きな制限にはならず、AIDEは非常に実用的なフレームワーク

といえる。

AIDEでは、マクロ生成時には、イベントマネージャは高レベルイベントに関するさらに詳細な情報をアプリケーションから引き出す。そしてこの情報を基に高レベルイベントの汎化を行い、汎化されたイベント列によってマクロを生成する。マクロ実行時には、イベントマネージャはマクロ中の汎化されている高レベルイベントをアプリケーションに実体化させ、それを記録した後でアプリケーションに対応する動作をさせる。マクロ生成時には、まずループを検出し、次に変数の汎化を行う。

① ループの検出では、同じ種類の高レベルイベントが同じオブジェクトから発生したことを探す。

② 変数の汎化では、ループ中で対応するイベントの引数の連続列から何らかのパターンを見つける。現在サポートしている列は以下のとおりだが、これら以外の列をサポートするように拡張することは容易である。

数列：定数、線形、昇順、降順

文字列：同じ部分文字列、日付、月名

点列：2つの数列から成る

クラス列

列の推論は、Eagerの推論エンジンと基本的には同じである。例えば、数列の推論の場合、4、9とくれば4から9までの変化分5の等差数列と推論し、さらに14がくれば4から14までの変化分5の等差数列となる。次の値が15のときは等差数列ではなくなるので、4から15までの今度は昇順数列となる。このように複数の例を与えることで、より一般化されたマクロの定義がなされる。

PBEシステムにおいて、ユーザの意図をいかに推論するかが非常に重要である。AIDEでは、以下の二つの解決策を用いている。

オブジェクト選択：AIDEでは、アプリケーションの開発者が強力な検索コマンドを用意することを推奨している。ユーザはこの検索コマンドによって、共通の性質を持ったオブジェクトを検索する。そして、そのときには自然と「意図した」選択のパターンを示してくれることになる。さらにこのような検索コマンドを繰り返して用いることによって、比較的複雑な検索（論理和、積、否定など）も実現できる。

マクロの組合せ：入れ子状のループやサブルーチン呼び出し、再帰呼び出しの認識は一般に難しい。AIDEではアルゴリズムを小さなマクロに分けて、それぞれのマクロの間で呼び出すという方法をとっている。この方針はシステムの推論エンジンの負担を減らし、ユーザが自然にアルゴリズムを記述できるようにしている。

(4) Eager

Eager[Cypher91]はHyperCard上での操作に対するマクロを作成するシステムだが、Metamouseと同様に推論したループを積極的に実行してユーザに示している（図1.4-13)。従来のPBE-GUIシステムと異なり、Eagerでは、ユーザにその推論の正当性を問い合わせることによってユーザの作業を中断することをしない。代わりに、Eagerはユーザの動作のループを検出し次に行うであろう動作（例えばあるボタンを押すこと）を推論すると、その動作を先取りしてユーザにどのような推論をしたか示す（*anticipation*という：例えばそのボタンを緑色で反転させる)。この動作を何回か繰り返して、ユーザがEagerの推論が正しいと確信したら、ユーザはEager Iconをクリックする。すると、Eagerはそれ以降の動作を推論に従って自動的に行う。

Eagerの設計の大きな方針は以下の4点である。

① ユーザの作業を妨げることを最小限に抑える。よって、Eagerはユーザに陽には問合せを行わない。例えば、ループの最初をユーザは明示的に示す必要はない。

② ユーザの作業を推論し示す際、ユーザの実際の動作と同レベルで示す。例えばボタンのクリックを推論したら、そのボタンを反転させる。

③ ユーザの作業から一般的な法則を推論してそれをユーザに示す際、その一般則をその場の具体例に当てはめた形で示す。例えば、選択ボタンi(i = 1,2,3,...)を押すと推論したら実際に次にユーザが押すはずのボタンを反転させる。

④ Eagerの推論の正当性の確認は、ユーザの通常の動作を通じて行われる。つまり、Eagerによって反転されたボタンをユーザが押したら推論の正当性を認めたことになり、また他のボタンを押したら推論の誤りを示したことになり、この場合Eagerは新たな推論を始める。

EagerはHyperCardからユーザの高レベルな動作に関する情報を得て、そこから以前のユーザの動作との類似性を探す。この場合の「類似性」とは、以下のようなものである。

・コマンドが同じタイプである。

・数値データが、①同じ、②連続、③線形、④ある程度の誤差を許して線形である。

・文字データが関連している。例えば、月の名前、曜日など。

Eagerは類似している2点を見つけると、その2点の間の部分が1回の繰り返しに相当すると仮定する。Eagerはまた、HyperCardに関して以下のようなdomain knowledgeを前提知識として用いている。

・CardはStackの一部である。よって連続したカードに関する操作はループであり、それはstackの最後まで続くと推論できる。

・「button-location-X」という数値は画面上の座標なので、そのデータは線形であるかもしれない。

EagerがHyperCardから受け取るユーザの作業情報は高レベルなので、実際にはその中にいくつか

の小さな動作を含んでいることがある。その場合、anticipationはその小さな動作を各々ユーザに示す必要があるので、Eagerは高レベルな作業と低レベルなユーザの各々の動作を両方記録する必要がある。また、ユーザがある作業を行う場合、同じ結果を得るためでもいくつかの方法がある。Eagerでは、そのような低レベルな差異は無視し高レベルなコマンドとしての作業の同一性を調べる。

(5) Pursuit

Pursuit[Mondugno94]は、視覚的なシェル言語のプログラムを例示によって構築するシステムである。それぞれのプログラムの状態や、操作対象は、アイコンとそれを含むパネルによって表現される。例えば、`reports'といったディレクトリ内の`.tex'といった拡張子を持つファイルをすべて圧縮するという操作は、図1.4-14のように表現される。操作の列は、このようなパネルの列として、デモンストレーションを行う間、常時表示・更新が行われる。Pursuitは操作の列を自動的に複合したり、操作対象を自動的に推論したりする機能を持つ。また、汎化により、ループを検出したり、複数例から条件分岐を検出する機構を持つ。さらに、ダイアログボックスによるユーザとの対話に対して、自動的に応答したりする機能もあるが、これらは「メタな」ダイアログボックスにより指定する。プログラムの変更は、追加例示をシステムが自動的に要求することがある。例えば、ファイルの集合に対するループ時にシステムが例外を検出すると、その例外の扱いに関して追加の例示を行うように要求する。また、結果の視覚的なスクリプトを直接操作のエディタで編集することも可能である（図1.4-15)。

Pursuitは、操作領域をシェルプログラミングに限ることにより、有効な推論を行えるようにし、豊富なアシスタンスを例示時にシステムが提供することによって、ユーザが自分の意図を伝えやすくなっている。かつ、エンドユーザにも直感的に分かりやすく、情報が豊富な視覚的シェル言語をデザインすることによって、推論されたプログラムの結果表示や修正に対応している。このように、領域を限定し、かつその領域でなるべくユーザの負担を減らすような例示の手法は、今後の大きな研究課題であろう。

(6) Chimera、Mondrian

Chimera（History-Based Macro）[Kurlander92]、Mondrian[Lieberman92]は、ともに図形エディタのマクロをPBEによって作成しようという試みである（図1.4-16～18)。どちらのシステムも幾何的な関係の推論機構や、マクロ適用のパラメタの汎化推論機構を備えている。特徴的なのは、マクロをユーザが例示のため実行した時のグラフィカルなコンテクストのスナップショットの列として表現している点である。これは、例えばChimeraではStoryboardと呼ばれ、ユーザは視覚言語として直接操作で編集が可能である。のみならず、そのコンテキストに表示されているグラフィカルなオブジェクトの存在がマクロ適用の条件になったりする。従来の多くのシステムがグラフィカルな操作に対するマ

クロでもテキストで記述していたのに比較すると、その分かりやすさは画期的である。Mondrianはまた、推論結果を合成音声でユーザにフィードバックするのもユニークである。

(7) Metamouse

Metamouse[Maulsby89]も、同じく図形エディタのマクロをPBEで作成するシステムである。先の二つの研究に比べて特徴的なのは、Metamouseはユーザの操作列の中で繰り返しを検出し、ループを自動生成する点である。また、その推論したループをユーザの操作に先行して「実行して見せる」ことで、ユーザに具体的に推論の中身を示している点も画期的である。なお、Metamouseの解説は、[萩谷91]に詳しいので、ここでは割愛する。

(8) GA Graph Layout

GA Graph Layout[Masui94]は、Genetic Algorithmの中でも、特にGenetic Programmingの技法を用いて、グラフのレイアウトをユーザ例から決定していくシステムである（図1.4-19)。グラフレイアウトは古くから研究されているが、ユーザのレイアウトの善し悪しに関する直感的な好みを正確に定式化するのは、個々の性質が複雑に絡み合い、困難である。このシステムでは、ユーザに好みの例を提示してもらうことにより、それをシステムが学習していく。具体的には、ユーザはあるトポロジのグラフに対して、二つの幾何的レイアウト例を提示され、どちらか好ましい方を選ぶ。この操作が、数十の様々なトポロジとそのレイアウトに対して繰り返される。これらの例を基に、レイアウトを行うアルゴリズムが用いる評価関数を、Genetic Algorithmの一手法であるGenetic Programmingと呼ばれる最適化手法により生成する。生成された評価関数は、システムが提示した例に関する様々なユーザの好みを反映したものであり、それを新たなグラフのデータに適用することが可能になる。

(9) ProDeGE+

ProDeGE+[Sassin94]は、MetamouseやChimeraに類似した、図形編集に対するマクロ生成をPBEで行えるシステムである（図1.4-20)。特徴としては、図形編集に関する領域知識を新たにシステムに追加することが可能で、それにより強力な汎化機能を有するようになることである。[Sassin94]では、2次元の凹多角形の立体化という、大がかりなデモンストレーションの例を挙げており、そこでの幾何推論や汎化推論はChimera/Metamouseと比較して大変強力であるが、実際の推論エンジンはまだ実装されておらず、果たして本当にそのような推論が可能かどうか不明である。

1.4.6 まとめ

PBE-GUIは、まだまだ「若い」分野であり、それだけに様々な研究がなされている。しかし、少な

い例示からユーザの意図を完全に把握するのは、本質的に困難な問題であり限界もある。本節では、単に過去の「例示によるプログラミング」の手法に限定されず、インタラクティブシステムの特性を生かし、ユーザから積極的に必要な情報を得る、そのためのインタフェースの様々な試みを紹介した。今後は、より賢い推論により、より精度の高いプログラムを得るのみならず、より多くの情報を自然にユーザから獲得できたり、ユーザに提示したり、さらにユーザが容易に修正できるようになるシステムの研究が盛んになるだろう。

もう一つの方向としては、より高度で先進的なインタフェースへの適用である。本節で紹介したPBE-GUIシステムは、すべて2次元図形エディタなどの、いわゆるデスクトップインタフェースに限られていた。他の節で紹介されているように、最近では3次元、アニメーション、画像、表情合成、バーチャルリアリティや音声などのマルチメディアインタフェースや、Mosaicなどのネットワーク上の先進的なインタフェースの研究が盛んである。そのようなインタフェースの構築への適用、あるいはユーザスクリプトの作成支援などの領域はまだ未探索である。今後、PBE-GUIのそのような新しいインタフェースへの適用が大きな課題である。

今後、この技術を実際の製品に生かした例が登場するのも間近であろう。我国でも、先進的インタフェースの研究の一環として、この分野の研究が推進されることを期待する。

【参考文献】

[Cypher91] Cypher, A: Eager : Programming Repetitive Tasks by Example, Proceedings of Human Factors in Computing Systems (CHI'91), ACM, pp.33-39 (1991).

[Cypher93] Allen Cypher et. a. (eds.): Watch What I do: Programming by Demonstration, The MIT Press (1993).

[Fischer92] Fisher, G. L., Busse, D.E., Wolber D. A.: Adding Rule-Based Reasoning to a Demonstrational Interface Builder, Proceedings of UIST'92, ACM, pp.89-97 (1992).

[Frank94] Frank M.R., and Foley, J.D.: A Pure Reasonig Engine for Programming by Demonstration, Proceedings of UIST'94, ACM, pp.95-102 (1994).

[萩谷91] 萩谷昌巳: 視覚的プログラミングと自動プログラミング, コンピュータソフトウェア, Vol. 8, No. 2, pp. 27-39 (1991).

[Halbert93] Daniel C. Halbert, D.C: SmallStar: Programming by Demonstration in the Desktop Metaphor, in Watch What I Do: Programming by Demonstration, Alan Cypher et al. (eds.), The MIT Press, Chap. 5, pp.103-124 (1993).

[Hudson93] Hudson, S. E., and Hsi, C.:A Synergistic Approach to Specifying Simple Number

Independent Layouts by Example, Proceedings of Human Factors in Computing Systems (CHI'93), ACM, pp.285-292 (1993).

[Kurlander91] Kurlander, D., and Feiner, S.: Inferring Constraints from Multiple Snapshots, Technical Report Columbia University Computer Science, CUCS 008-91 (1991).

[Kurlander92] Kurlander, D. and Feiner, S.: A History-Based Macro By Example System, Proceedings of UIST'92, ACM, pp.99-106 (1992).

[Lieberman92] Lieberman, H.: Dominoes and Storyboards: Beyond ``Icons on Strings'', Proceedings of IEEE Symposium on Visual Languages, IEEE-CS, pp.65-71 (1992).

[Masui94] Masui, T.: Evolutionary Learning of Graph Layout Constraints from Examples,, Proceedings of UIST'94, ACM, pp.103-108 (1994).

[Maulsby89] Maulsby, D. L., Witten, I. H., and Kittlitz, K. A.: Metamouse: Specifying Graphical Procedures by Example, Proceedings of SIGGRAPH'89, ACM. pp.127-136 (1989).

[宮下92] 宮下, 松岡他: Declarative Programming of Graphical Interfaces by Visual Examples, Proceedings of UIST'92, ACM, pp.107-116 (1992).

[宮下94] 宮下, 松岡他: Interactive Generation of Graphical Interfaces by Multiple Visual Examples, Proceedings of UIST'94, ACM, pp.85-94 (1994).

[Mondugno94] Mondugno, F., and Myers, B.: A State-Based Visual Language for a Demonstrational Visual Shell, Proceedings of IEEE Symposium on Visual Languages, IEEE-CS, pp.304-311 (1994).

[Myers88] Myers, B. A.: *Creating User Interfaces by Demonstration*, Academic Press (1988).

[Myers90] Myers, B. A.: Creating User Interfaces Using Programming by Example, Visual Programming, and Constraints, ACM TOPLAS, Vol. 12, No.2, pp.143-177 (1990).

[Myers93] Myers, B. A., McDaniel, R. G., Kosbie, D. S.: Marquise: Creating Complete User Interfaces by Demonstration, Proceedings of Human Factors in Computing Systems (CHI'93), ACM, pp.293-300 (1993).

[Olsen88] Olsen, D. R. Jr., and Dance, J.R.: Macros by Examples in a Graphical UIMS,Computer Graphics & Applications, IEEE Vol. 8, No. 1, pp.68-78 (1988).

[Perinot93] Piernot, P. P., and Yvon, M. P.: The AIDE Project: An Application-Independent Demonstrational Environment", in Watch What I Do: Programming by Demonstration, Alan Cypher et al. (eds.), The MIT Press, Chap. 18, pp.383-402 (1993).

[Sassin94] Sassin, M.: Creating User-Interface Programs with Programming by Demonstration, Proceedings of IEEE Symposium on Visual Languages, IEEE-CS, pp.304-311 (1994).

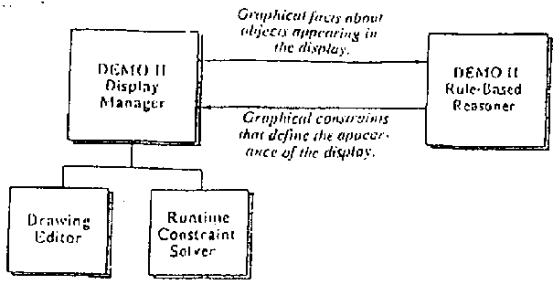


Figure 1: DEMO II Architecture.

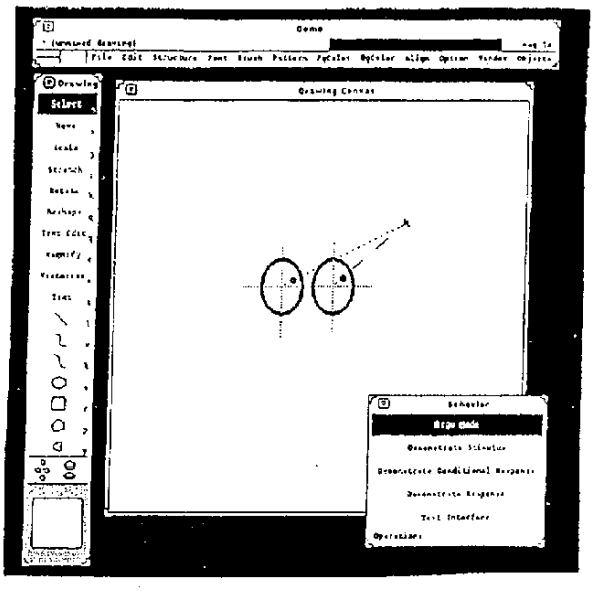


Figure 2: DEMO II User Screen.

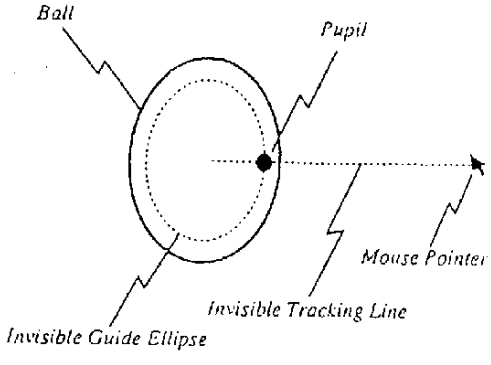
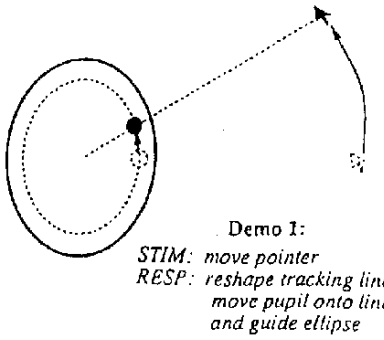
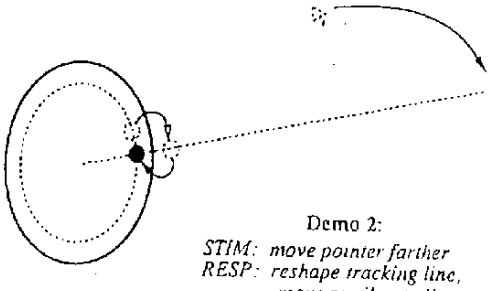


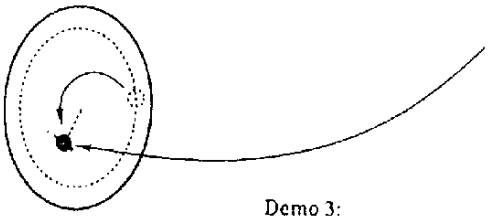
Figure 6: Initial Xeyes Drawing.



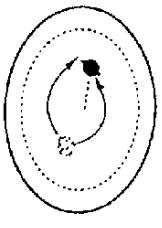
Demo 1:
STIM: move pointer
RESP: reshape tracking line, move pupil onto line and guide ellipse



Demo 2:
STIM: move pointer farther
RESP: reshape tracking line, move pupil onto line and guide ellipse



Demo 3:
STIM: move pointer inside ball
RESP: move pupil onto pointer



Demo 4:
STIM: move pointer inside ball
RESP: move pupil onto pointer

Figure 7: The Four Xeyes Demonstrations.

図1.4-1　DEMOIIによるXeyesの例示（出典：[Fischer92]）

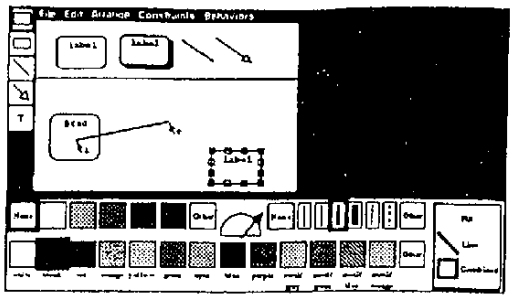


Figure 1:

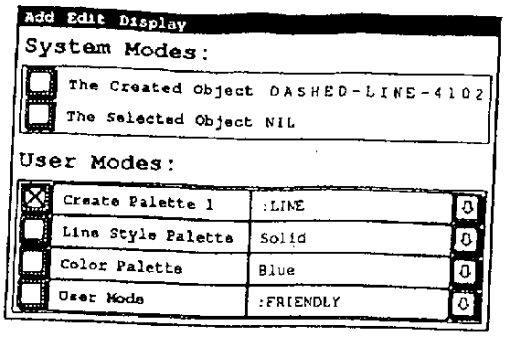


Figure 4:

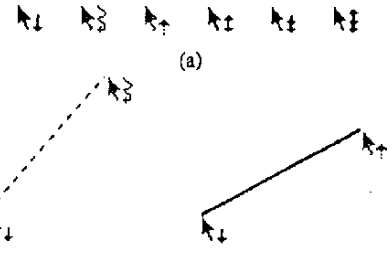


Figure 2:

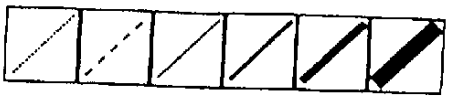


Figure 5:

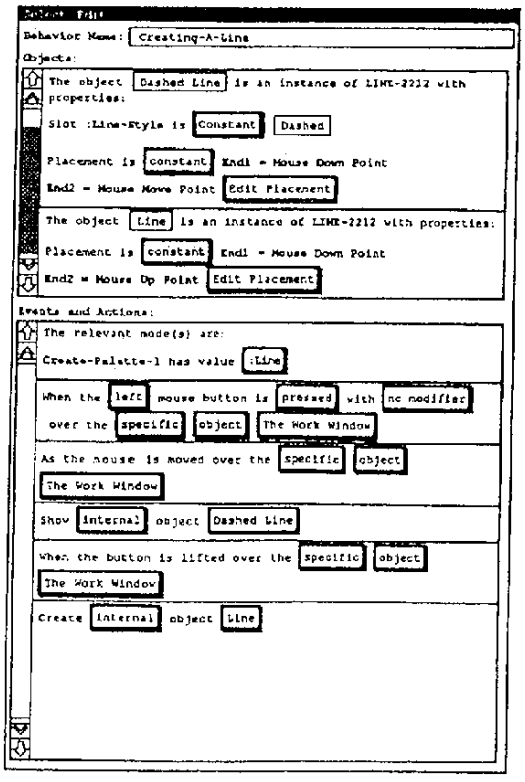


Figure 3:

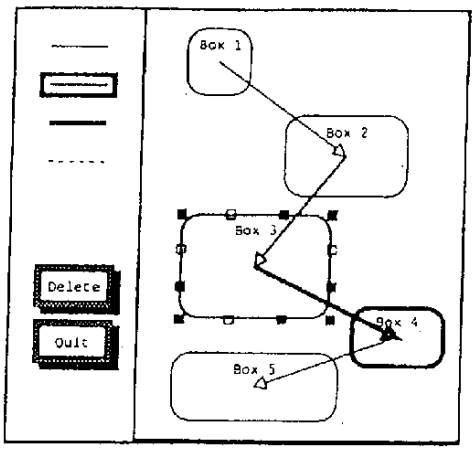


Figure 6:

図1.4-2　Marquiseによる図形エディタの例示（出典：[Myers93]）

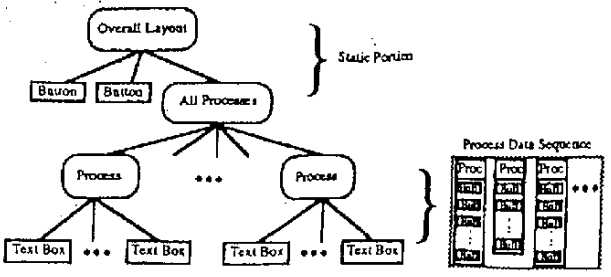


Figure 1. The Structure of an Example Layout

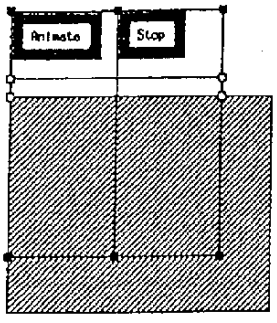


Figure 2. Using the Grid for a Static Layout

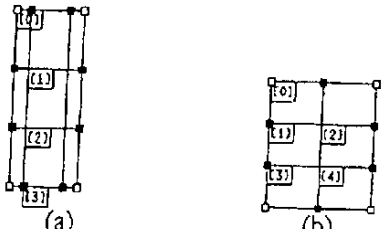


Figure 3. Two Example Layouts

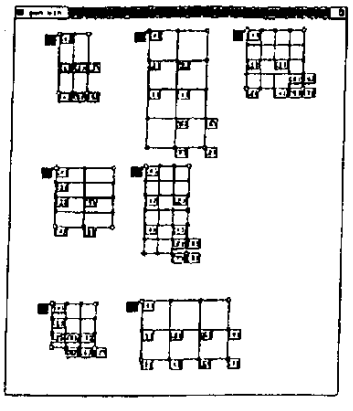


Figure 4. New Examples Illustrating Generalizations of Figure 3b

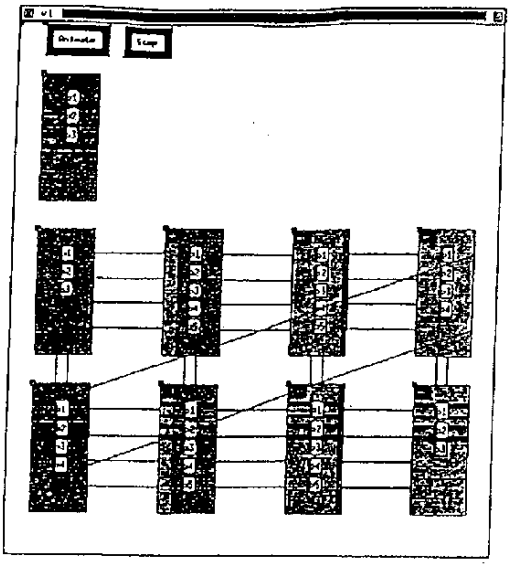


Figure 5. A Completed Visualization

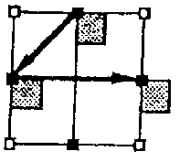


Figure 7. An Example Path

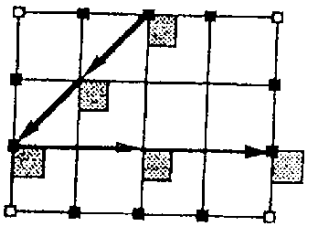


Figure 8. An Interpolation of Figure 7

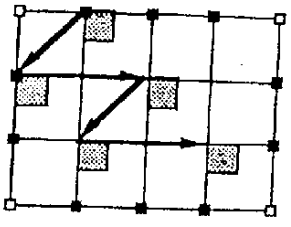
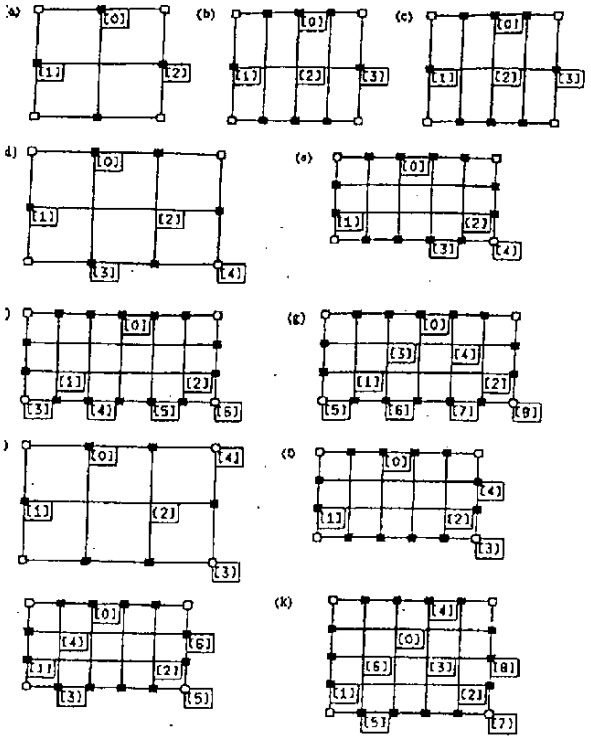


Figure 9. An Example Repetition of Figure 7

図1.4-3　Layouts by Exampleによる図形配置の汎化戦略（出典：[Hudson93]）



Examples of using generalization strategies: (a) original layout, (b) — (k) after applying generalization strategies (1), (3), (4), (7), (8), (9), (10), (13), (14), and (15).

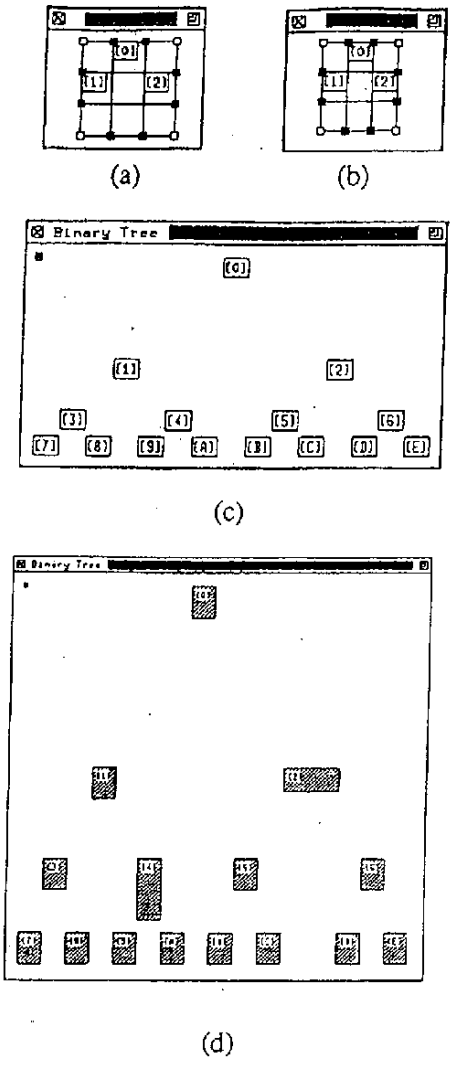


図1.4-4　Layouts by Exampleによる汎化の例と二分木のレイアウト生成（出典：[Hudson93]）

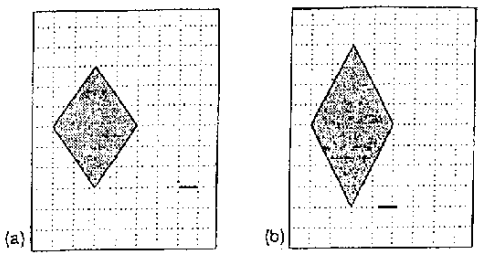


FIGURE 1. Two snapshots of a rhombus and line

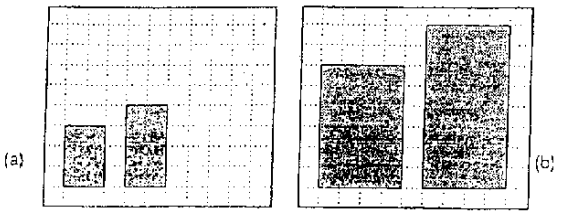
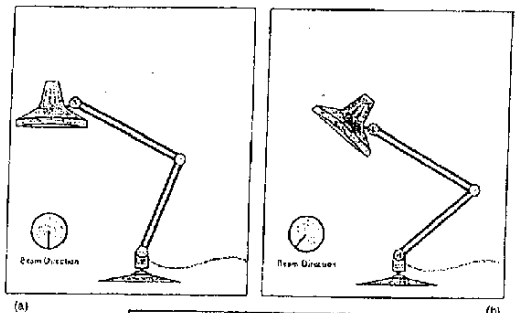
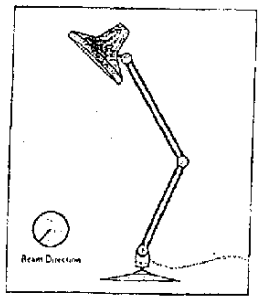


FIGURE 7. Two snapshots of a simple scene





IGURE 4. Teaching Luxo constraints. Three snapshots of valid configurations, provided as input.

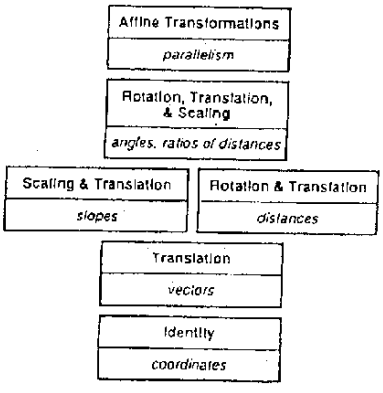


FIGURE 6. Transformations and the geometric relationships that they maintain. Reprinted with permission from [Bier86].

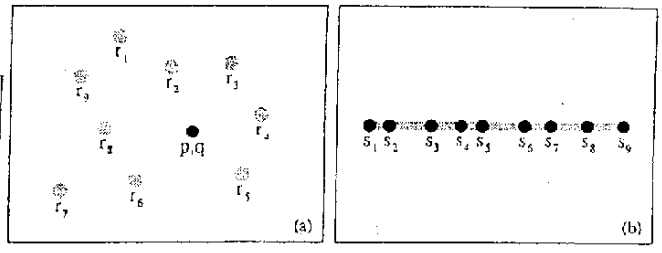


FIGURE 8. Two geometric relationships that lead to redundant constraints.

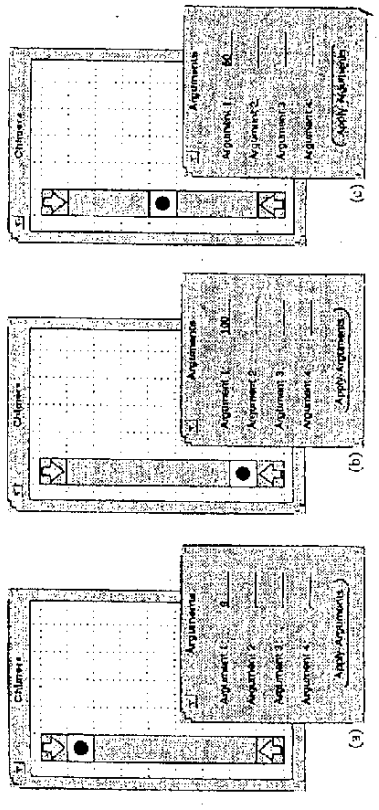
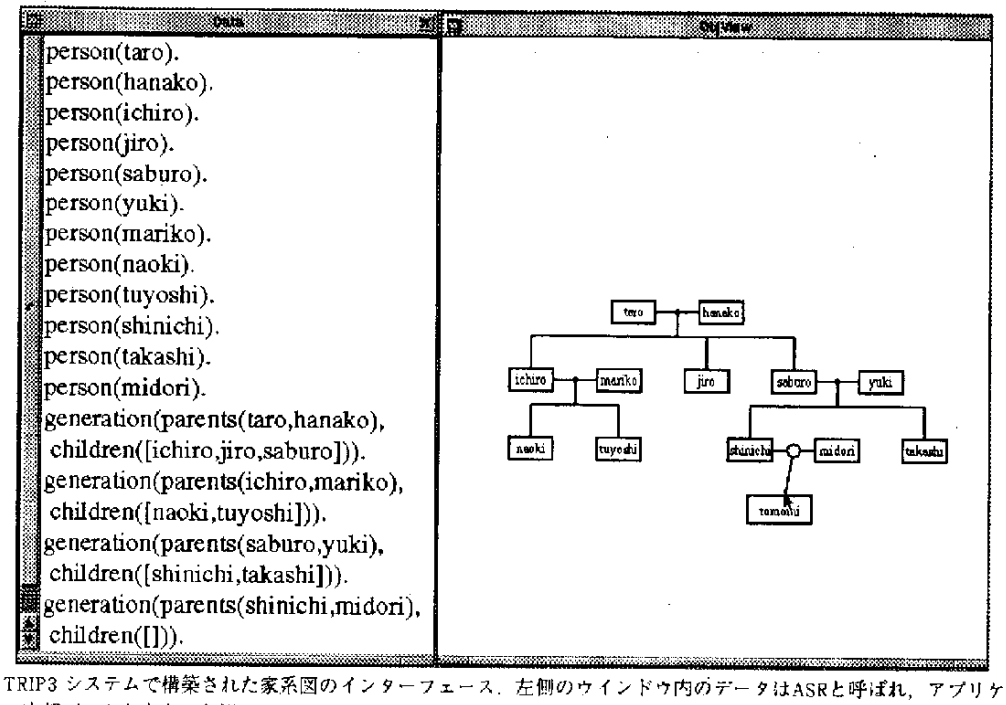
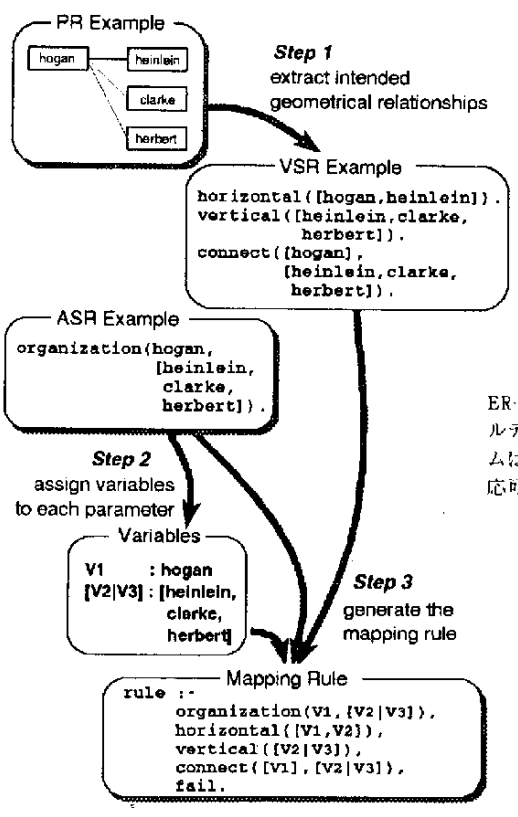


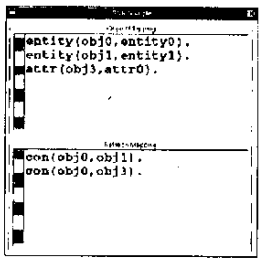
FIGURE 10. Dimensioning the height of a scrollbar. Initially two snapshots, (a) and (b), are specified. A new value for Argument 1 is entered in (c), and the scrollbar adjusts automatically.

図1.4-5　幾何制約の複数のスナップショットからの推論の例（出典：[Kurlander91]）

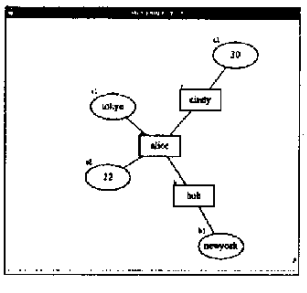


TRIP3 システムで構築された家系図のインターフェース．左側のウインドウ内のデータはASRと呼ばれ，アプリケーションの内部データを表す．右側のウインドウはその図形表現であり，ユーザはMacDraw風のエディタで自由に編集できる．それぞれのウィンドウでの編集は，自動的に対応するデータ・図形に反映される．この対応関係を表すマッピング規則が，例示プログラミングによって生成される．





ER-Diagram の例示．プログラマは，ER-Diagramのサンプルデータと，それに対応する図形表現を与える．システムは，それらの例示を汎化し，より一般的なデータに適応可能なマッピング規則を生成する．



生成されたマッピング規則により，エンドユーザ向けのER-diagramのインターフェースが構築される．

図1.4-6　TRIP3システム（出典：[宮下92]）

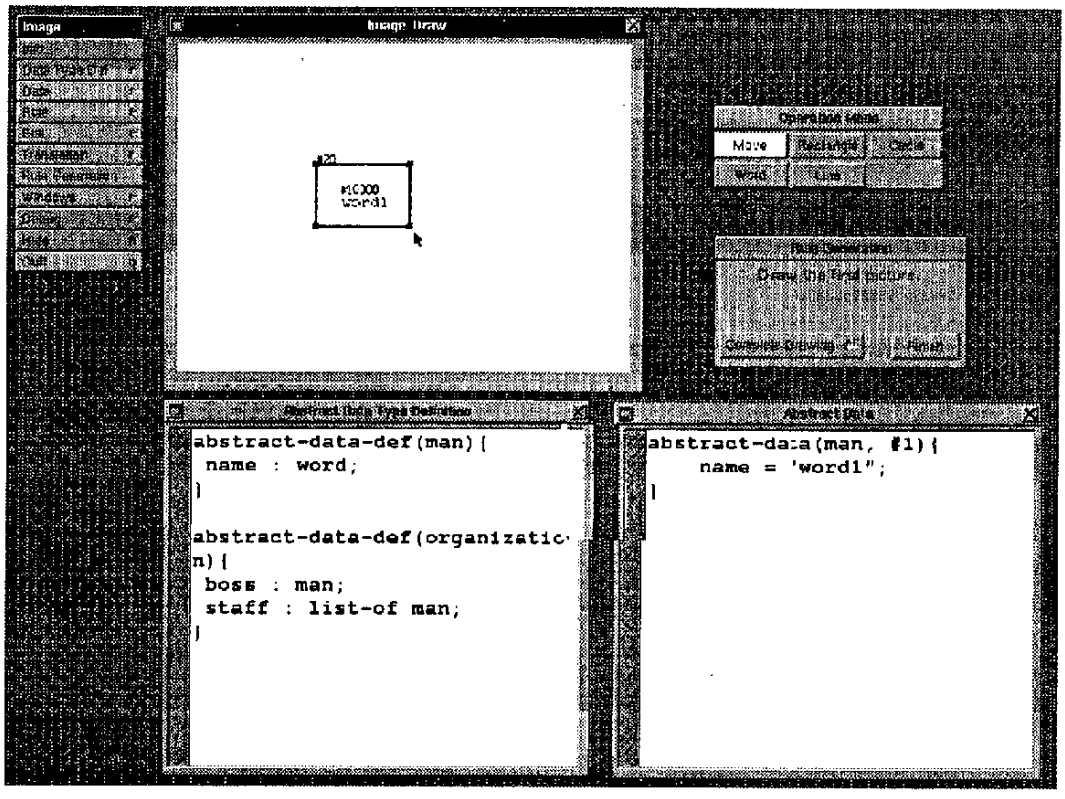
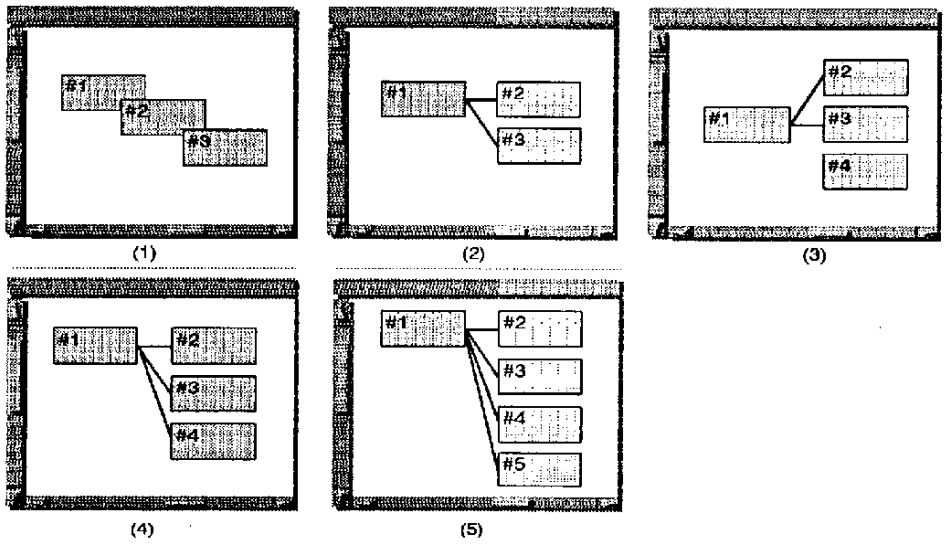


IMAGE システムの全容：画面の上部が図表現の例示の編集・修正を行う図形エディタ，左下のウィンドウがデータ構造の型定義を表現し，右下のウィンドウがシステムが生成した新たな例示に対する型のインスタンスを表示する．



IMAGEでの組織図の例示．(1)ユーザが直前に例示した「人物」の視覚表現と，組織図の型定義の情報を用い，「ボスに対して部下が二名」という最も簡単な型のインスタンスを右下のウィンドウに提示するとともに，図形エディタ上で対応するボスと二名の部下を表示する．(2) ユーザはそれを編集し，型のインスタンスに対応する組織図を完成させる．(3)システムは新たに部下の人数を三名に増やすと共に，その図を表示する．(4) ユーザは図を見て，以下の点を図形エディタ上で修正する：(i) 部下の集合は上辺でボスと整列されるべきである．(ii) 三人目の部下もボスと線で結ばれているべきである．システムはその修正を受け，図変換の規則を修正する．(5) システムは修正された変換規則を用い，さらに部下を四人に増やした例を表示する．ユーザは，今度は視覚化に満足し，例示プロセスの終了をシステムに通知する．

図1.4-7　IMAGEシステム（出典：[宮下94]）

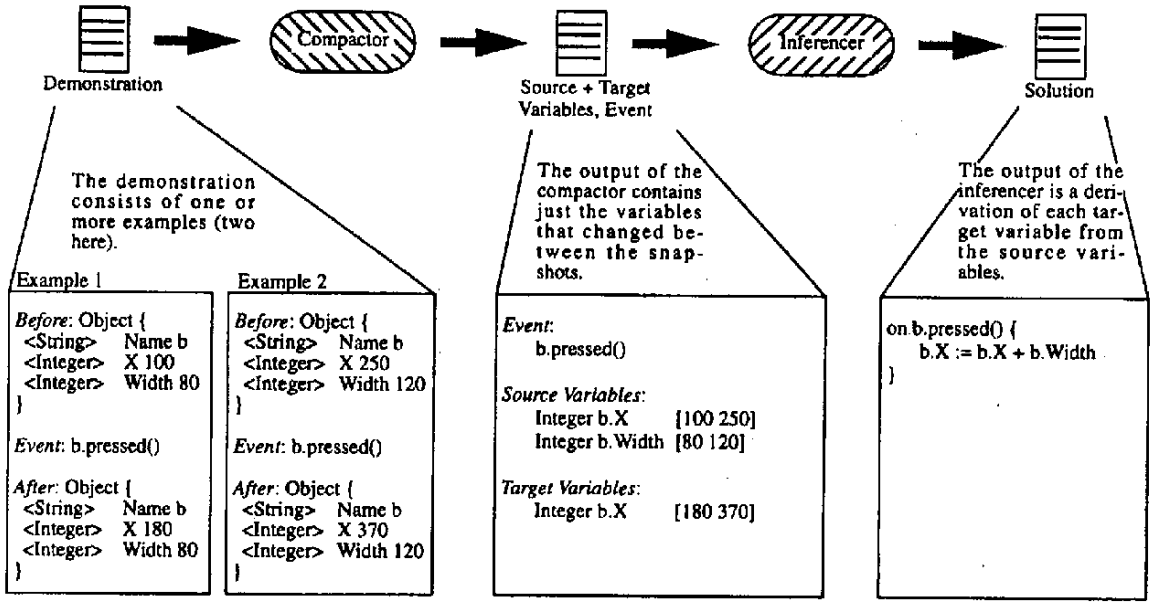


図1.4-8　Inference Bearの推論・汎化アルゴリズム。複数の例示及びそれぞれの操作の前後のオブジェクトの状態よりSource/Target変数を決定し、変数間の関係を推論する。

（出典：[Frank94]）

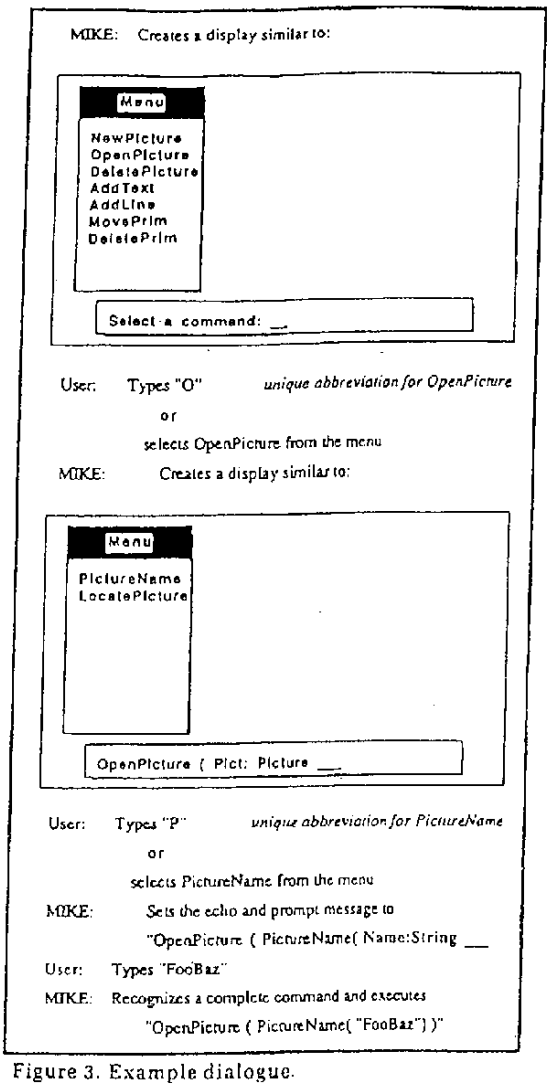


Figure 3. Example dialogue.

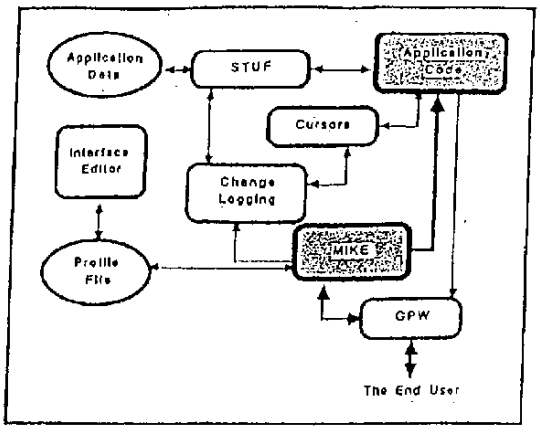


Figure 4. System architecture.

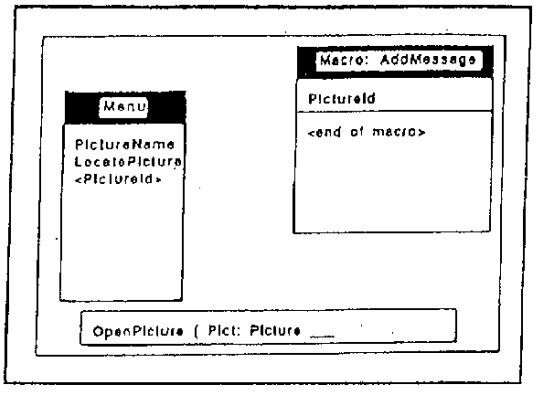


Figure 5. Creating a macro.

図1.4-9　MIKEのシステムアーキテクチャとマクロ生成の例（出典：[Olsen88]）

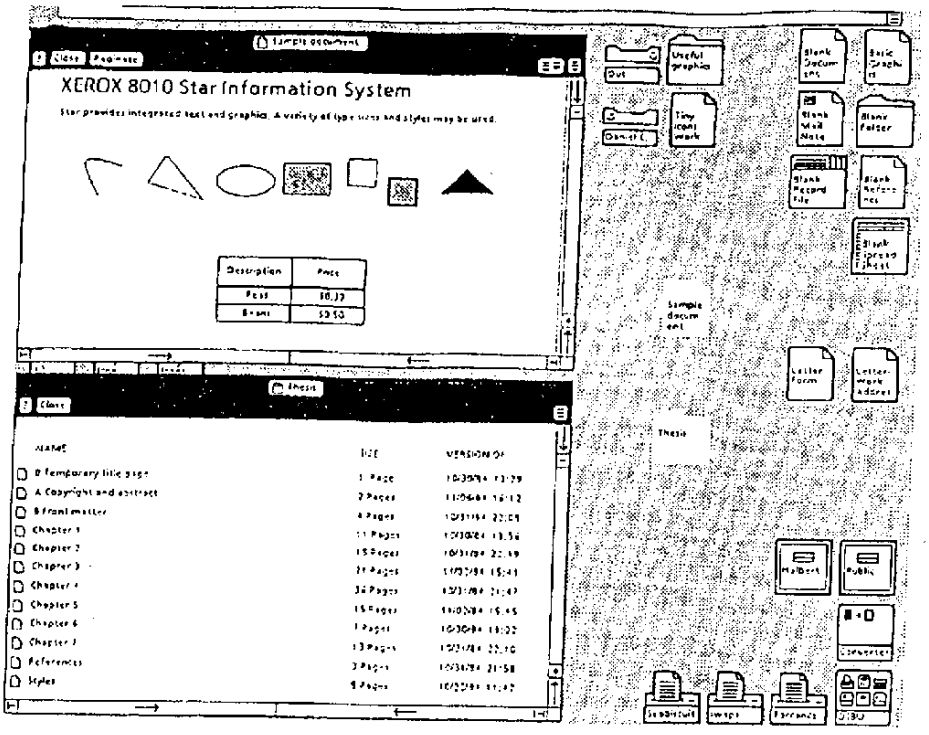


Figure 1. A Star desktop.

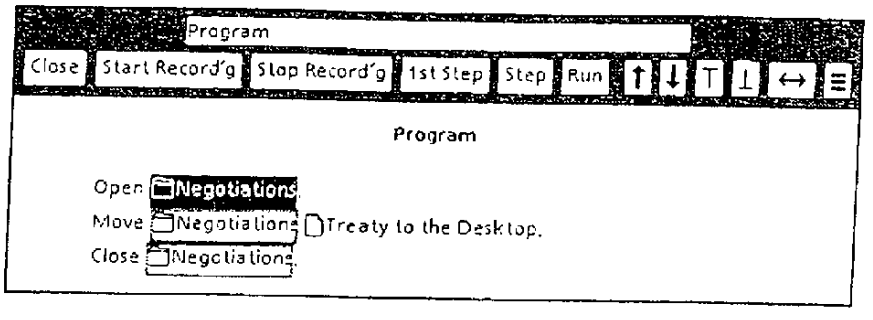


Figure 3. A program to move a named document out of a folder.

図1.4-10　SmallStarにおけるデスクトップと簡単なプログラムの例（出典：[Halbert93]）

図1.4-11 SmallStarにおけるメールプログラムとファイル移動プログラム（出典：[Halbert93]）

Figure 9. A mailing form.

Figure 10. The mailing form program, before the addition of the conditional.

Figure 11. The mailing form program.

Figure 6. Program to move any document out of a folder.

Figure 7. Adding a set iteration loop to a program.

Figure 8. Program to move all the documents out of a folder.

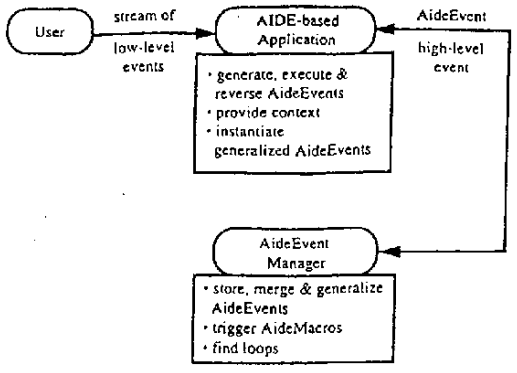


Figure 1. The Architecture of the AideWorkbench.

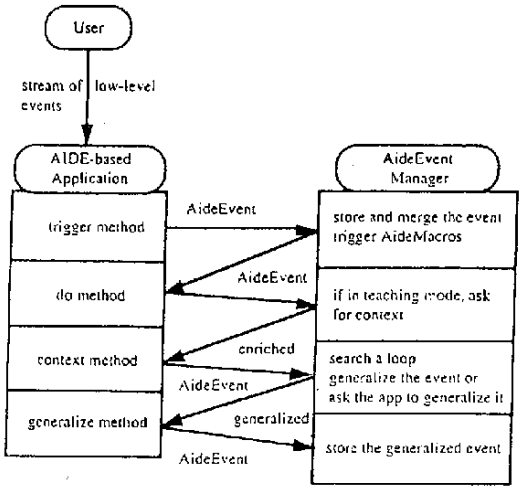


Figure 2. The dialog between an AIDE-based application and the AideEvent manager in teaching mode.

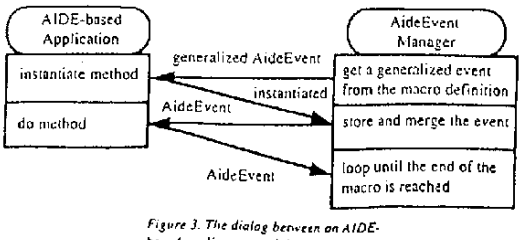


Figure 3. The dialog between an AIDE-based application and the AideEvent manager in macro execution mode.

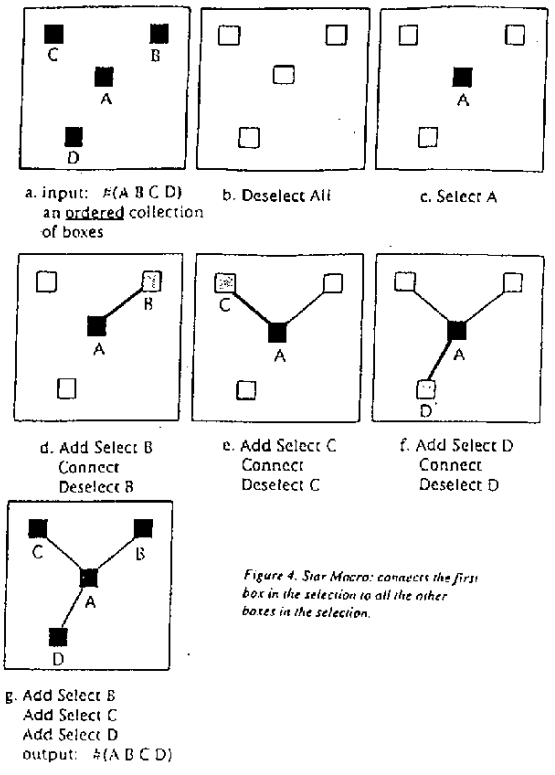


Figure 4. Star Macro: connects the first box in the selection to all the other boxes in the selection.

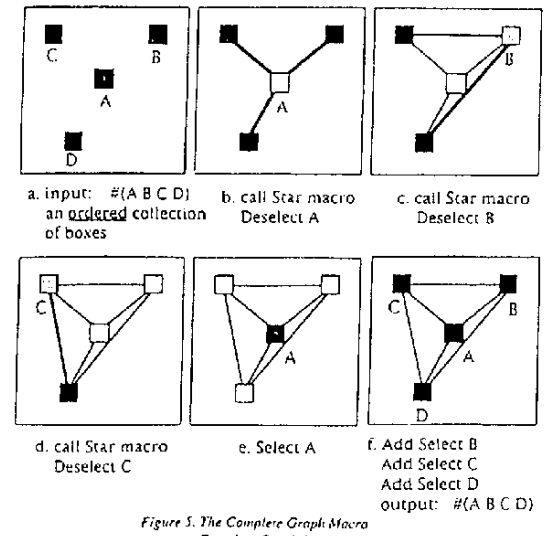


Figure 5. The Complete Graph Macro creates a Complete Graph from a set of boxes.

図1.4-12　AIDEのアーキテクチャとアプリケーションとの通信法及びマクロ生成の例

（出典：[Perinot93]）

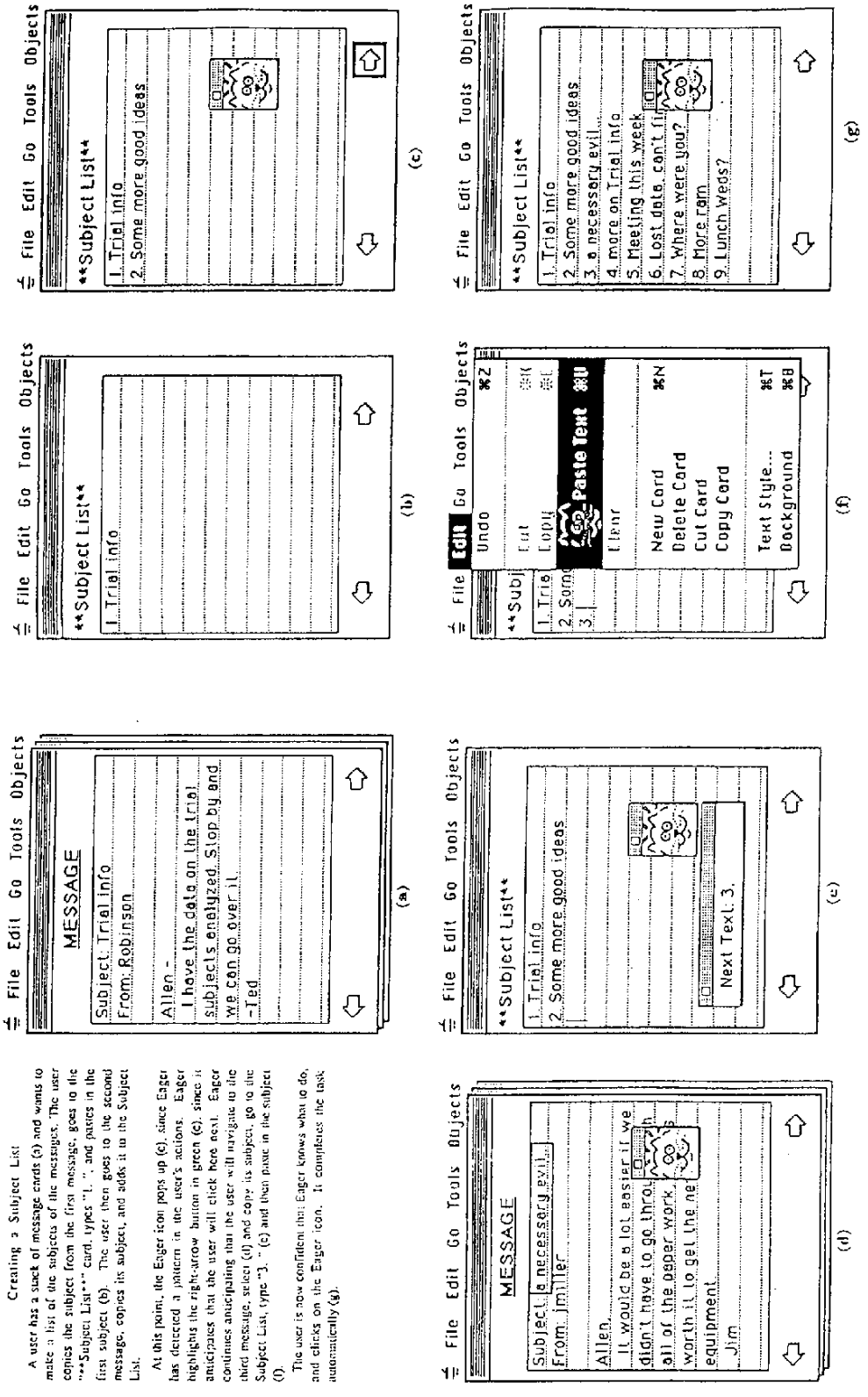


Figure 1.

図1.4-13　EAGERでの「メッセージカードの群からその主題のリストのカードを作る」プログラムの推論の例（出典：[Cypher91]）

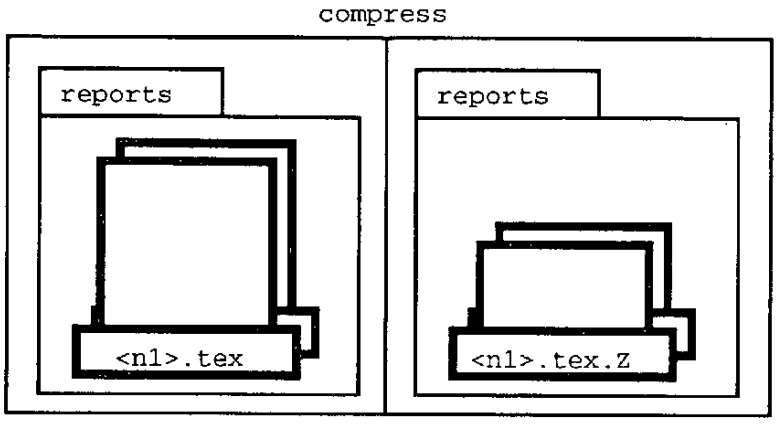


図1.4-14　Pursuit における圧縮の視覚的表現

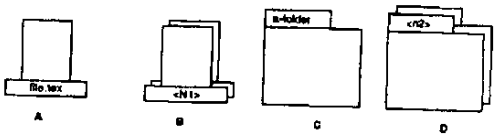


Figure 1: The main data types in Pursuit: a) a file; b) a set of files; c) a folder; d) a set of folders.

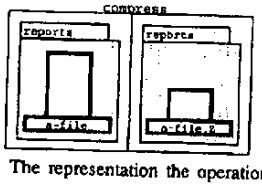


Figure 2: The representation the operation `compress a-file`.

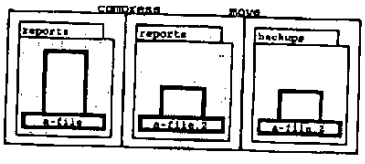


Figure 3: The visual representation of a program to compress the file `a-file` and then move it to the backups folder.

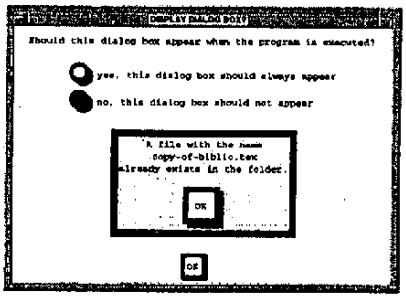


Figure 8: The Pursuit "meta" dialog box displayed when the user clicks on the dialog box icon in Figure 7. The user has indicated that the inner dialog box should not be displayed if the copy operation fails during program execution.

Figure 10: A Pursuit dialog box asking the user to confirm the set to loop over. The icon for the set of files is the same icon found in the third panel of Figure 6. The dialog box appears when the user executes the `rename` operation on two members of the set representing the copied files.

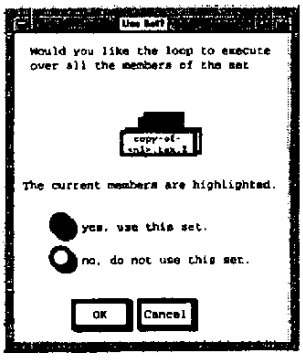


Figure 9: The Pursuit script that copies each `*.tex` file in the `papers` folder. If the copy operation fails because of the existence of a file with the output file name, the program deletes that old output file, and re-executes the copy operation. Users can see the other possible outcomes of the `copy` operation by clicking on the conditional marker.

図1.4-15　Pursuitの視覚的なプログラムの例。条件分岐を汎化によって生成するが、ユーザも直接編集可能。ダイアログ入力を扱うメタダイアログボックス等もある。(出典：[Mondugno94])

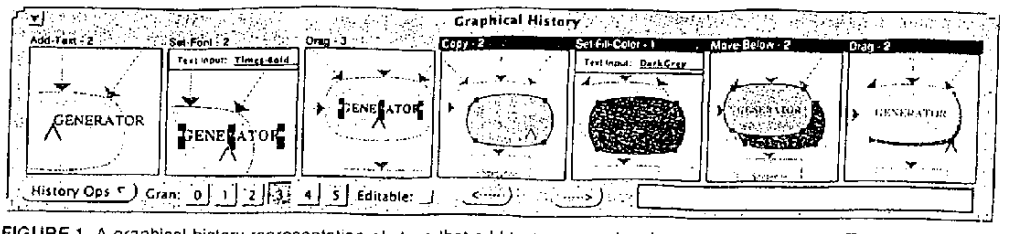


FIGURE 1. A graphical history representation of steps that add text to an oval and create a drop shadow. These steps were used in creating part of Figure 2. Panels whose labels are shown in reverse video have been selected by the user to create the macro shown in Figure 3.

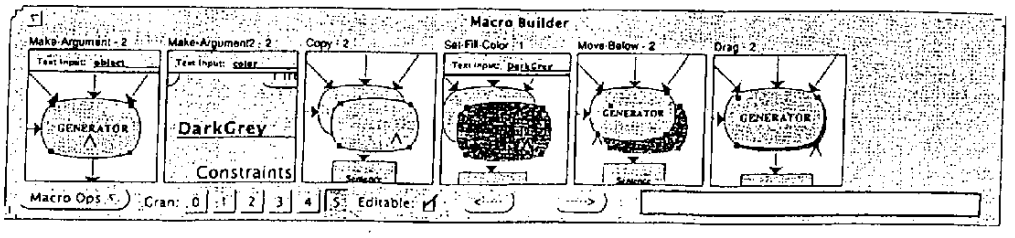


FIGURE 3. Macro Builder window containing operations to add a drop shadow to an object.

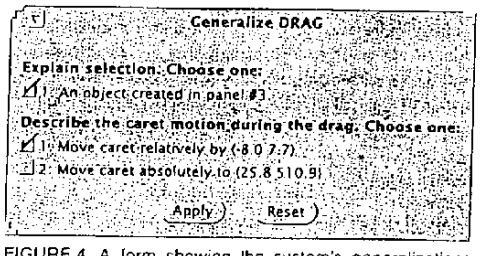


FIGURE 4. A form showing the system's generalizations for the last panel of Figure 3.

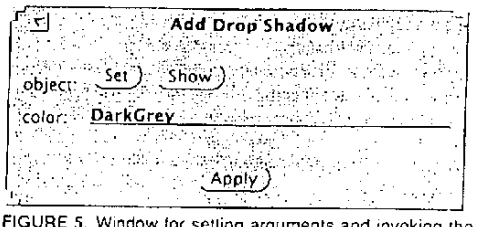


FIGURE 5. Window for setting arguments and invoking the drop shadow macro.

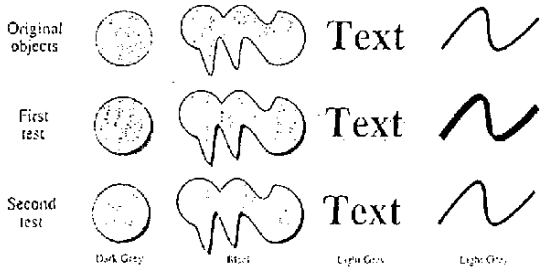


FIGURE 6. Testing the macro. The first row contains a set of test objects. The next row contains the results of invoking the original macro on these objects, using the colors named at the bottoms of the columns. The final row shows the results of invoking the debugged version of the macro.

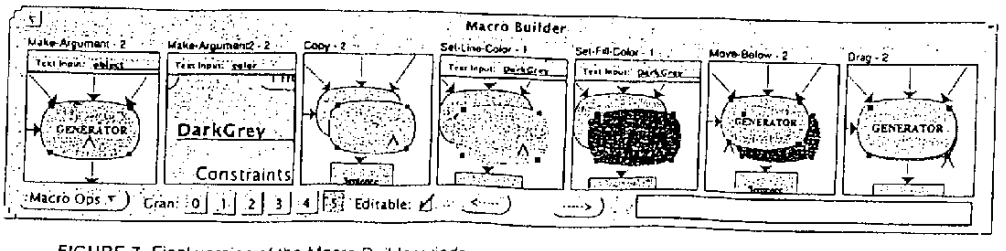


FIGURE 7. Final version of the Macro Builder window containing operations to add a drop shadow to an object.

図1.4-16　Chimeraでの「影付けマクロ」のgraphical historyの編集と汎化法の指定による生成

（出典：[Kurlander92]）

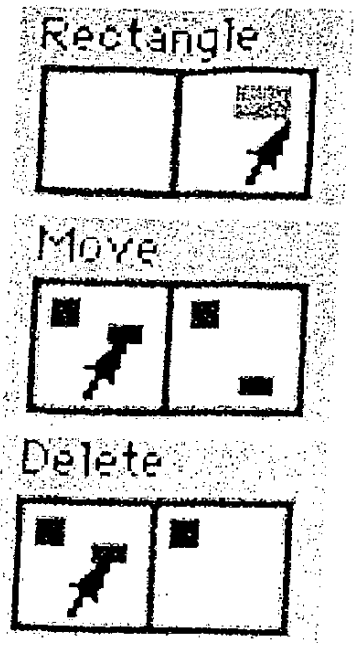


Figure 4. Some of Mondrian's domino icons

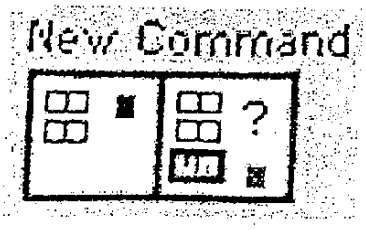


Figure 6. New command icon

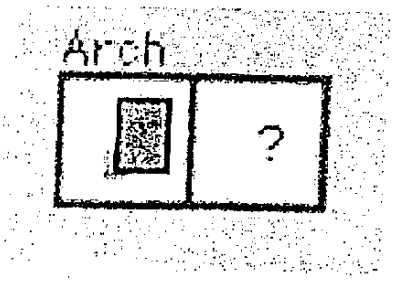


Figure 7. Arch icon just after the start of the definition

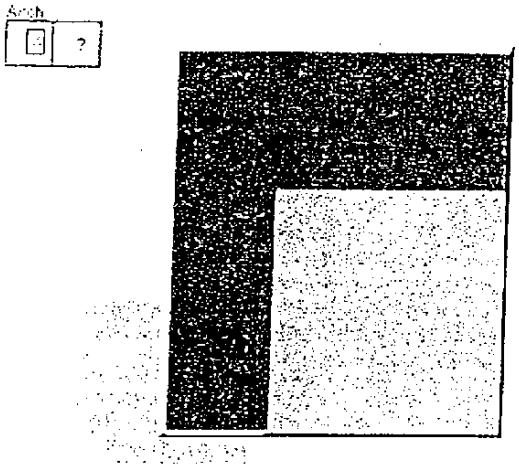


Figure 8. Drawing the Arch

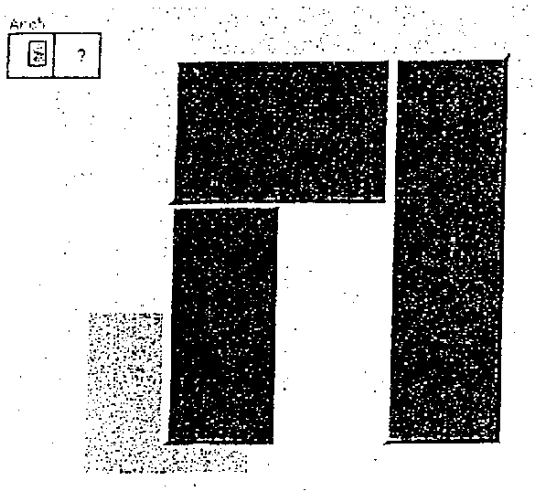


Figure 9. Selecting the three rectangles that make up the Arch

図1.4-17　Mondrianにおける"Domino"とアーチ生成のプログラムの例(1)（出典：[Lieberman92]）

Figure 10. Final version of the Arch icon

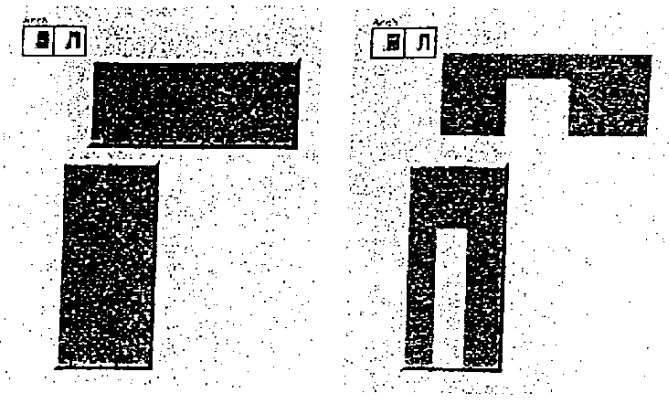


Figure 11. Before and after applying the Arch operation

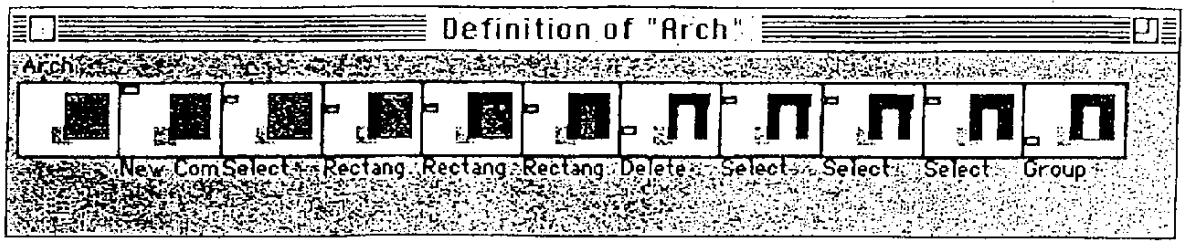


Figure 12. Storyboard for the Arch

```
(DEFUN ARCH (INTERACTOR SELECTION)
  (RECTANGLE-COMMAND (LEFT-TOP (NTH 0 SELECTION))
                     (POINT-ON-OBJECT (NTH 0 SELECTION) 12/95 1)
                     INTERACTOR
                     'RECTANGLE-5)
  (RECTANGLE-COMMAND (LEFT-TOP (FIND-OBJECT-NAMED INTERACTOR 'RECTANGLE-5))
                     (POINT-ON-OBJECT (NTH 0 SELECTION) 93/95 8/51)
                     INTERACTOR
                     'RECTANGLE-6)
  (RECTANGLE-COMMAND (POINT-ON-OBJECT (FIND-OBJECT-NAMED  INTERACTOR
                                      'RECTANGLE-6)
                                      76/91
                                      1/14)
                     (RIGHT-BOTTOM (NTH 0 SELECTION))
                     INTERACTOR
                     'RECTANGLE-7)
  (REMOVE-OBJECT INTERACTOR (NTH 0 SELECTION))
  (GROUP-COMMAND (LIST (FIND-OBJECT-NAMED INTERACTOR 'RECTANGLE-5)
                       (FIND-OBJECT-NAMED INTERACTOR 'RECTANGLE-6)
                       (FIND-OBJECT-NAMED INTERACTOR 'RECTANGLE-7))
                 INTERACTOR
                 'GROUP-8))
```

図1.4-18　Mondrianにおけるアーチ生成のプログラムの例(2)とStoryboard、及び汎化生成されたLispのプログラム（出典：[Lieberman92]）

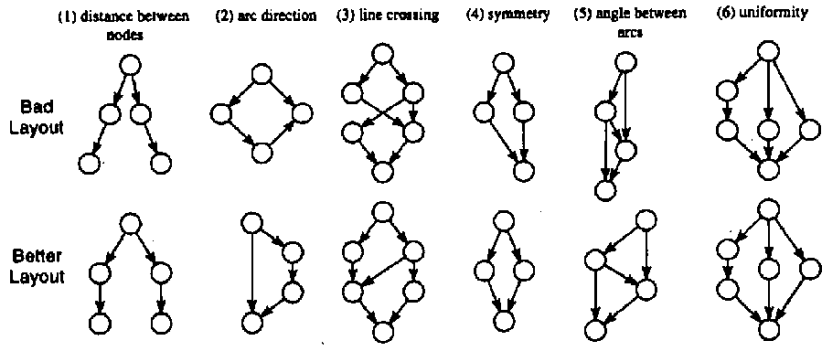


Figure 1: Constraints used in the layout of directed graphs.

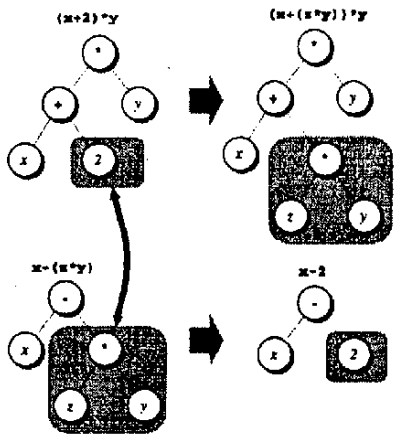


Figure 4: Crossover operation.

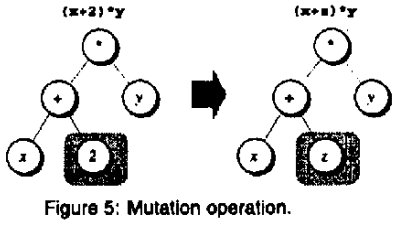


Figure 5: Mutation operation.

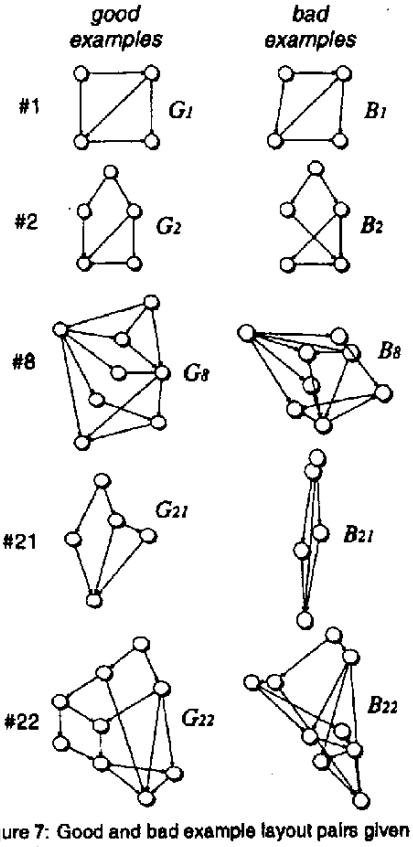


Figure 7: Good and bad example layout pairs given to the system.

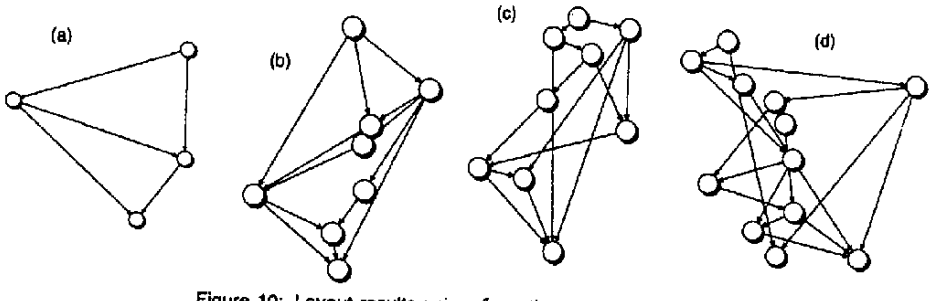
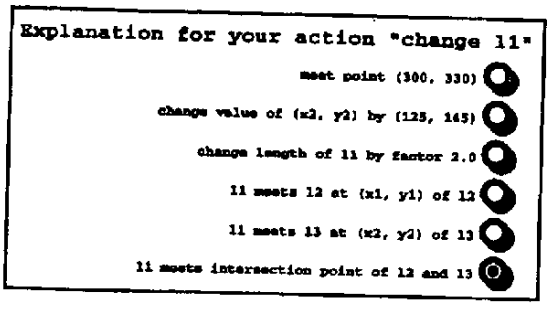


Figure 10: Layout results using f_6 as the evaluation function.

図1.4-19　Genetic Programmingによるグラフレイアウト、あるトポロジに対する一組のレイアウトに対し、人間がどちらがよいかを指示、GPによって学習する。(出典：[Masui94])

Figure 2: Enlarging l_1 to meet l_3 might have various explanations



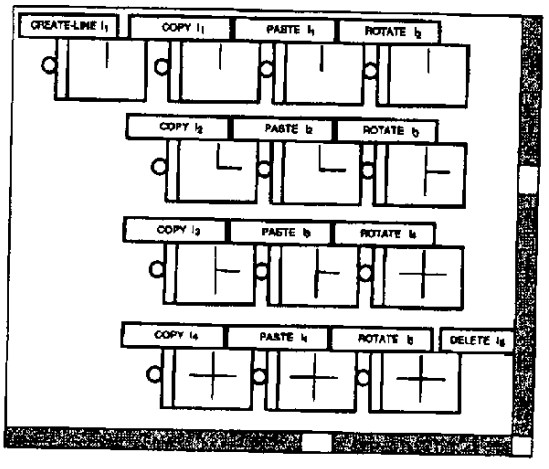


Figure 6: two-dimensional History Board

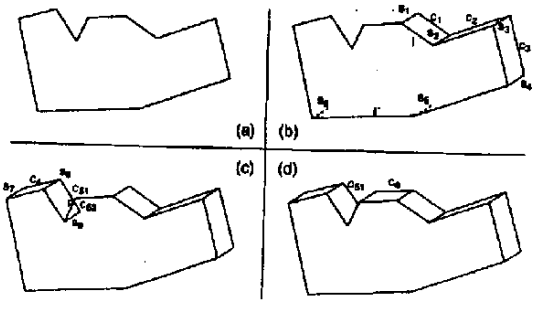


Figure 8: 3–dimensional extension of an object

図1.4-20　ProDeGE+システム。ユーザ動作の意図の多様な可能性から選択を要求する。生成されたマクロは、2次元上のStoryboardにより編集可能。2Dの多角形オブジェクトを3Dに変換するような複雑なマクロも生成可能とされている。(出典：[Sassin94])

2.　マルチモーダルインタフェース

2. マルチモーダルインタフェース

2.1 マルチモーダルインタフェースの現状と展望

2.1.1 はじめに

コンピュータと人とのインタフェースを考えると、現在、最も盛んなのは視覚を用いたインタフェース、すなわち、ビジュアルインタフェースである。ビジュアルインタフェースの元になるGUI（Graphical User Interface）が最初に作られて、ほぼ四半世紀が経過した。そろそろ、ビジュアルインタフェースに代わる次世代インタフェースを真剣に研究開発してもよいのではないだろうか。そこで、ここでは、ビジュアルインタフェースの次のインタフェースの一つの候補として、最近注目されているマルチモーダルインタフェースを取り上げる。

ここでは、最初に、ビジュアルインタフェースの持つ意味と問題点を述べた後、マルチモーダルインタフェースのアプローチとは何かを述べ、次にいくつかの具体例を交えてマルチモーダルインタフェースの現状を紹介する。最後に、マルチモーダルインタフェースの今後の課題と方向性を述べて締めくくる。

2.1.2 ビジュアルの意味と問題点

現在、GUI、3D視覚化等、ビジュアルインタフェースの全盛期といってもいいほど、視覚的なインタフェース、すなわち、ビジュアルインタフェースが用いられている。これは、それ以前のコマンドラインインタフェースと比較するとその特徴と意義がはっきりする。つまり、ビジュアルインタフェースでは、

- 今まで隠されていた情報、データ等が見えるようになる（可視性）。
- 処理するコマンドなどを記憶再生する代わりに、メニューに現れた中から再認することにより記憶の負荷が減る（記憶の外在化）。
- 操作対象物に相当するものが画面に表示され、それを直接操作できる（直接操作）。

といったメリットがあり、特に初心者にとって分かりやすく使いやすいインタフェースである。一方、コマンドラインインタフェースに比べると、

- マウス操作が伴うのでキーボードからの入力に比べて遅い。
- システムが大きくなる。

といった問題点もある。

さて、人間の認知という観点からは、従来のコマンドラインインタフェースが記号化・符号化されたコマンドでのやり取りだったのに対して、ビジュアルインタフェースでは、操作対象と処理内容を視覚的に表示して、それに対して直接やり取りする点が際だっている。すなわち、人の視覚認知とい

う特性をうまく使っているわけである。その応用としてデスクトップメタファなどがある。

人間の認知の観点からの問題としては、現状のビジュアルインタフェースが視覚偏重である、すなわち視覚のみに頼り過剰に使用している点が指摘されている。

したがって、ビジュアルインタフェースの次のインタフェースとしては、視覚以外にも複数のモダリティを用いたマルチモーダルインタフェースが有望な候補として考えられる。

2.1.3 マルチモーダルインタフェースのアプローチ

マルチモーダルインタフェースに特徴的な要素を述べると次のとおりである。

1. 身振りや音声など、視覚以外のモダリティを用いる。
2. 上記の目的のために、コンピュータ側が目や耳、口といった能動的な機能も持つ。
3. 従来の文字情報だけでは、表わしきれない感情や気持ちといった二次情報の伝達も可能にする。
4. 各モダリティの役割を明確にして、それぞれのモダリティに合った使い方をする。
5 複数のモダリティの複合化、連動化を意識的に行う。
6. モダリティ間の変換を支援していく。

感覚器でいえば、従来は手の動き（キーボードとマウス）以外は視覚しか用いなかったが、マルチモーダルインタフェースにおいては、音声を意識的に用いる。他に、体や手や腕などの身体運動もコミュニケーションの手段とする。他の五感では触覚をできるだけ利用したいが、難しい問題がまだ多い。味覚や嗅覚はかなり先の課題になる。

コンピュータ側に目や耳、口といった機能を持たせることにより、コンピュータがより能動的な存在になる。これはコンピュータにセンサだけでなくアクチュエータとしての機能を持たせ、いわばロボット化した存在として捉えると分りやすい。

音声や体の動きなどを用いることにより、感情表現や意思表示などが容易になる。これは、従来のインタフェースにおいては捨て去っていた部分である。

複数のモダリティを用いた場合に、一つのモダリティですべてのモダリティの代わりをするということではなく、モダリティごとの得意、不得意によって使い分けをすることが重要になる。例えば、音声タイプライタなどのように音声をタイプライタ代わりにするのは意味がないが、音声が必要な状況で利用することが大事なのである。

さらに、複数のモダリティを持たせることが伝達の冗長性をもたらすばかりでなく、一つのモダリティから別のモダリティへの変換を支援することにより、例えば、障害者や高齢者向きのインタフェースへの応用の道が自ずと開けてくる。

2.1.4　マルチモーダルの具体例

(1)　音声と音の利用

マルチモーダルシステムにおいては、視覚を除いて最初に注目すべきは音声や音の利用である。まず、音声については、単語レベルの音声認識は特定話者で単語数が限定されたものは実用化されている。現在、不特定話者や連続音声認識が研究・試作段階にある。音声認識は日進月歩の進展を遂げているが、音声認識の問題点は実際的な応用と無関係に今まで研究が行われてきたことにあるということが、音声認識の関係者自身の反省の弁として語られている。その意味でも、マルチモーダルインタフェースの一部として音声認識をしっかり位置付け、実際的な応用場面を想定した研究開発を進める必要があるだろう。

音声認識と並んで、音声対話で必要となるのは音声合成であるが、こちらは、現在、漢字仮名混じり文を読み上げる能力を持っている。アクセントやイントネーションがややおかしいとか、女性の声が合成しにくいといった問題があるが、機械が出す声と思えば実用的に使うことも可能である。実際、音声合成は、打ち込み原稿の確認や視覚障害者のインタフェースなどの面で実用的に使われている。

音声が文字などに比べて持つ特徴としては、次のようなことが挙げられる（一部、音声を含む音全般にもいえる)。

・キーボードなどと比べて非接触型のインタラクションが可能。
・符号化されたボタン式などと比べて自然言語に近い言葉で指示可能。
・ディスプレイと違いずっと見続けていなくてもよい。
・警告などのときに、より注意を喚起しやすい。
・揮発性も持つため繰り返しが必要なことがある。

このような特性を考慮して音声を利用できる場面をさらに考えていく必要がある。

音に関しては、音声と共通の特徴とそれ以外の面がある。音固有のものとして、例えば、聴覚化（auralization）がある。これは、視覚化（visualization）と同様、情報やデータなどを音情報として表現する方法であるが、現在、並列プログラミングの聴覚化[Kazama94]、GUIの聴覚化などその可能性を探るべく盛んに研究され始めている。

(2)　身振りの認識

身振りをコンピュータの入力に用いようとする研究開発は、例えば、仮想現実（Virtual Reality）においても行われてきた。この場合、データグローブやHMD（ヘッドマウントディスプレイ）を装着して位置検出するのが普通である。マルチモーダルインタフェースにおけるジェスチャ入力においては、このような特殊な機器を体に装着することなく、自然な体の動きを画像認識により認識し、コンピュータ入力することを特徴とする。

ジェスチャ入力の研究の一例として、MAIの例[Yasumura94]を紹介する。これはシリコングラフィックス社のINDYを用いて、秒15フレームで人物画像を動的に認識する。認識方法として背景差分法を用いている。メッシュは30×30の大きさである。この方式で手や頭の動きを実時間で認識可能である。

次に、その具体的なデモンストレーションを示す。例えば、次のようなジェスチャ入力が可能である。

1. 仮想楽器の演奏とフィードバック（図2.1-1、注：図はいずれも本節末に掲載）
2. 複数ユーザによる仮想テニス（図2.1-2）
3. スーパーボールと創発的な面白さ（図2.1-3）
4. お絵描きツールのためのドラッグ機能（図2.1-4）
5. ハンドジェスチャによるプレゼンテーション（図2.1-5）

ここで、いずれも音とグラフィックス上でのフィードバックが、非常に重要な役割を果たしている点を指摘しておきたい。特に、音を出す仮想楽器の場合には映像上での変化が、また、テニスやスーパーボールなどでは当たった瞬間の音によるフィードバックが重要であるとの指摘がなされている。スーパーボールは、電子空間内の仮想オブジェクトが人間に当たって跳ね返るだけであるが、動画で見てもらえば分かると思うが、予想以上の面白さ（＝創発性）を含んでいる。オブジェクトの選択とドラッグは、ジェスチャ入力の実用化を目指した研究であるが、工夫次第ではマウスなどに代わる入力手段になり得る可能性を示している。

(3) その他の研究例

マルチモーダルインタフェースのその他の研究例としては、顔の表情が変化するコンピュータエージェントと人（ユーザ）とのインタラクションを追求する研究[Naito94]などもある。これなどはマルチモーダルインタフェースとエージェント（これも、次世代インタフェースの有力な候補）との両方に跨る研究として興味深い。

また、アートとしてマルチモーダルインタフェースを捉えると、マイロン・クルガーの人工現実の研究[Krueger91]やビンセント・ジョン＝ビンセントのマンダラシステム[Sakane89]などがある。

2.1.5 今後の課題と方向性

マルチモーダルインタフェースは、それ単独でも研究の意味は十分ある。例えば、音声を用いたコマンドシステム（音声シェルなど）は、これから続々実用化されていくだろう。また、ジェスチャ入力なども、適切な利用場面を設定すれば、研究開発により一層はずみがつくだろう。

しかしながら、マルチモーダルがこれから大きく伸びていくのはその単独の技術というよりも、他

の関連技術や関連分野との関係においてである。

その意味で、これからマルチモーダルインタフェースの研究において忘れてはならないのは、次の研究パラダイムとの関連性である。

1. だれでもが使える（障害者や高齢者にも）
2. どこでも使える（ユビキタス性）
3. 代理人として使える（エージェント性）
4. 離れていても使える（ネットワーク性）
5. 現実世界に意味を持つ（双対空間性）

第一は、いうまでもないことであるが、マルチモーダルの中でもモーダル変換の技術が役に立つ。次は、コンピュータをあからさまな形で人の前に置くのではなく、日常物の中に埋め込み隠してしまうことである。第三については、音声や身振りが意味を持つのが人間に近いエージェントに対してであることから納得できるであろう。さらに、ネットワークを利用した遠隔地点間のコミュニケーションは、マルチモーダルの場合にも避けて通れない。最後に、マルチモーダルを役に立てたいのは、仮想空間だけではなく、仮想と現実の二重写しになった双対空間であることの理解が重要である[Wellner93]。

2.1.6 おわりに

マルチモーダルインタフェースが、本当にビジュアルインタフェースに代わる次世代インタフェースになり得るかは、だれか偉い人の予言を待つような話ではなく、実際にマルチモーダルインタフェースの名の下に本当に意味のある研究開発ができるかどうかにかかっている。

その意味でも、一人でも多くの人がこのテーマに興味を持ち、研究に参加してくれることを待ち望むものである。

【参考文献】

[Access93] 安村通晃編, アクセス研究会, 障害者向けコンピュータ・インターフェイスへの序奏, 慶應義塾大学環境情報研究所 KEIO-IEI-RM-93-004, 1993.

[Allen80] Allen, J., and Perrault, C. R., Analyzing Intention in Utterances, Artificial Intelligence, Vol.15, pp.143–178, 1980.

[Aukstakalnis92] Aukstakalni, S., and Blatner, D., Silicon Mirage – The Art and Science of Virtual Reality –, Peachpit Press, 1992. 邦訳: 安村・伊賀・織畑共訳, シリコン・ミラージュ, トッパン, 1994.

[Blattner92] Blattner, M. M., and Dannenberg, R. B., eds, Multimdeia Interface Design, ACM Press, 1992.

[Carberry90] Carberry, S., Plan Recognition in Natural Language Dialogue, MIT Press, 1990.

[Harashima94] 原島・廣瀬・下條編, 仮想現実学への序曲, 共立出版, 1994.

[Kay92] Kay, A., 鶴岡雄二訳, 浜野保樹監修, アラン・ケイ, アスキー出版, 1992.

[Kazama94] 風間・佐藤, 並列プログラムの聴覚化, 日本ソフトウェア科学会 WISS'94, 1994.

[Krueger91] Krueger, M. W., Artifitial Reality II, Addison-Wesley, 1991. 邦訳: 下野隆生, 人工現実 ― インタラクティブ・メディアの展開―, トッパン, 1991.

[Laurel90] Laurel, B., Ed, The Art of Human-Computer Interface Design, Addison-Wesley, 1990. 邦訳: 人間のためのコンピューター, アジソンウェズレイ, 1994.

[Naito94] 内藤・竹内, Situated Interface: 社会的インタラクションに向けて, 日本ソフトウェア科学会 WISS'94, 1994.

[Sasaki87] 佐々木正人, からだ: 認識の原点, 東京大学出版会, 1987.

[Sakane89] 坂根巌夫監修, 不思議の国のサイエンスアート展図録, 1989.

[Sato94] 佐藤宏之, 伊賀聡一郎, 安村通晃, 遠隔デスクトッププレゼンテーションシステムのデザイン, 計測自動制御学会ヒューマンインタフェースシンポジウム94, 1994.

[Shneiderman92] Shneiderman, B., Designing the User Interface, 2nd ed., Addison-Wesley, 1992.

[Tanaka94] 田中公二, 伊賀聡一郎, 岡部 学, 安村通晃, マルチメディア語学学習環境の開発と評価, 情報処理学会 コンピュータと教育研究会, 33-4, 1994.

[Visual94] ビジュアルインタフェース調査研究WG, ビジュアルインタフェースの研究開発報告書, 日本情報処理開発協会, 1994.

[Yasumura93] 安村通晃, 伊賀聡一郎, マルチメディアからマルチモーダルへ, 日本ソフトウェア科学会 WISS'93, 1993.

[Yasumura94] 安村通晃, 今野 潤, 八木正紀, マルチモーダルプラットフォームMAIの構築に向けて, 日本ソフトウェア科学会 WISS'94, 1994.

[Wellner93] Wellner, P., et.al. Eds, Computer-Augmented Environments: Back to the Real World, Comm. ACM, Vol.36, No.7, 1993.

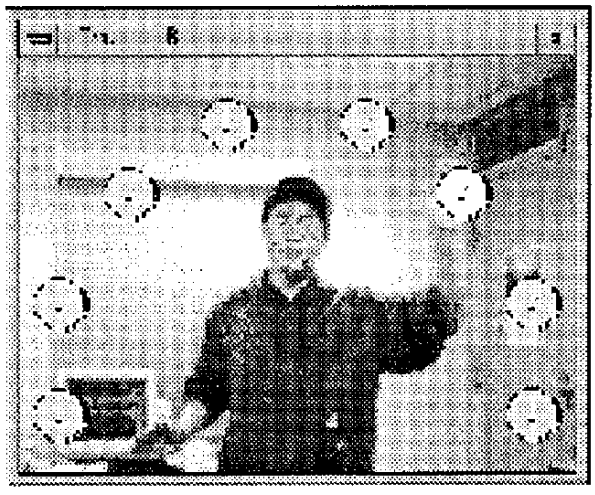


図2.1-1　仮想楽器の演奏と視覚的フィードバック

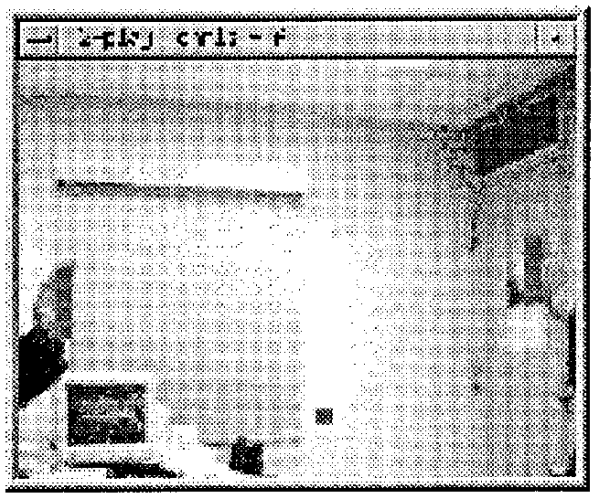


図2.1-2　ふたりで楽しむ仮想テニス

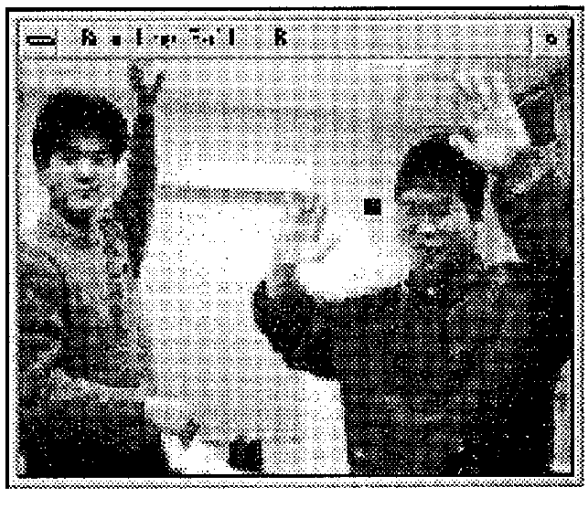


図2.1-3　スーパーボールでの創発性

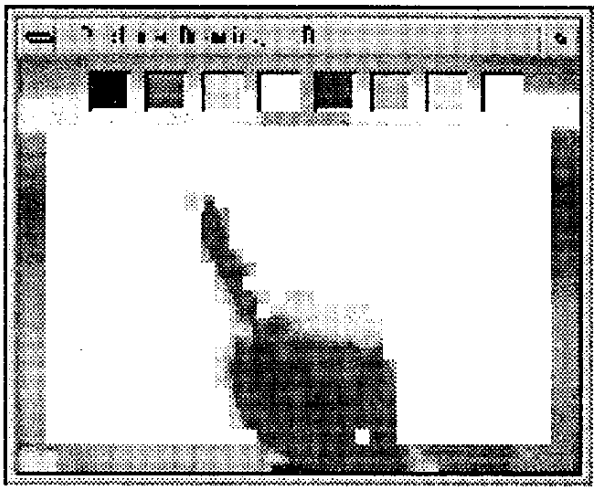


図2.1-4　お絵描きツールのためのドラッグ機能

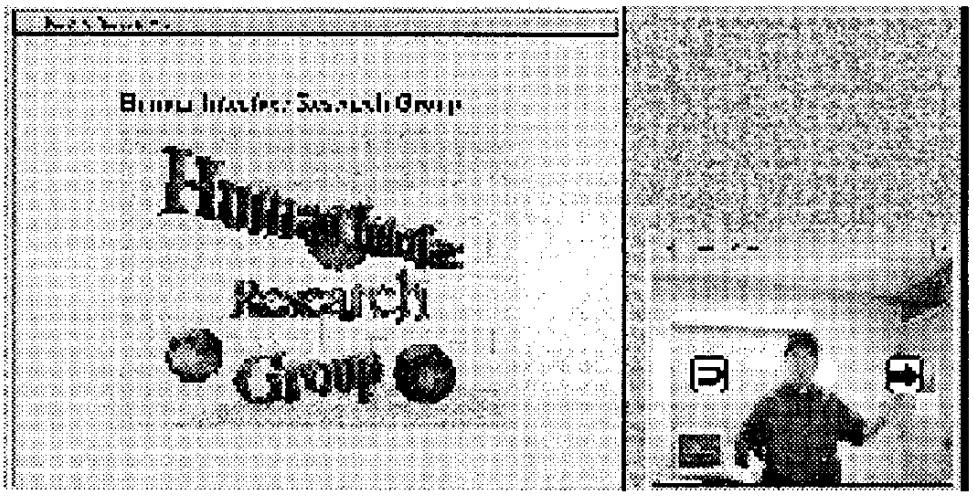


図2.1-5　ハンドジェスチャによるプレゼンテーション

2.2 マルチモーダルインタラクションと社会性

2.2.1 はじめに

コンピュータと人間とのインタラクションを設計する際、二つの理想のモデルがある。一つは人間どうしの向かい合っての（face-to-face）会話であり、他はコンピュータをよく馴染んだ道具とするモデルである。向かい合っての会話の一つの重要な特徴はチャネルの多重性である。チャネルとは特定の符号化方式を持つコミュニケーション媒体である。例えば、音チャネル、画像チャネルはそれぞれ言葉、表情を運ぶチャネルである。モダリティとは外界から信号を受け取るのに使われる感覚である。一つのチャネルは複数のモダリティで感知されることもある。視覚、聴覚、触覚などはモダリティの例である[Blattner92]。

マルチモーダルインタラクションは、複数のモダリティに作用する複数のチャネルを持ったインタラクションと定義できよう。理想的なマルチモーダルインタラクションを実現するためには人間がどのように情報を獲得するか、また、どのような情報に人間はセンシティブであるかについて研究する必要がある。

通常の人間どうしの向かい合っての会話においてはたくさんのチャネルが使われ、たくさんのモダリティが活性化される。会話は様々な認知レベルに属する多数のアクティビティに支えられている。例えば、音声のシンタクティックな処理とセマンティックな処理とは連携しているし、オブジェクトレベルの処理（会話のゴールに密着）とメタレベルの処理（会話の調整に関与）とは並行して行われている。結果として会話は高度に柔軟で頑健なものとなり、一つのチャネルの失敗を他のチャネルで取り戻したり、あるチャネルのメッセージを別のチャネルが説明したりするようになる。

マルチモーダルの例を挙げてみよう。例えば、部屋にある何かを指し示すのに、いろいろな方法がある。指さす方法、視線で示す方法、言葉で記述する方法、頭など手以外の部位で示す方法などである。どれか一つの方法だけで指示するよりも、複数使う方が曖昧さが少なくて分かりやすい。また、ボールの投げ方を説明するのに言葉だけでなく身振りを交えるが、これは投げるという動作を教えるのに、どちらも片方だけでは不十分な説明しかできないものが相互に補足しあう例である。また、一枚の絵を前にして「これはいい絵だ」と冷笑しつつ言えば、これは単に「下手な絵だ」という以上の何かを暗示している。これは言葉と表情のいわば対立技法とでも言うべきもので、これにより意味の明暗をよりくっきりと浮かび上がらせている。言語的なコミュニケーションにおいては言葉が主要な役割を果たすことは間違いないが、そのような場合でも複数のモダリティが相互に作用して言語的意味を修飾し、ときには意味を反転させることもある。

複数のモダリティを駆使する場合、様々な連携の仕方があるが、一つだけ共通しているのは同期である。すなわち、複数のモダリティで意味をめぐって干渉しあうのは、同じ時間にそれぞれがサインを出しているときに限る。時間が大きくずれたモダリティ間の干渉はない。コンピュータでマルチモ

ーダルインタラクションを実現する際にまず問題になるのは、この同期の問題である。コンピュータがマルチモーダルなメッセージを生成する場合は、複数の出力装置（例えば、映像と音声）の間で同期を保証するといった問題になる。また、コンピュータがマルチモーダルなメッセージを解釈する場合の問題としては、映像中に現れたキューと音声に現れたキューとを時間軸で重ねて解釈するというメカニズムが必要になる。

本節では、特定のチャネル／モダリティ、すなわち、表情に焦点を当てつつマルチモーダルインタラクションについて考える。人間は進化の過程で表情を用いた洗練された会話形態を発展させてきた。会話的な表情の役割について知ることは、人間が複数のチャネル、モダリティを用いてどのようにして会話を制御しているか、すなわち、会話の力学を理解することにつながる。その知識を適用することにより、どんなに微妙な情報も洗練された会話的表情を用いて表現できるような新しいユーザインタフェースができるだろう。

face-to-faceという言葉やinterfaceという言葉が示唆するように、顔はコミュニケーションにおいて本質的な役割を果たしている。実際、「向かい合って（face-to-face）」のコミュニケーションというのは、霊長類から人類への長い歴史において進化してきた最も根源的なコミュニケーションのスタイルである。人間の大脳は、このような形態のコミュニケーションに適応していると見て間違いない。霊長類の大脳には、顔情報処理用の領域があることが神経生理学では知られている[Perret84]。このことは、顔情報の処理能力が進化の過程の自然淘汰で生き残る上での重要な要因だったことを示唆している。

表情は心理学、行動生物学など様々な分野で研究されてきている。表情研究には二つの代表的な立場がある。一つは表情を情緒との関連において見ようという立場であり[Ekman84]、他は表情を社会という文脈の中で捉え、それを会話的な信号として見る立場である[Fridlund85]。後者の研究者達は、表情を表わす言葉として従来のfacial expressionの代わりにfacial displayを用いている。こちらの方が情緒的ニュアンスが少ないからである。

本節では、以後後者の立場で議論を進める。後者の立場をとると、顔は情緒的情報と同様に会話的信号を表情に符号化して伝達する独立した会話チャネルである。大脳に表情処理の専用回路があることからも表情を一つのモダリティと考えることができよう。これらの考えに基づき、認知的負荷を軽減しつつ、コンピュータと人間のインタラクションをより緊密にかつより効率良くすることを目的に、表情を会話の新しいモダリティとして持ち込むことが可能である。

顔は人間にとって社会的なインタフェースである。人間は社会的動物である。表情は自分に向けたものではなく、他者に向けられたものである。表情は他者とのよりよい社会的関係を発展させるように進化してきたはずである。表情は本来会話的であり、それゆえ社会的な慣習などによりその表現が規制を受ける[Chovil89]。それゆえ、表情の研究は社会的なインタフェースの重要な特性を明らかに

するという期待もできる。

顔・表情を使ったインタフェースは、一般に擬人的インタフェースと呼ばれることがある。擬人的モデルは近年議論の多い話題である[Don92]。おもしろさだけを狙ってアドホックに設計された擬人的インタフェースもいくつか存在する。しかし、GUIDESのように注意深く設計されたインタフェースはユーザインタフェースの新しい可能性を示している[Don91]。

本節の構成は以下のようになっている。まず、2.2.2節で（株）ソニーコンピュータサイエンス研究所（以後、ソニーCSL）で開発中のシステムを例に、顔・表情の合成について説明し表情アニメーションの導入を行う。2.2.3節で会話的な表情についての研究を紹介し、合成された会話的な表情を音声対話システムと統合した実験システムを2.2.4節で紹介する。2.2.5節では、ビデオカメラを通して得た外界の状況に応じて表情を表示する表情アニメーションシステムを紹介する。ここでは特に視線の生成に焦点を当てている。コンピュータがマイクやビデオカメラにより聴覚・視覚を備え、外界を多少とも認識するようになるとどのような問題があるのか、あるいは、その後どのような展開が考えられるのか、これらについて2.2.6節で考える。

2.2.2 顔・表情を作る

ソニーCSLでは、表情豊かな顔のアニメーションの研究を進めている[Takeuchi92]。現在の焦点は、顔の3次元モデリングと表情合成である。ここでは、この過程で開発中のアニメーション実験環境について説明する。この環境は、任意の顔の3次元モデルを短期間に作り、それに様々な表情を簡単に与えることができるようにしており、これにより自然でかつ印象的な表情を探ることを目指している。この実験環境は以下の三つのサブシステムからなる。

① 3次元モデリングシステム

② 表情エディタ・アニメーションシステム

③ 筋肉エディタ

これらのシステム間の結合は弱く、ファイルによってデータを交換している。このため、各システムの独立性が高く、それがために各システムの構成を他のものに改造するといったことが容易にできるようになっている。以下では、各システムの概要について説明する。

(1) 3次元モデリングシステム

人は自分のよく知っている顔の表情に対してセンシティブに反応する。この心理的傾向をうまく利用するためには、日常的な媒体、例えば写真やビデオなどから、そこに写っている顔を3次元的にモデル化する技術が必要である。このため、一連の（撮影角度や被写体にマーカを付けておくといった制限のない）2次元顔画像から対応する3次元モデルを作るシステムを開発した。顔は3次元的にモデ

ル化されている。現在の顔は約500のポリゴンからなっている。システムは、初期モデルとして一つの3次元モデルをもってスタートし、2次元画像一枚ごとにそれを写真中の顔に合うように変形するという操作を繰り返し、徐々に似た顔に仕上げていく。具体的には、各2次元画像につき以下のことを行う。

(a) 顔を抽出：現在は、両目、鼻、顎の四点を人間が指定する。

(b) 顔の向きを推定：上の4点に対応するモデル上の4点の射影像が2次元画像中の4点にうまく重なるように、顔の回転（姿勢）、少数のスケーリングパラメータを推定する。

(c) 顔の変形①：上で求めたスケーリングをモデルに施し、大まかな変形をする。

(d) 顔の変形②：さらに上で求めた回転もモデルに施し、2次元画像と重ね合わせ、輪郭のずれを補正する。ここで使う手法は、Terzopoulosらの方法である。すなわち、モデルを3次元のアクティブネットとし、画像から得られるイメージポテンシャルの下で力学的な平衡解を求める。

(2) 表情エディタ・アニメーションシステム

コンピュータグラフィクスでは、表情は顔を構成するポリゴンの局所的変形として実現される。Watersは顔の筋肉をシミュレートすることにより、より自然な表情が生成できることを示した[Waters87]。ソニーCSLのシステムでも、このWatersにより定義された筋肉の動きをシミュレートする方程式を使っている。解剖学によれば、顔には様々な収縮特性を持つ筋肉が数百本あるとされている。現在16本の筋肉を扱っているが、これらはWatersがFacial Action Coding System（FACS）[Ekman78]との対応を考えながら定義したものである。この他に10個のパラメータを持つ。これらは、口の開閉、顎の上下、目の動き、まぶたの開閉、首の向きを制御する。

アニメーションはフレームごとに各筋肉の緊張の度合いを指定し、表情を作ることにより行われる。現在、各筋肉の動きは、スプライン曲線で書かれたコードにより滑らかにコントロールすることができる。EkmanらのFACSが表情の静的な記述であるのに対し、ソニーCSLの記述は動的であり、そのため様々な微妙な表情表現が可能になった。また、各筋肉の動きを、スライダでコントロールすることもできる。

表情の作成、編集のために、対話的なスプライン曲線のエディタが用意されている。

また、短期間の表情に対応したシーンという概念を持ち、これをいくつかつないで比較的長い期間の表情変化を容易に作成することができるようにもなっている。

3次元モデリングシステムは顔の向きを考慮しつつ、モデルによくフィットする顔画像を切り出すことができる。この2次元画像をテクスチャとして3次元のモデルにマップすることができる。これにより、元の被写体にかなりよく似た顔が得られる。また、3次元の顔にグローシェーディングを与えることもできる。

性能に関しては、現在、SGI power seriesでリアルタイムでアニメーションできている。

図2.2-1に顔モデラと表情アニメーションシステムの関係を示す。

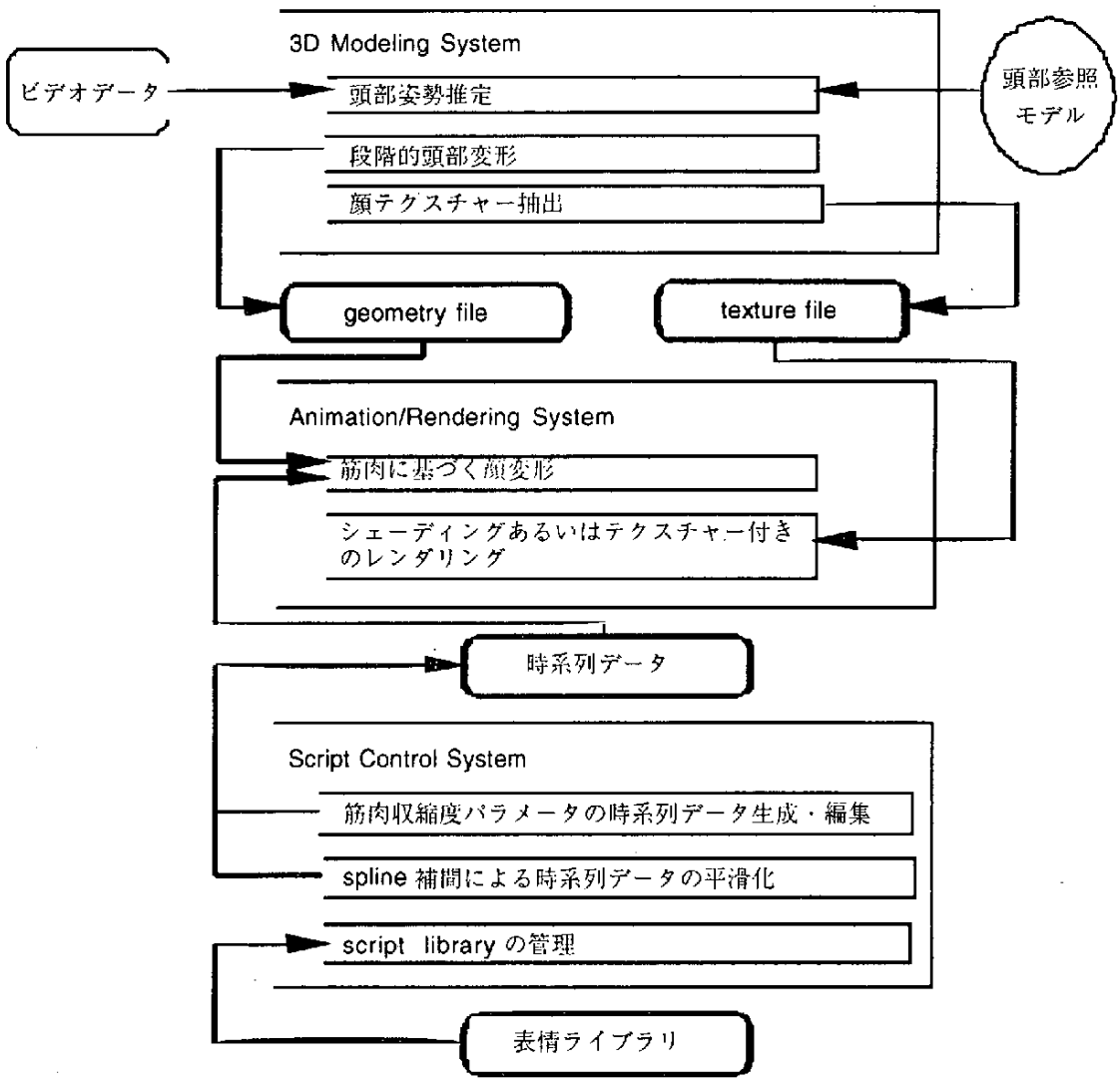


図2.2-1　顔／表情アニメーションシステムの構成

(3)　筋肉エディタ

解剖学によれば、顔には様々な収縮特性を持つ筋肉が数百本あるとされている。表情はこれらの筋肉のすべてもしくは一部が緊張し、それに応じた顔面の収縮が起きることにより作られる。したがって、顔の形が同じでも、皮膚の下の筋肉のセッティングが異なれば違った表情になる。望みの表情を合成しようとすれば、顔形状の設計や筋肉の収縮率の時系列の設定だけでなく、筋肉の設置条件の修正・筋肉の追加・削除などが必要になる。この目的のため、筋肉エディタという筋肉の設置条件を

様々に変更できるエディタが開発されている[内藤93a, 内藤93b]。

2.2.3 会話的な表情

表情は既に長い間科学的研究の対象となってきた。Darwinが表情の二つの面として情緒に関係するものと会話に関係するものを同定したことは、よく知られている[Darwin65]。ここで会話的表情研究の簡単な紹介をする。

(1) 会話的表情の理論

FridlundとGilbertは、表情の役割は本来音声による会話に情報を追加することであり、情緒的情報を提供することではないという見方を提案した[Fridlund85]。この理論には二つの仮説がある。

第一の仮説は、表情は本来会話的であるということである。表情は他者に情報を運ぶために用いられる。運ばれる情報は情緒的情報もあるかもしれないが、他の種類の情報もありうる。それらは例えば話の区切りを示すものや、話者の話を理解しているというサインや、傾聴者のコメントなどである[Chovil89]。

表情はそれ自身でも会話を行うことができる。すなわち、表情は他の会話行為とは独立にメッセージを送信することができる。ウィンクのような顔記号（facial emblem）、傾聴者のコメント（同意あるいは反対、不信あるいは驚き）などはその例である。表情は他の会話行為と共同して情報を提供することもある。例えば、音声のストレスパターン、声の調子、手の動作が果たすのと同じように、話し言葉の曖昧性の解消を表情で行うこともある。これは表情で情報を補足したり、例示したりすることによりなされたり、また、メッセージがどのように受け取られるべきかについての会話、すなわちメタ会話（例えば、冗談を言うときに笑いながら言うこと）によったりもする。

第二の仮説は表情は本来社会的であるということである。表情は他者に情報を伝える目的で使われ、その表現は背後にある情緒的過程よりは社会的な状況により、より多く統制されるという仮説である。

(2) 会話の中の表情

表情をそれらの会話的役割に応じて分類しようという試みはいくつかある[Chovil89, Ekman69, Sherer80]。同様な試みとしては、EkmanとFriesenによる情緒的表情の分類がよく知られている[Ekman84]。情緒的表情は基本的に状況独立、すなわち、その意味はいつも同じである。情緒的分類と異なり、会話的分類においては、顔記号を除きすべてが状況依存である。すなわち、会話的表情の解釈は、それが現われる状況に依存している。そのときの会話の文脈や他のチャネルとの時間的前後関係（同時あるいは少しの遅れなど）などが、表情の解釈に重大な意味を持つ。この意味で、会話的表情は特にマルチモーダルの中で生きてくるものといえる。

表2.2-1　会話的表情

SYNTACTIC DISPLAYS	
1. Exclamation marks	Eyebrow raising
2. Question marks	Eyebrow raising or lowering
3. Emphasizers	Eyebrow raising or lowering
4. Underliners	Longer eyebrow raising
5. Punctuations	Eyebrow movements
6. End of an utterance	Eyebrow raising
7. Beginning of a story	Eyebrow raising
8. Story continuation	Avoid eye contact
9. End of a story	Eye contact
SPEAKER DISPLAYS	
10. Thinking/ Remembering	Eyebrow raising or lowering, Closing the eyes,
	Pulling back one mouth side
11. Facial shrug/ "I don't know"	Eyebrow flashes, Mouth corners pulled down,
	Mouth corners pulled back
12. Interactive/ "You know?"	Eyebrow raising
13. Metacommunicative/	Eyebrow raising and looking up and off
Indication of sarcasm or joke	
14. "Yes"	Eyebrow actions
15. "No"	Eyebrow actions
15. "Not"	Eyebrow actions
17. "But"	Eyebrow actions
LISTENER COMMENT	**DISPLAYS**
18. Backchannel/Indicating attendance	Eyebrow raising, Mouth corners turned down
19. Indication of loudness	Eyebrows drawn to center
Understanding levels	
20. Confident	Eyebrow raising, Head nod
21. Moderately confident	Eyebrow raising
22. Not confident	Eyebrow lowering
23. "Yes"	Eyebrow raising
Evaluation of utterance	
24. Agreement	Eyebrow raising
25. Request for more info	Eyebrow raising
26. Incredulity	Longer eyebrow raising

表2.2-1は、いくつかの会話的表情分類の結果[Chovil89, Ekman69, Sherer80]をまとめたものである。表は三つの大分類からなる。

syntactic displays.

(a)　特定の語、句、節を強調する表情

(b)　発話の構文的側面にかかわる表情

(c)　話の全体構造にかかわる表情

speaker displays.

(a)　発話中の事柄を例示する表情

(b)　発話中の事柄に情報を追加する表情

listener comment displays.

話者以外の人が話者の発話に対する反応として作る表情

表2.2-1に示した26個の表情を、前項で紹介したソニーCSLのシステムで合成したものを図2.2-2に示す。少年の顔のテクスチャをマップしてある。二つの表情、「微笑（smile)」と「中立（neutral)」が追加してある。「中立」の表情はどの筋肉も弛緩しているときの表情であり、会話的信号を何も示さないときに用いる。

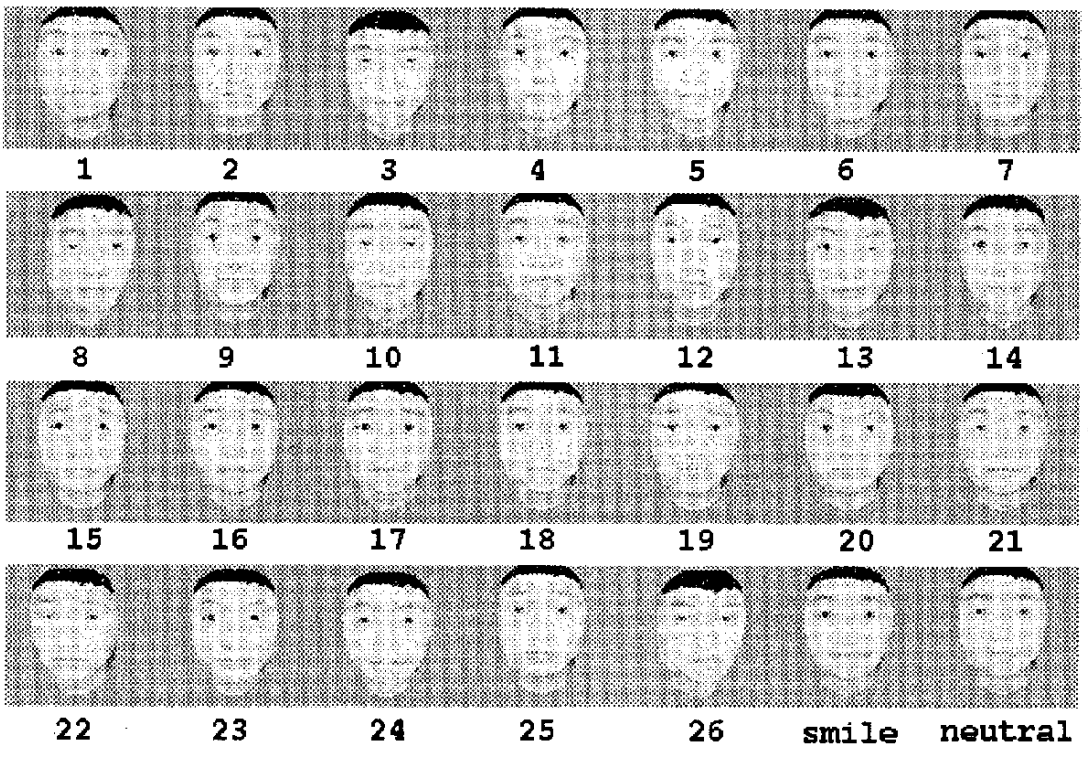


図2.2-2　合成された会話的表情

2.2.4　音声対話との統合：音声入力との統合

コンピュータと人間のインタラクションにおける会話的表情の効果を調べる実験環境として構築された音声対話システム[Takeuchi93]がある。実験システムは二つのサブシステムからなる。一つは様々な表情を持つ3次元の顔を生成する顔アニメーションサブシステムで、他は入力音声を認識、解釈し、音声出力を生成する音声対話サブシステムである。現在アニメーションサブシステムはSGI 320VGXで、音声対話サブシステムはSony NEWSワークステーションでそれぞれ稼働しており、両サ

ブシステムはLANで通信し合っている。図2.2-3にシステムのアーキテクチャを示す。

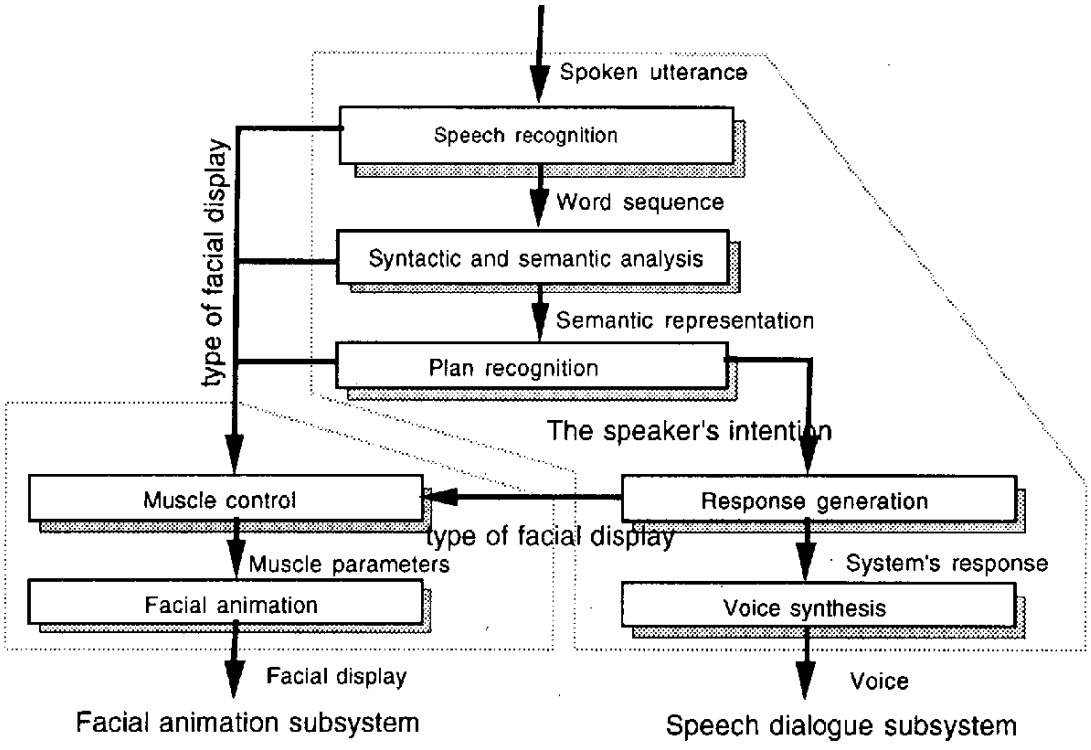


図2.2-3　システムアーキテクチャ

アニメーションサブシステムは毎秒15フレーム生成でき、ほぼリアルタイムアニメーションを実現している。Speech recognitionモジュールは、東京工業大学と共同で開発された[Itou92]。また、Response generationモジュールはソニー中央研究所で開発された規則合成に基づく音声合成ソフトウェアを使っている。

音声対話システムは対話において重要ないくつかの典型的な会話状況を認識し、これらを表情と関連付ける。例えば、入力音声が認識できなかったときや結果が構文的に正しくなかったとき、listener comment display #22 "not-confident"を表示する。話者の質問がシステムの知識の外であった場合はシステムの表情は"facial shrug"となり、「その質問にはお答えできません」という応答をする。会話状況と表情との対応表を表2.2-2に示す。

実験システムを用いた実験により、表情がシステムとのインタラクション、特に初期の接触に役立つこと、また、表情を通した初期のインタラクションが、後のインタラクション（表情のないインタラクションでさえ）を改善することが示されたと報告されている[Takeuchi93]。これらの結果は、表情が、コンピュータシステムに対してユーザの抱く精神的なバリアを軽減するのに大いに効果があることを定量的に証明しているといえよう。

表2.2-2　会話状況と会話的表情の対応

Conversational situations	Facial displays
recognition failure	listener comment display #22 "not-confident"
syntactically invalid utterance	listener comment display #22 "not-confident"
many recognition candidates with close scores	listener comment display #21 "moderately confident"
beginning of dialogue	listener comment display #18 "indication of attendance"
introduction to a topic	syntactic display #7 "beginning-of-story"
shift to another topic	syntactic displays #7 "end-of-story" and #9 "beginning-of-story"
answer "yes"	speaker display #14 "yes"
answer "no"	speaker display #15 "no"
out of the domain	speaker display #11 "facial shrug"
answer "yes" with emphasis	listener comment display #23 "yes" and syntactic display #3 "emphasizer"
violation of pragmatic constraints	listener comment display # 26 "incredulity"
reply to "thanks"	listener comment display #23 "yes"
...	...

2.2.5　視線を作る：映像入力との統合

前項では、音声と視覚（表情）という二つのモダリティを使う例を紹介した。音声については人間とコンピュータ双方が使い、表情についてはコンピュータだけが使った（人間の表情をコンピュータは無視した）。表情についてこのように非対称であるのは、コンピュータが視覚を持たないからである。そこで、ここではコンピュータに視覚を持たせたソニーCSLのシステム[内藤94]を紹介する。コンピュータが視覚を得れば、外界の状況をより深く理解することができ、それにより他の情報（音声情報、キーボードコマンドなど）をその文脈に基づいて解釈できるであろう。しかし、現実にはコンピュータの処理能力の限界がある。あくまでインタラクティブなシステムを設計しようとしているのであるから、基本的に入力に対するコンピュータの反応は十分に速いものでなくてはならない。ところが、入力の処理（解釈）を十分に行おうとすると計算量が増大していき、時間もどんどんかかっていく。つまり、入力の処理の深さと応答時間とは正比例（深ければ深いほど時間もかかる）する。インタラクティブなシステムを設計するためには、適当な妥協点を見い出さなくてはならない。妥協点は適当な応答時間内で可能な処理の内容を与えるものである。これは一般に使用するハードウェアに依存する。ソニーCSLのシステムの場合、SGI VideoLabという装置を用い、ビデオ画像を毎秒30枚メモリに取り込み、これについて以下のような簡単な演算を行うことを可能にしている[内藤94]。

① 入力画像を粗くサンプリングする。

② 予めとっておいた画像（参照画像と呼ぶ）との差分をとる。

③ 上で変化部分として検出された点を領域化し、重心を求める。

参照フレームとして、だれもいない部屋の画像を使い、かつビデオカメラを固定すると、以上の処理により、ユーザの位置を各フレームごとに抽出することができる（部屋の中で動くものは人間だけであるというヒューリスティクスを使っている）。

こうして得られた情報を基に、ユーザに視線を向けることや、ユーザの出入室、激しい動きに応じた応答をすることなどが可能になる。図2.2-4に全体の模式図を示す。

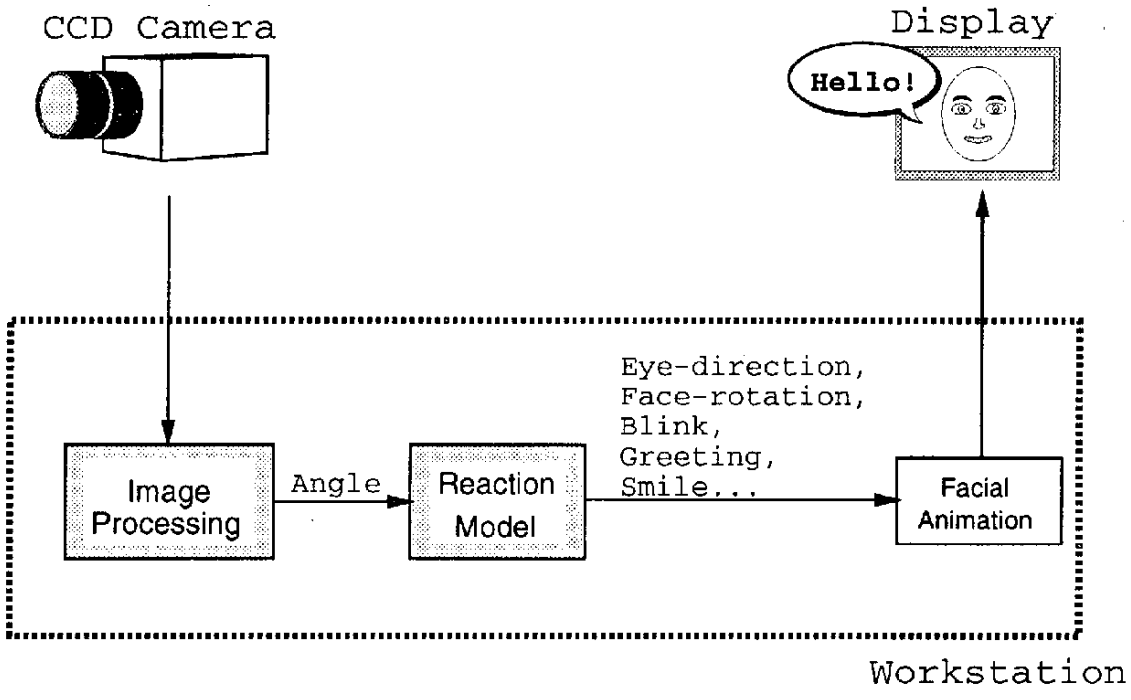


図2.2-4　画像入力に基づく表情アニメーションシステム

得られた情報に基づく動作の中で、視線の生成が最も興味深い。視線は3次元中のある対象を見ているかのように眼球を描くことにより表現されるのだが、ユーザの位置が得られたことにより、ユーザを見つめる視線を近似的に生成できる。人間どうしのインタラクションにおいて、視線の使い方は非常に多彩で容易に形式化できない。アイコンタクトとも呼ばれる視線のインタラクションは、それが置かれた文脈の中で様々な意味を帯びる。例えば、注意を向けていることを知らせる場合もあれば、何かを始める合図だったり、また何かを目で指し示すときには、そのものと相手とを交互に見つめたりもする。また特別な表情を相手に伝えたいときは、相手の視線を捕まえなくてはならない。視線を逸らすというのも視線にまつわる振舞いの一つである。視線のこの微妙さ、洗練度の高さは、文脈によって意味が様々に変わりうることからきている。これは視線がマルチモーダルインタラクションの重要な要素であることを意味している。

ところで、アイコンタクトというのは本来的には3次元の事象である。例えば、A、B、C三者がいれば、CからはA、Bの一方だけに選択的にアイコンタクトを仕掛けることが可能である。ところが、Cをコンピュータディスプレイとしたときには、これができない。A、B両者からもCが自分を直視しているように見えるからである。この原因は、ディスプレイCが平面であることによる。これを解決する方法について、[内藤94]はいくつか方法を述べている。図2.2-5に示しているのはその中の一つで、ディスプレイをコンピュータが直視したい方向に物理的に回転させる方法である。

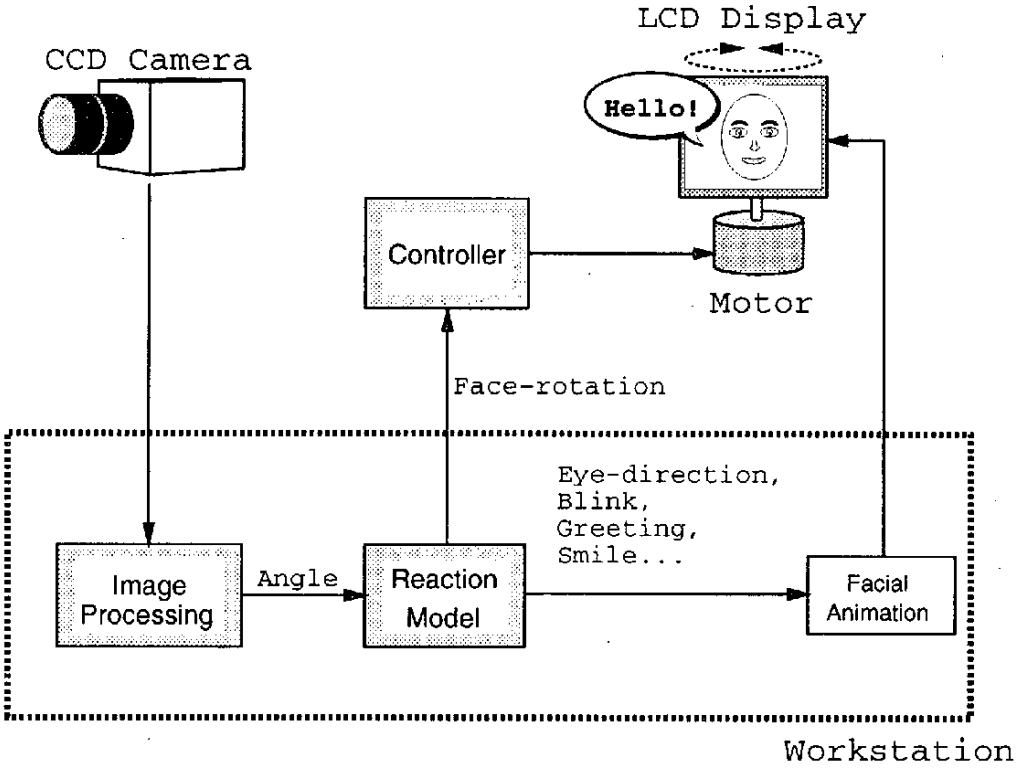


図2.2-5　モータで首を回転させる表情アニメーションシステム

現在の画像処理は上述のように限られており、個別の物体認識や情景理解などを行っていない。これらが段階的にでも実時間で可能になれば、視線の意味付けもより多彩なものになるであろう。

2.2.6　インタラクションの社会性：Social Agentへ

今後の研究方向として、さらに多くのチャネル、モダリティを開発し、それらを統合することが考えられる。その中で、音声認識や合成における韻律情報の処理、及びジェスチャや表情の認識は大きく進むであろう。

今まで実現されたコンピュータと人間との対話システムは過剰規制ぎみであった。これは会話が限

られたチャネルだけで行われ、その狭いチャネルで情報の衝突を避けるために規制が必要なためであった。多くのチャネルを用いることにより会話を規制する必要はなくなり、その結果として粒度の細かい、自由に割り込むことが可能で、より多くの自然な自発的な発話を誘引するような新しいスタイルの会話が可能になると思われる。このような会話は我々の日常的な会話にきわめて近く、それゆえにまたコンピュータをより一層身近かなものに感じさせてくれるだろう。

前項でも述べた画像処理の問題は、いわゆる音の理解の問題にも共通している。従来の音の認識は音声認識が独占してきたが、マルチモーダルインタラクションの立場からは、人間の出す音でも、言葉にならない「あー」「うー」にも意味がある。また人間以外の出す音、雨の音や、風の音、コップの落ちた音や、戸の閉まる音など、外界の音はすべて意味があるかもしれない。コミュニケーションは抽象的な信号ゲームではなくて、現実世界で起きている様々なインタラクションの一つである。その意味はそれが生起している状況に深く依存している。その意味の解読には記号化された文の解釈だけでは足りなくて、状況自体に対する深い理解が必要である。マルチモーダルインタラクションにおける視覚や聴覚の意義はここにある。

マルチモーダルインタラクションの研究を進めていると、様々に織りなされているメッセージをどう読み取るか、逆に多様なモダリティを駆使してどう表現するかという問題になる。言語と身振りの協調による3次元空間での対象物の指示といった限定された問題ならば、比較的容易に解けるであろうが、一般に視線や表情を参照しながら言語や身振りの解釈をするのは困難であろう。こうした分野はまだ全く形式化されておらず、今後それが進むともあまり思えない。言語の形式化に多くの努力がなされ今日に至っている。それは言語の理解を深めたが、その難しさもまた認識されている。マルチモーダルなインタラクションは言語活動よりはるかにインフォーマルなものであり、文化に依存したものでもある。マルチモーダルなインタラクションを理解するにはもっと別のアプローチが必要なのではないだろうか？

そのようなアプローチとして、social agentの研究が考えられる。Social agentは外界の情報をセンサを通じて絶えず入手し、それに基づいて自律的に動作するプログラムである。センサの入力には、外界の状況を伝える画像や音などが含まれる。Social agentの動作は、ディスプレイやスピーカ、マニピュレータなどを使ったマルチモーダルな表現である。ディスプレイには顔があり、様々な表情を表示できるはずである。スピーカは様々な調子の声で話せるはずであり、マニピュレータは何らかの身振りができるはずである。

Social agent研究のポイントは、外界からのマルチモーダルなメッセージを主体的に読み取るための「主体」、また、外部に対して何かを主体的に表現するときの「主体」、この「主体」の設計である。主体の設計は同時に他の「主体」を認知する機構の設計でもある。まだ、始まったばかりのアプローチで多くを説明できないが、その一部について[Takeuchi95]に書かれておりそちらを参照されたい。

【参考文献】

[Blattner92] Blattner, M : Multimedia and Multimodal User Interface Design: CHI'92 Tutorial Course Note 4. ACM Press, 1992.

[Chovil89] Chovil, N.: Communicative Functions of Facial Displays in Conversation. Ph.D. Thesis, University of Victoria, 1989.

[Darwin65] Darwin, C.: The Expression of Emotion in Man and Animals. University of Chicago Press, Chicago, 1965.

[Don92] Don, A. and Brennan, S. and Laurel, B. and Shneiderman, B.: Anthropomorphism: from Eliza to Terminator 2, In Proc. CHI'92 Human Factors in Computing Systems (Monterey, May 3-7, 1992), ACM Press, pp.67-70.

[Don91] Don, A. and Oren, T. and Laurel, B.: GUIDES 3.0, in Proc. CHI'91 Human Factors in Computing Systems (New Orleans, April 27-May 2, 1991), ACM Press, pp. 447-448.

[Ekman69] Ekman, P. and Friesen, W. V.: The repertoire of nonverbal behavior - categories, origins, usage, and coding, Semiotica 1 (1969), pp. 49-98.

[Ekman78] Ekman, P. and Friesen, W. V.: Facial Action Coding System. Consulting Psychologists Press, Palo Alto,1978.

[Ekman84] Ekman, P. and Friesen, W. V.: Unmasking the Face. Consulting Psychologists Press, Inc., Palo Alto, California, 1984.

[Fridlund85] Fridlund, A. J. and Gilbert, A. N.: Emotions and facial expression, Science, 230 (1985), pp. 607-608.

[Hindus92] Hindus, D. and Brennan, S.: Conversational Paradigms in User Interfaces: CHI'92 Tutorial Course Note 11. ACM Press, 1992.

[Itou92] Itou, K. and Hayamizu, S. and Tanaka, H.: Continuous speech recognition by context-dependent phonetic HMM and an efficient algorithm for finding N-best sentence hypotheses, in Proc. ICASSP'92, IEEE Press, pp. I 21-I 24.

[内藤93a] 内藤, 竹内, 所：表情設計のための筋肉エディタ, 日本ソフトウェア科学会第10回大会論文集, 1993

[内藤93b] 内藤, 竹内, 所：筋肉エディタによる表情アニメーションの向上, 情報処理学会グラフィクスとCADシンポジウム, 1993.

[内藤94] 内藤, 竹内, 所：視線を伴った表情合成システム, 日本ソフトウェア科学会インタラクティブシステムとソフトウェアについてのワークショップWISS'93論文集, 近代科学社, 1994.

[Perret84] Perret, D. I. et al.: Neurones responsive to faces in the temporal cortex: studies of

functional organization sensitivity and relation to perception. Human Neurobiology, 3 (1984) 197-208.

[Sherer80] Sherer, K.R.: The functions of nonverbal signs in conversation, in The Social and Psychological Contexts of Language, St. Clair, R. N. and Giles, H. (Eds.), Lawrence Erlbaum, Hillsdale, NJ, 1980, pp. 225-244.

[Takeuchi92] Takeuchi, A. and Franks, S. A: Rapid Face Construction Lab. Tech. Report. SCSL-TR-92-010, Sony Computer Science Laboratory, Inc., Tokyo, 1992.

[Takeuchi93] Takeuchi, A. and Nagao, K.: Communicative Facial Displays as a New Conversational Modality, In Proc. INTERCHI'93 Human Factors in Computing Systems (1993), ACM Press.

[Takeuchi95] Takeuchi, A. and Naito, T.: Situated Facial Displays: Towards Social Interaction, In Proc. CHI'95 Human Factors in Computing Systems (1995), ACM Press.

[Waters87] Waters, K.: A muscle model for animating three-dimensional facial expression, in Computer Graphics 21, 4 (July 1987), 17-24.

2.3 マルチモーダルにおける音声の役割

2.3.1 はじめに

人間どうしの日常のコミュニケーションでは、話し手は音声を中心に、表情や視線、身振りや手振りを用いて、情報を発信している。一方、聞き手は、音声信号から、言語情報と、話し手の感情や状況、性別や年齢などの個人性情報とを同時に受信している。

人間にとって自然な音声メディアは、コンピュータとのインタフェースとしても望ましく、音声認識はキーボードに代わる情報入力手段として、音声合成は言語情報のビジュアル表示に代わる情報提示手段として期待されてきた。音声認識と合成機能とを利用する音声対話システムの研究も活発化しており、音声は、他の入出力メディアの代替手段としてではなく、コンピュータとの自然な対話メディアとして重要視されるようになってきた。また、音声の有するhand-free、eye-freeの特徴や、他のメディアとの連合が容易であるという特徴を、マルチモーダルインタフェースとして活用する試みが盛んになってきた。また、最近のコンピュータのマルチメディア化の進展により、音声入出力機能を装備したパーソナルコンピュータが普及しつつあり、音声認識や音声合成機能が外付けハードウェアなしで容易に利用できる環境が整備され、音声が本格的に応用できる段階に至った[Arons94]。

ここでは、マルチモーダルインタフェースの観点から音声の特徴と音声情報処理について述べる。また、マルチモーダルインタフェースに関わる音声の研究事例と今後の展望について述べる。

2.3.2 音声の性質

音声は、知能や感情とも関わる人間の根元的な情報伝達手段であり、永年にわたって下記の側面から研究が行われてきた。

- 音響信号
- 音声言語
- 音声と聴覚
- 知覚と生理
- 音声情報処理技術（符号化、認識、合成、対話）

マルチモーダルインタフェースとして音声を活用するためには、多様な音声の性質を把握することが必要である。以下、音声の重要な性質について述べる。

音声は、音響時系列信号であり、主な役割は言語情報の伝達であるが、感情や話者の個人性も同時に伝達する。音響信号であるため、背景騒音の影響を受けるが、同時に騒音源にもなり得るという性質がある。

話し言葉は、言い淀みや言い直し、語順の逆転や省略を含み、書き言葉よりも制約が少なく表現が多彩である。また、音声言語により、文字言語では表せない強調や疑問、微妙なニュアンスを、アク

セントやイントネーション、発話速度を変化させることにより伝えることができるという長所がある。

音声が、コンピュータの他の情報メディアと根本的に異なる点は、発声器官から出力され、聴覚器官に入力されるという人間にとっての双方向性のある（入出力）メディアであるという点である。発話は、聴覚からのフィードバックの影響を受けながら行われるなど、音声の生成と認識は互いに密接に関係している。また、音声は、文字や画像とは異なり、声帯と声道（唇や舌の位置や形状）に基づく音声生成モデルがあり、さらに、神経レベル、筋肉レベル、調音レベル、信号レベル、音声学レベル、言語レベルにおいても、音声の生成や認識について多くの研究がなされている。

音声によるコミュニケーションでは、話し手は、脳の多様な機能を統合して伝えたい内容を音声言語として表現し、発声器官を動作させて音声を生成している。聞き手の方も、単なる聴覚器官への音の受容の他に、脳の様々な機能を動員して、音素、音節、単語、文等の音声言語の構文意味情報や、音声波に含まれる感情等のパラ言語情報を抽出し、話し手の意図や状況を理解している。また、聞き手は、話し手の表情やジェスチャなどの視覚情報と、聴覚情報を連合して、話し手の意図や話の内容を理解している。

2.3.3 マルチモーダルインタフェースと音声情報処理

現状のマルチメディア技術は、図2.3-1(a)に示すようにメディアの種類と情報量の拡大に終始し、複数のメディアをbitレベルで入出力（記録再生）しているだけの場合が多い。使いやすく自然なインタフェースを実現したり、情報が価値を生み出すためには、大前が指摘するように、単なるbit表現レベルの情報伝達ではなく、内容（content）理解の方が重要である[Takebayashi92, 大前92]。音声情報処理においては、音声符号化や音声蓄積交換はこの(a)のモデルに対応する。音声応答をこのモデルにより実現すると、肉声を録音するコストを要するという欠点はあるが、明瞭で品質の高い合成音が得られるという利点があり、商用の音声応答システムなどに採用されている。

図2.3-1(b)に示すメディアの理解機能はユーザが入力した音声、キーボード、マウス、画像情報を処理するプロセスであり、ユーザの意図理解を目的とする。このメディア理解のプロセスは、多大な情報量を有するパターン空間から情報量の少ない計算機の内部表現への写像である。この情報の圧縮の過程で認識エラーや曖昧性が生じる。したがって、マルチモーダル対話システム構築のためには、エラーの削減とともに曖昧性への対処が必要となる。特に、音声メディアによるコンピュータとの対話を考えると、音声信号から単語や文への表層的なメディア変換よりも、発話の内容理解や意図理解と音声と他のメディアとの融合が大切である。

一方、図2.3-1(c)のメディア生成のプロセスは、コンピュータの内部表現からパターン空間への1対多のill-formedな情報量を増やす写像である。すなわち、生成可能なパターンが多数存在し、理解プロセスとは別種の曖昧性が生じる。情報を提示する場合、人間は情報の内容よりも付加されたメディ

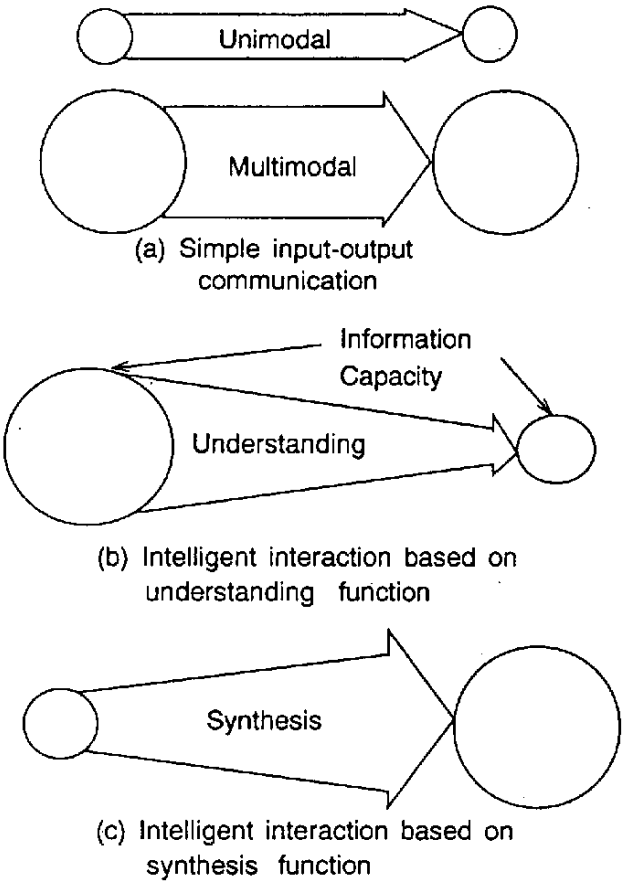


図2.3-1　マルチメディア通信モデルの比較

アそのものを注視する傾向があるので、人間の知覚、認知に合致したメディアの生成が必要である。(c)に相当する音声情報処理は、任意のテキストを音声に変換するテキスト合成であり、文全体のイントネーションや単語のアクセントを決定した後に、音声の素片をつなぎ合わせて合成音声を出力する。現状の技術レベルでは、自然性の向上に重点を置いているが、将来はイントネーションやアクセントと発話速度の制御により、意図や感情の表現能力が高まることが予想される。

以上のような特徴を有する音声のインタフェースとしての長所を以下にまとめる。

・特別の訓練が不要で負担や制約の少ない情報伝達メディアである。

・迅速で高速な情報発信ができる。

・言語情報の他に、話し手の意図や感情、個人性情報も同時に伝達する。

・非接触で信号を伝達するため、動作に関する制約が少ない。

・hand-free、eye-freeであり、他の入出力手段と併用できる。

音声を用いてインタフェースを設計する際に留意すべき点を以下にまとめる。

・音声入出力は環境騒音を発生し、周囲に迷惑をかける恐れがある。

・音声認識エラーを回避するのは困難である。

・音声認識は雑音や不用意な発声に弱く、発話様式や単語数などの制約が強い。

・文-音声変換による合成音声の品質は録音編集型に比べ不十分である。

・音声情報処理は文字情報処理よりも多くの計算機パワーと通信容量を必要とする。

・音声は時系列メディアであるため、視覚情報と比べて情報伝達速度が遅い。

・音声は一過性のメディアであり、聞き逃した部分だけを聞き返すことは困難である。

マルチモーダルインタフェースにおいて音声を有効利用するためには、上記の音声の長所と留意事項について十分検討することが必要である。

2.3.4 マルチモーダルインタフェースとしての音声利用の事例

(1) 音声自由対話システム TOSBURG II [竹林94]

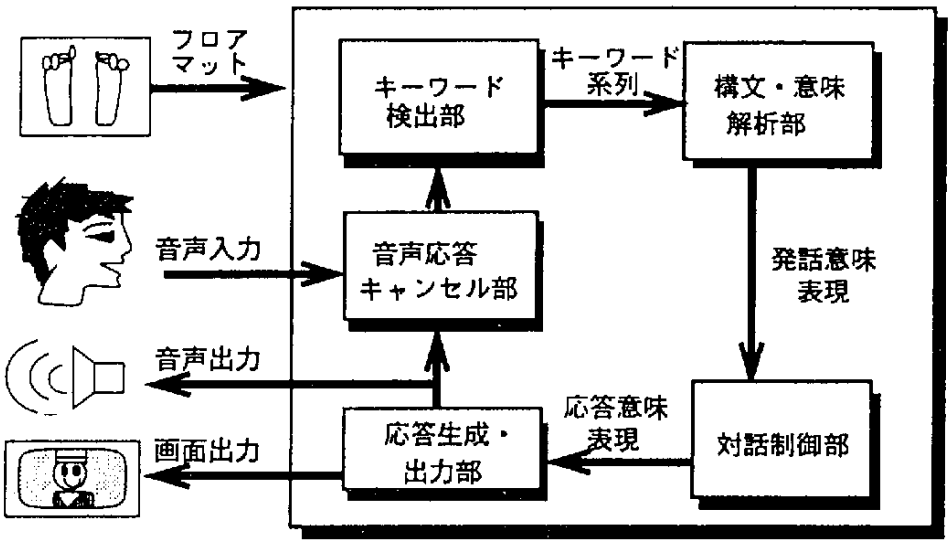


図2.3-2 音声自由対話システムTOSBURG IIの構成

図2.3-2にシステム構成を示すTOSBURG IIは、「不特定のユーザに対して何ら制約を設けない」という条件下で、通常の話し言葉で音声応答を遮って音声入力できる初の音声自由対話システムである。

音声応答キャンセル部は、適応フィルタを用いて入力音声信号中から音声応答成分を常時引き去り、システムが音声応答中でもユーザの割込み発話を随時受け付けることを可能とした。音声理解部はキーワードに基づき自由発話を理解し、対話制御部は、対話の状況と対話の履歴情報を用いて対話音声理解と適切な応答意味表現の生成を行い、ユーザ主導型の対話を実現する。応答生成部では、応答意

味表現から、合成音声とその応答文、グラフィクスからなるマルチモーダル応答を生成出力する。システムからユーザへの応答は、合成音声だけでなく種々のメディアを利用しているので、音声メディアの一過性という欠点を補うことができる。さらに、確認の応答音声と同時に注文の内容をアイコンなどによって視覚的に提示することにより、総合的なメッセージを簡潔に伝えることができる。また、店員の姿を表わすアニメーションを提示し、音声応答に同期させて口を動かすことにより、動いている実感のある目標（attention getter）を具体的に示し、ユーザが自然にシステムに音声入力できる雰囲気を作るとともに、店員という対話の相手を明示することにより、自然な発話を促そうとした。さらに、店員の表情は、対話の状況に合わせて、微笑んだり、申し訳なさそうな顔に変えたりし、システムからの応答の理解を助けている（図2.3-3参照）。

(2) マルチタスク／マルチモーダル対応音声認識インタフェース[橋本92]

音声認識機能のサーバ化により、音声と他の入力メディアの統合化とマルチタスク対応を実現した。図2.3-4に示すように音声認識サーバは、マルチタスク環境において一つのプロセスとして常時動作する。クライアントである複数の応用プログラムは、音声認識サーバに対して認識対象語彙を通知し、認識処理を依頼する。音声認識サーバは、依頼に従って認識処理を行い、結果をクライアントに送信する。このソフトウェア音声認識サーバでは、マルチモーダルな対話の実現をサポートするため、音声フォーカスとキーボードフォーカスを任意に設定可能とし、入力フォーカス情報をユーザに提示する。応用プログラムとして、マルチモーダル入出力の利点を活かし、ソフトウェアテキストリーダの編集、DTPシステム、マルチメディアメール、ウィンドウ制御等での有効性を確認した。特に、マルチタスク環境化で音声の並行入力機能を活かし、他の作業の途中での音声コマンド入力による、「ながらメール読み上げ」が効果的であった。

(3) 非言語音声の認識と合成システム[金沢94]

音声メディアの中でも非言語音声は日常の会話において頻繁に用いられており、話し手の意図や感情を手短に聞き手に伝達し、会話を円滑に進行する上で重要な役割を果している。例えば、文字で「えー」と書かれた非言語的情報を音声で表現する場合、図2.3-5に示すようにアクセント、ピッチ、持続時間を変化させることにより、意図や感情（肯定、驚き、納得、相槌、つなぎ）等の伝達が可能である。本システムはファジー論理に基づきアクセント、ピッチ等の韻律情報を制御し、感情を持つ非言語音声を合成する。また、ピッチとエネルギー情報を表現するため生のスペクトルパターンを用い、非言語音声による感情の認識を可能とした。軽快なレスポンスが必要な音声インタフェースとして有用であり、また、ユーザ以外の周囲の人にとっては、非言語音声なのでさほど耳障りでないという傾向があった。

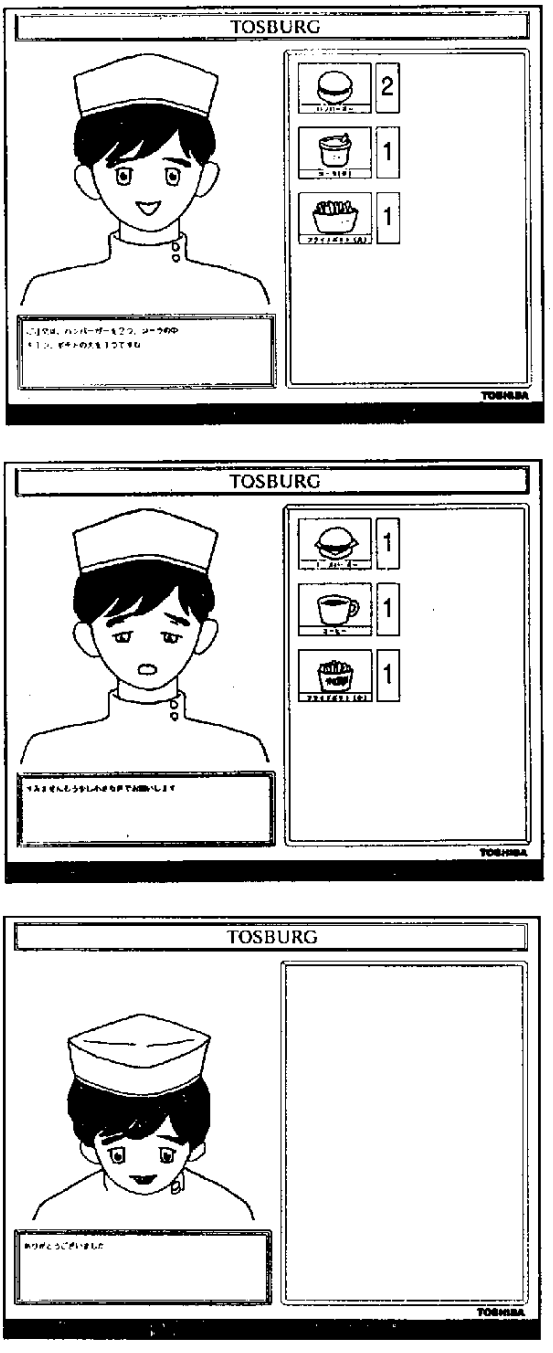


図2.3-3　視覚メディアによる応答と表情の例

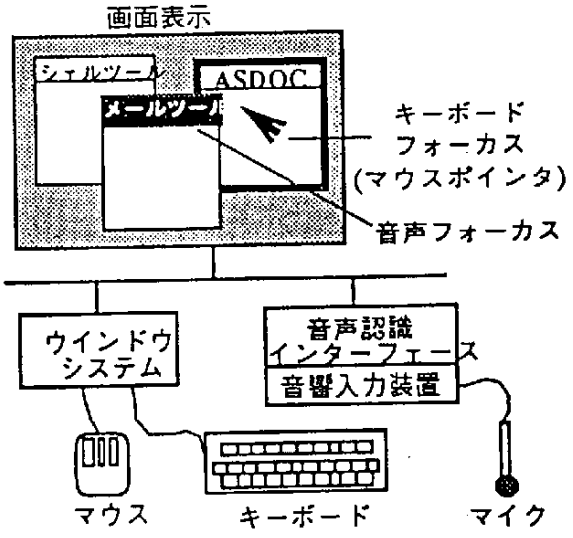


図2.3-4　ソフト音声認識サーバの構成（出典：[橋本92]）

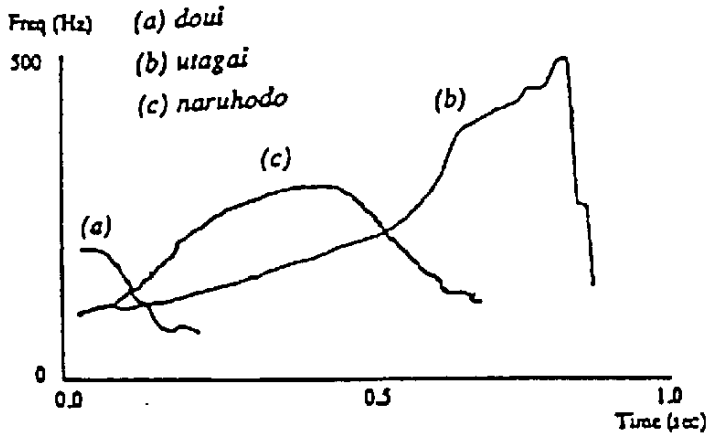


図2.3-5　非言語音声の基本周波数パターン（出典：[金沢94]）

(4)　マルチモーダル対話システム MultiksDial [神尾94]

本システムは、入力手段に音声認識装置とタッチパネル、出力手段に音声規則合成装置とディスプレイを備え、入出力の双方をマルチモーダル化している。また、補助入力手段に光電センサを使用し、ユーザの状況を検知しながら操作ガイダンスを提示することにより、スムーズな対話を実現している。本システムは地理案内システムなどのような自動化情報システムへ応用が検討されている。図2.3-6にシステム構成を示す。

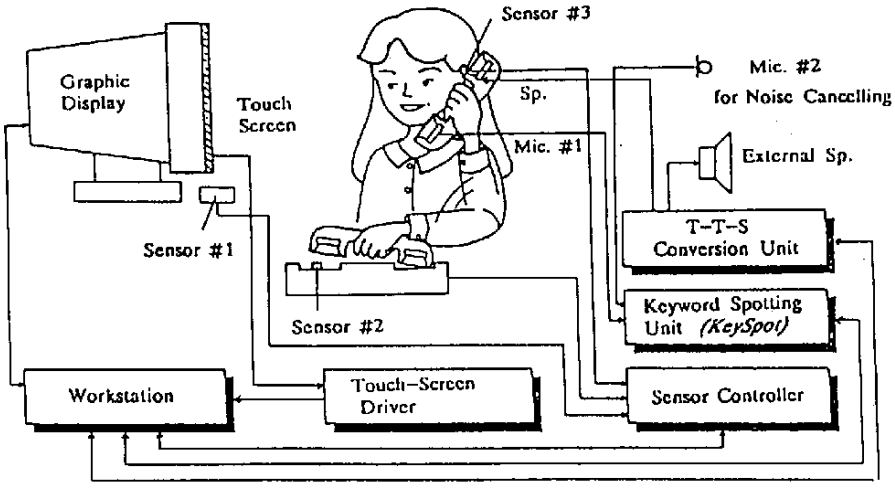


図2.3-6　マルチモーダル対話システムMultiksDialのシステム構成（出典：[神尾94]）

(5)　人物像と音声による知的インタフェース Human Reader [末永92]

図2.3-7に示すように視覚を利用するサブシステムである、頭部の動きを検出するHead Reader、及び手指の動きを検出するHand Readerにより、スクリーンの前に座った利用者の顔及び手指の動作を抽出理解する。同時に発声された音声を音声認識ユニットによって認識する。また、画像生成システムにより様々な表情を有する顔画像をCGで生成し、音声と同期させてスクリーン上に表示し、人間とコンピュータのより自然なインタラクションを助ける。応用としては、ワークステーションのウィンドウ切り換え、ビデオメール操作、プレゼンテーションシステムが検討された。

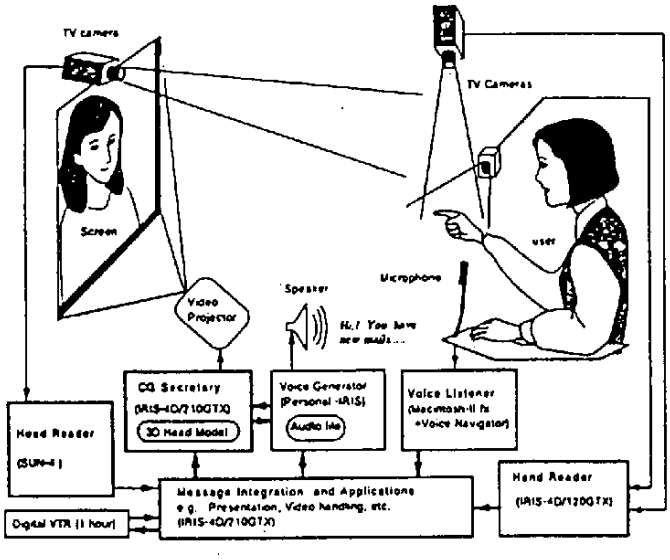


図2.3-7　Human Readerの実験システム（出典：[末永92]）

(6) マルチモーダル遠隔会議システム HuMaNet [Berkley94]

HuMaNetは、図2.3-8のように、音声認識によるコマンド入力、規則合成によるテキストの読み上げ、話者認識によるデータベースアクセスのセキュリティチェック、マイクロホンアレイによる音声検出と音源位置推定とそれに基づくカメラの制御、画像圧縮による静止画、動画の伝送など、様々な要素技術を統合したマルチモーダル遠隔会議システムである。音声認識については、連続して発声された単語を認識するとともに、認識対象以外の語彙のリジェクト性能を高め、不用意な発声による誤動作を防いでいる。現在、AT&Tベル研究所とRutgers大学を結んだ実験が行われている。

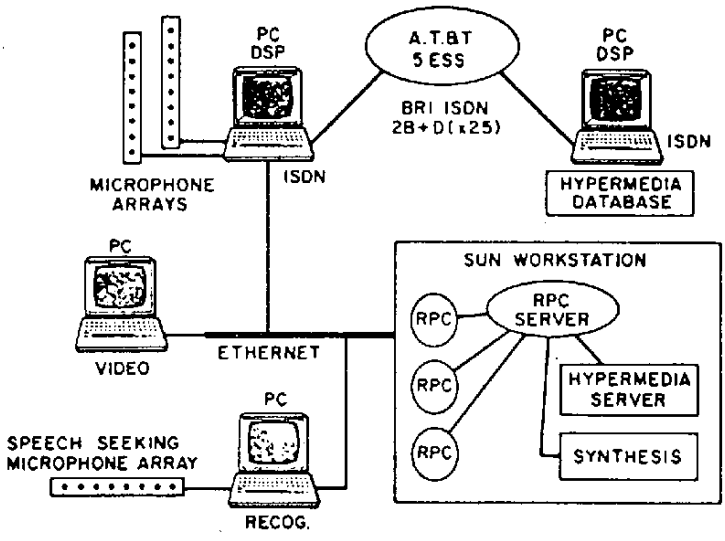


図2.3-8 HuMaNetのシステム構成（出典：[Berkley94]）

以上の事例のように、音声は他のメディアと様々の形で並行して利用でき、マルチモーダルインタフェースの中核的存在として研究開発が活発に行われている。

2.3.5 おわりに

Alan Kayが主張するように、次世代のコンピュータとのインタラクションは、デスクトップメタファに基づくGUIの「See and point」から、音声を中心とした「Ask and tell」に移行すると考えられる[Mclaughlin89]。

そのときの音声の利用は、キーボードやマウスの代替手段としてではなく、音声メディアの自然、迅速、hand-free、eye-freeの特長を活かし、他の入出力メディアを併用して、ユーザの意図を理解して適切な応答を返すAgentに基づくユーザ中心のマルチモーダルインタフェースであることが望ましい。そのためには、知的処理に必須の知識ベースを充実させ、音声処理、自然言語処理、知識処理技

術の高度化を図り、さらに、視覚メディアとの融合を深める必要がある。

【参考文献】

[Arons94] Arons, B. and Mynatt E.: The Future of Speech and Audio in the Interface, ACM SIGCHI Bulletin, 26(4),pp.44-48(1994).

[Berkley94] Berkley, D.A.: Flanagan, J. L., Hipley, K. L. and Rabiner, L. R.: A Multimodal Teleconferencing System Using Hand-Free Voice Control, Proc. Int. Conf. on Spoken Language Processing '94, pp.555-558 (1994).

[橋本92] 橋本秀樹, 永田仁史, 竹林洋一 :ワークステーションにおける音声認識インタフェースの検討, 情報処理学会研究会 HI-46-3, pp.17-24(1992).

[神尾94] 神尾広幸, 松浦博, 正井康之, 新田恒雄 : マルチモーダル対話システム MultiksDial, 信学論 D-II, J77-D-II(8), pp.1429-1437 (1994).

[金沢94] 金沢博史, クリス マエダ, 竹林洋一: 計算機との対話のための非言語音声の認識と合成, 信学論 D-II, J77-D-II(8), pp.1512-1521 (1994).

[Laurel91] Laurel, B.: Computers as Theatre, Addison-Wesley Pibulishing Company(1991).

[Mclaughlin89] Mclaughlin, F.: ICSE 11 prompt engeineers to reflect on yesterday, look to tomorrow, IEEE Computer, pp.110-112(1989).

[大前92] 大前研一: 情と知のパッケージ,信学誌,75(11),pp.1169-1174(1992).

[末永92] 末永康仁, 間瀬健二, 福本雅朗, 渡部保日児: Human Reader: 人物像と音声による知的インタフェース, 信学論 D-II, J75-D-II(2), pp.190-202(1992) .

[Takebayashi92] Takebayashi, Y.: "Integration of Understanding and Sysnthesis Functions for Multimedia Interfaces" in Multimedia Interface Design, Blattner ed., pp.233-256, ACM Press Book(1992).

[竹林洋一94] 竹林洋一: 音声自由対話システム TOSBURG II -ユーザ中心のマルチモーダルインタフェースの実現に向けて-, 信学論 D-II, J77-D-II(8), pp.1417-1428(1994).

2.4 マルチモーダルインタフェースの将来性

2.4.1 話のあらすじ

人と人とのコミュニケーションは基本的にマルチモーダルである。二人の人間が会話しているときには相手の話に耳を傾けるだけでなく、相手の目や表情、手振りなどにも目がいく。つまり、複数の伝達様式（モダリティ）を組み合わせて人は自分の気持ちを表現し、相手はそれを総合的に読み取る。計算機と人間のインタフェースは長い間、ディスプレイ（もっと古くはテレタイプ）とキーボードと相場が決まっていた。それを人間にとってもっと自然なマルチモーダルなものにしようというのが、最近のマンマシンインタフェース研究の大きな流れである[安村94]。この節では、この「マルチモーダル」という動きに将来性があるのかどうかを検討する。計算機とのインタフェースを考えた場合、いろいろなモダリティを計算機からの出力に利用する話と、計算機への入力に利用する話が存在する。これらは別々に検討する。また、技術的な可能性、将来性とそれらが社会に受け入れられるかどうかという問題も独立に議論することにする。

2.4.2 技術的将来性／可能性

(1) 複数のモダリティを計算機からの出力に利用すること

ディスプレイとキーボードという標準的なハードウェアを利用している場合でも、ディスプレイには簡単なスピーカが付いていて、エラーや誤入力を知らせるためのビープ音や、打鍵に対するフィードバックとしてのキークリック音などを出力していた。これが一番原始的な「音」というモダリティの使用方法であろう。ディスプレイに表示される映像も昔はテキストという文字情報がほとんどだったが、最近は図や、動画も使われるようになってきた。「映像」というモダリティには実に様々な情報が載せられる。この二つのモダリティ：「音」と「映像」の二つを組み合わせたものは既に多くの実例が存在する。計算機とのインタフェース以前に、「音」と「映像」の組合せは映画やテレビという媒体で既に十二分に実験、実証済みなので、いまさらその有効性を云々する必要はないだろう。ただ、個々の使われ方を見てみると、以下のように何に重点を置くかはアプリケーションによって様々である。

(a) テレビゲーム

まず思い付くのは、今やほとんどの家庭に普及したと言われるテレビゲームである。ただし、今まではハードウェアの制約もあり、「音」よりは「映像」中心のインタフェースになっていた。試しに音声をオフにしてもたいていのゲームが実行できることから、「映像」優位は明らかだが、「音」が反射神経レベルのフィードバックと、興奮を盛り上げる役割を果たしていることは注目に値する。つまり、それぞれのモダリティごとに、非常に明確な役割分担がなされているのである。ハードウェアや

コストなどの制約のきつい分野だけに、この役割分担が際だっているということもできる。新しいハードウェアを引っ提げて、新規参入組が「3次元」「リアル」などと言って攻勢をかけてきているが、想像力を刺激するためにはあまりリアルではいけないという専門家の意見もある。

(b) （語学用）CAI

CAIのシステムの中にも、「音」と「映像」の両者を併用したものが多い。典型的なのが語学学習用のCAIだろう。語学の場合、「音声」はかなり重要である。「映像」の方は、まずテキストの表示に使われ、それから会話等の状況を一目で分からせるための写真やビデオとして利用される。またここでは、単語（綴り）とその発音のように視覚情報と音声情報の対応がきちんとついていることが重要である。

(c) エキスパンドブック

CD-ROMに収録されたいわゆる「エキスパンドブック」も、最近は数多く出版されるようになってきた。しかし、よく見てみると内容によって、やはり「音」や「映像」あるいはテキスト（もちろん映像の一種）のどれかに重点が置かれているものが多い。音楽家の作品を紹介するようなCD-ROMでは「音」中心、美術館の作品巡りのようなCD-ROMでは「映像」中心、そして元々が本として出版されたもの、百科事典、人名録、各種のデータベース、そして普通の本などのCD-ROM版では、圧倒的にテキストが中心である。それらの中には、元々の本のテキスト情報プラス、著者が書斎などでいくつかの話題に関して語っているビデオの組合せといった安易な作り方のCD-ROMも多い。まだまだディスプレイが紙のクオリティに達していない現状では、元の紙に印刷された本の方がよっぽどましという例も散見される。

(d) WWWページ

爆発的な普及を続けているWWWは、テキストだけでなく映像や音声もネットワークを介して提供できるというのがその宣伝文句の一つである。ただし、先頭ページの美しいタイトルバックとしての映像や、MIDI等で提供される音楽が時折目や耳を楽しませてくれるが、その情報のほとんどはテキスト情報である。もちろん「映像」でしか表現できないもの、「音」でしか表現できないものは厳然として存在し、それらをネットワークアクセシブルにしようという努力は日々続けられているが、まだまだネットワークの容量など技術的な問題も残っている。

(e) 野心的な試み

たいていのマルチモーダルが、「音声」と「映像」という人間の聴覚と視覚に訴える媒体を利用し

ているのは当然のことである。それらが人間の情報入力チャネルとしてはメインのものだからである。しかし、さらに別のモダリティを使ってみようという試みもいくつかなされている。

一つは顔の表情及び、視線を利用しようという試みである[内藤 et al. 93]。これはディスプレイに顔が表示されるのでもちろん「映像」の利用であるが、この顔の表情や視線に情報を担わせているという点が斬新である。人間は顔やその表情に非常に敏感なため、無意識の内に表情の変化に反応してしまうという面白い結果が得られているが、正確な情報の伝達などにはそれほど向いていないようでもある[内藤 et al. 94]。

目の不自由な人のために、テキスト情報を点字表示機に変換して表示するというような応用[安村 et al. 93]は、触覚を利用した出力の例と考えることができる。このような障害者向けのインタフェースには視覚や聴覚以外の感覚を利用したものがいくつか考えられるが、健常者用のインタフェースで視覚や聴覚以外の感覚を利用するインタフェースはまだ多くはない。

(f)　マルチモーダル出力の可能性

以上見てきたように、既に存在するインタフェースを含めてマルチモーダルな出力を計算機とのインタフェースに利用する可能性は大きく、またその効果も大きいことが分かる。したがって、マルチモーダルな出力は今後大いに普及することが予想される。ただし、たった二つのモダリティであっても、その使われ方、役割分担はアプリケーションごとに千差万別で、なんでもかんでもマルチモーダルがいいというわけではない。また、現状では特に容量に関して、技術的な制約もかなりきついことが分かる。

(2)　複数のモダリティを計算機への入力に利用すること

人間はマルチモーダルな情報を的確にキャッチする能力に優れているが、計算機は一般には視覚情報や聴覚情報、あるいはその他の感覚情報の認識は非常に苦手である。したがって、今まではキーボードという甚だしくバンド幅の狭い入力機器に頼ってきたわけだが、最近は各種の認識技法を駆使したインタフェースも出現しつつある。しかしそれらは一つのモダリティを利用するもので、まだ複数のモダリティをうまく利用する入力方法は少ない。そこで単一モダリティの入力方法をいくつか眺めてみることによって、それらを組み合わせたマルチモーダルな入力の可能性を探ってみることにしよう。

(a)　手書き文字認識

手のひらサイズの電子手帳に利用されて再度脚光を浴びている手書き文字認識であるが、一昔前の入力手段としての評価は厳しいものであった。キーボード等の既存の入力手段との入力速度の差は歴

然としているし、認識率もそれほど高いものではない。キーボードアレルギーの利用者や、手帳のようにごく短いパッセージの入力という限定された状況だからこそ利用されているという面が強い。どんなに、かな漢字変換が不便でも手書き認識よりは速い。また、紙に印刷されたテキストの入力という場面ではOCRが活躍しているが、手書きの原稿などにはまだ適用できないのが現状であろう。

(b) 音声認識

音声合成が単純なものとはいえ既に実用に供されているのに対して、音声認識はまだまだ技術的な問題が多く、実用には至っていない。計算機とのインタフェースに音声を利用する試みや、市販の装置なども存在するが、認識対象は登録済みの単語に限られていて、不特定話者の連続音声の認識はこれからである。

(c) ジェスチャ

ポインティングデバイスの代わりに指による指示、あるいは特殊な棒による指示を利用する例が報告されている[大橋 et al. 94]。大橋等の研究では、指示棒と音声の両方を利用して、例えば積木の組立てを指示するというようなアプリケーションを考えているので、真の意味でのマルチモーダル入力になっている。ただし音声認識は特定話者の単語単位の認識ということで制約がきつい。このようなシステムがどれぐらい有効かは今後の検討待ちであろう。

(d) 視線入力

視線を捉えて、これもポインティングデバイスなどの代わりに利用しようという試みはいくつかあったが、視線を正確に捉えるためには、まだまだ大がかりなアイカメラ装置が必要で、これも実用の域には達していない。画面を3×3に9等分したうちの一つというぐらいの精度でよければ簡単な装置でも視線入力が可能だが、このような精度ではあまり複雑なアプリケーションは組み立てられない。アイカメラ装置はまだ実験のためのデータを得る装置であって、一般向けの入力装置にはなり得ていない。

(e) 表情、触覚、その他

その他のモダリティとして、表情、触覚、嗅覚、味覚などが考えられ、嗅覚や味覚などに関してはかなり精度の良いセンサなども開発されているようであるが、計算機とのインタフェースにそれらを使用したという話はまだ聞いたことがない。

(f) マルチモーダル入力の可能性

以上見てきたように、入力側は認識の仕組みが難しいために、単一モダリティでも実用化がかなり困難な状況にある。したがって、それらをさらに組み合わせてマルチモーダルにするという方向は、そう簡単に実現できるものではない。ただし、複数のモダリティを組み合わせると、単一モダリティでは困難であった認識が容易になるという可能性は十分に残っている。しかし、複数モダリティの統合による認識に関しては、認知科学的あるいは心理学的な研究が緒についたばかりなので、応用できるようになるまでには相当な時間がかかるものと思われる。

(3) 技術的問題点

以上のサーベイなどから、マルチモーダルインタフェースには以下のような技術的問題点があることが分かる。

(a) 複数のモダリティの統合

複数のモダリティの情報がバラバラに提示されたのでは、あまり利用者の役には立たない場合が多い。お互いが何らかの関連を持っているときに初めて情報が生きてくる。認識する場合も複数のモダリティを組み合わせることによって、より精度の高い認識が可能になる。複数のモダリティから得られた情報をどのような枠組みで統一的に扱うか、複数のモダリティへの情報の提示タイミングをどのように決めるかなど色々の問題を解決しなければならない。

(b) 適材適所の役割分担

「音」は利用者の注意を喚起するのに適している。感情に訴えるには「音声」がよい。論理的な内容はテキストで表現するのがよく、動きを表わすにはビデオが最適である。このように、表現したい情報ごとに、それを表現するのに最も適したモダリティが存在する。もちろん、通常あるモダリティで表現される内容を別のモダリティに変換することによって、思わぬ効果を生むこともある。「オーロラの光の変化を音楽にする」とか、「地震波の時間的変化を3次元立体表示する」などというのがそのいい例かもしれない。障害者用には、このような変換が不可欠である。しかし、通常はその情報に最も適したモダリティを利用するのが一番である。計算機とのインタフェースにおいてもこの原則は守らなければならない。ただ、どのような情報はどう表現するのが最適かに関するコンセンサスが、必ずしも得られていないのが現状である。

(c) 情報の出し過ぎの回避

たくさんのモダリティを使えば使うほど、ユーザに対してたくさんの情報を一度に提示することができる。しかしながら、人間の側の情報処理能力にも限りがあるので、多ければ多いほどよいという

ものではない。逆に情報の量を必要最小限にとどめる努力が必要である。最近のグラフィカルインタフェースは多分に情報出し過ぎの傾向がある。よく吟味し、モダリティも注意深く選択して、不要なものはできるだけ省略する勇気が必要である[斉藤93]。

(d) インタラクションの制御

マルチモーダルな情報提示を行うためには、いつどのモダリティの情報を提示するかというタイミングの制御が必要になってくる。さらに、利用者の状態や要求に合わせて、通常の提示モードを変更する必要性も生じてくる。このように人間がどのようにしてモード切り換えや、主導権の獲得などのインタラクションの制御を行っているかというのは、現在も活発に研究がなされている分野で、計算機で同様のことをどのようにやったらよいかに関しては、まだ十分な知見が得られていない。

2.4.3 社会が受け入れるかどうか？（社会的問題点）

計算機も人間やペットやロボットや会社（法人）のように社会を構成する一要素になりつつあるのが現在の情報化社会である。そのような中で、マルチモーダルなインタフェースを持った計算機が社会に受け入れられるかどうかは大きな社会的問題である。マルチモーダルな情報が計算機と人間の間で自由に行き来するようになると、人間は計算機を使っているというよりは、計算機というエージェントあるいはロボットとインタラクションしているという感覚が強くなってくる。そのようになってくると、単なる情報の表現手段の問題から、そのようなやりとり、インタラクションが社会的に見てどのような意味があるのかという話になってくる。この点をこの項では考えることにする。

(1) 計算機はどう見られるか？

今まで、主に計算機を利用してきたのは技術者、あるいは専門家と呼ばれる人種であった。彼らは、専門的スキルに磨きをかけ、計算機の微妙な使い方に自分の方を合わせることにそれほど抵抗を感じない人種である。しかし、これからは計算機の中味に関しては全く無知な一般ユーザ、子供などがどんどん計算機を使うようになってくる。そして、これら非専門家こそ、マルチモーダルインタフェースの主たるお客様である。それでは、彼らはどのようにエージェント化した計算機を捉えるだろうか？

当然考えられる反応は擬人化である。例えば、竹内ら[内藤 et al. 94]の試みているようなインタフェースを考えれば、ユーザがそれを擬人化して考える傾向を持つであろうことは容易に想像できる。そのようなインタフェースがなくても、人間は計算機システムに対して対人行動と同じような行動をとることが報告されている[Nass et al. 94]。その結果、ユーザはあたかも人間とインタラクションしているような気分で計算機とやりとりができるとなれば、マルチモーダルの将来はバラ色であるが、

はたしてそう簡単にいくだろうか？

「しりとり」という単純な遊びの領域ではあるが、上の報告とは反対に、相手が計算機であると分かっていると、インタラクションの楽しさが時間とともに減少してくるという興味深い報告もある[山本 et al. 94]。話はそう単純ではないのである[山本94]。人工知能などの技術の現状では、本当の人間の柔軟さを計算機でシミュレートするのは無理である。そのようなレベルで、表向きだけ人間のようなマルチモーダルインタフェースを作ったら、そして、そのインタフェースがうまく働かない状況が生じたら（その可能性は低くはない）、一般ユーザはそっぽを向いてしまうだろう。そのような危険性があることは十分に承知してかからなければならない。

(2) インタフェース製作コストの問題

マルチモーダルを安価にというのが安村等の主張であるが[安村 et al. 93, 安村 et al. 94]、一般には、複数のモダリティを利用するインタフェースは高くつく。そのようなコストを払っても見合うだけの効果があるのかということが問われることになる。映画やテレビ番組の製作費を見るまでもなく、情報提示を洗練していくとそれに応じて必ずコストがかかってしまう。製作用のツールを工夫するとか、うまい表示方法を考案するとかしないと、このコストの問題は回避できないと思われる。

(3) 人の適応（学習）に対応できるか？

人はいつまでも一般ユーザや初心者ではない。ある道具を使う経験を積むと、適応や学習によってスキルが向上し、初心者向けの「親切」なインタフェースがまどろっこしくなってくる。そのような利用者側の変化にインタフェースはついてこなければならない。あるいは、マルチモーダルなインタフェースがユーザとともに進化しなければならない。これもそう容易なことではない。

2.4.4 マルチモーダルインタフェースの将来

マルチモーダルインタフェースの可能性を技術的な可能性と、社会がそれを受け入れるかどうかという観点から考察してきた。計算機からの出力に色々なモダリティを利用する方向では多くの試みがなされ、近い将来多くのマルチモーダルインタフェースが出現するものと思われる。しかし、計算機への入力にマルチモーダルインタフェースを利用する方向の応用は認識技術のさらなる向上が必要なため、しばらく時間がかかりそうである。さらに、人工知能等の研究が進んで、自律エージェント、あるいはロボットと呼んでもよいような計算機が出現したとしても、それが社会の中に受け入れられるようになるためには、さらに一段上のロバストネスやフォールトトレランスが必要になってくると思われる。

計算機出力のマルチモーダル化から徐々にゆったりと真の意味でマルチモーダルなインタフェース

に発展していくというのが、妥当な将来像のようである。

【参考文献】

[内藤 et al. 93]内藤剛人, 竹内彰一, 所真理雄: 視線を伴った表情合成システム, WISS'93予稿集, pp.201-208, (1993).

[内藤 et al. 94]内藤剛人, 竹内彰一: Situated interface: 社会的インタラクションに向けて, WISS'94予稿集, pp.37-45, (1994).

[Nass et al. 94] Nass, C., Steuer, J., Henriksen, L., & Dryer., D. C.: Machines, social attributions, and ethopoeia: performance assessments of computers subsequent to "self-" or "other-" evaluations. International Journal of Human-Computer Studies, Vol.40, pp.543-559, (1994).

[大橋 et al. 94]大橋健, 山之内毅, 松永敦, 江島俊朗: 指示棒と音声が使えるコミュニケーション環境CoSMoSの提案, WISS'94予稿集, pp.29-36, (1994).

[斉藤93]斉藤康己: ビジュアルインタフェース研究の落し穴, ビジュアルインタフェースの研究開発報告書の1.4節, 日本情報処理開発協会, (1994.3).

[山本94]山本吉伸: 疑似対人行動 - 誘発の条件, 認知科学, Vol.1, No.2, pp.95-99, (Nov. 1994).

[山本 et. al 94]山本吉伸, 松井孝雄, 開一夫, 梅田聡, 安西祐一郎: 計算システムとのインタラクション - 楽しさを促進する要因に関する考察, 認知科学, Vol.1, No.2, pp. 107-120, (May 1994).

[安村 et al. 93]安村通晃, 伊賀聡一郎: マルチメディアからマルチモーダルインタフェースへ, WISS'93予稿集, pp.185-192, (1993).

[安村94]安村通晃: ビジュアルの次に来るもの, ビジュアルインタフェースの研究開発報告書の1.5節, 日本情報処理開発協会, (1994.3).

[安村 et al. 94]安村通晃, 今の潤, 八木正紀: マルチモーダルプラットフォームMAIの構築に向けて, WISS'94予稿集, pp.47-54, (1994).

3.　ソフトウェア開発における視覚化

3．ソフトウェア開発における視覚化

3.1　ソフトウェアの静的視覚化

3.1.1　ソフトウェアの視覚化と図形仕様

(1)　プログラムの図形仕様

ソフトウェアとりわけプログラムの視覚化は、コンピュータが開発された早い段階から行われてきている。事実、プログラムのロジックを視覚的に見せる流れ図の表記法は、コンピュータが登場してまもなく開発されたと言われている。流れ図は、処理を示す四角型の箱、判断記述のための菱形の箱、それらを結ぶ結線などを用いてロジックを記述する方法である。

初期のプログラミングは、限られたコンピュータメモリで効率よく実行するロジックの作成に多くの努力がなされたため、プログラミングスタイル（作法）はあまり重要視されなかった。また、プログラムを協同で作成する必要もなく、仕様は作成者が個人のメモとして作成するだけで十分であった。そのため見易さなどの工夫があまり行われなかった。しかし、開発するプログラムの規模が大きくなるに従い、作成されるプログラムは次第に複雑で理解しにくいものになっていった。また、プログラムの種類が増え、流通、再利用が促進されてくるに従って、流れ図の可読性の向上、情報の正確な伝播が重要になってきた。

プログラム（流れ図）の理解を複雑にし、信頼性を低下させているのは、無秩序な作成方法にあるとして、ダイクストラ（E. W. Dijkstra）は1960年代後半に構造化プログラミングを提唱した。この構造化プログラミングは、順次、判断、繰返しの3つの制御構造を基本とし、プログラムはそれらの組合せで構成する。また、処理は、1つの入り口と1つの出口のみが許される。構造化プログラミングの考え方の普及とともに流れ図に代わる多くのプログラムの表記法が開発された。主なものとしては、NSチャート、SPD（NEC）、PAD（日立）、YAC（富士通）などがある。

図形の具体的な表記法については、JIS X0128（プログラム構成要素及びその表記法）で詳しく対比されている。視覚的な観点からは、順次、判断、繰返しの制御構造が、直感的に把握でき、認知負荷がかからないものが望まれる。また、作図の観点からは、容易に記述でき、しかも変更がし易いものが望まれる。NSチャートは、処理全体を大きな四角い枠で囲み、その中を細分していくという方法をとっている。そのため見易さは向上しているが、中身を変更する場合は、書き直しとなる場合が多い。SPDは、処理の箇条書きを垂直線で結んで、構造を示す方法をとっている。そのため、記述の容易性は向上しているが、処理のまとまりの表現能力が落ちている。

なお、従来の流れ図の表記にもブロック構造が記述できるようにループ端（loop limit）と呼ばれる記号が追加された（JIS X0121：情報処理用流れ図・プログラム網図・システム資源図記号）。新しい表記法では、一組の台形の箱で一連の処理を囲むようになっている。下端の台形は倒置形である。

規模が数千ステップと小さく、対象とするシステムもそれほど複雑でないうちは、流れ図などを用いてプログラムの設計を行い、流れ図の上でロジックを推考し、コーディングを行う開発スタイルで十分であった。

しかし、開発するプログラムの規模が増大し、複雑になるに従い、直接ロジックを記述する方法は次第に困難になってきた。そのため問題領域で必要とされる機能を明確にし、コンピュータにマッピングさせる技法が要請された。こうした問題を解決させるために開発された技法が構造化分析、構造化設計と呼ばれる技法である。これらの技法は、手順とともにその表記法を定めている。

(2) 構造化分析／設計などで用いられる図形仕様

1970年代の始め、コンスタンチン（Constantine）らは、構造化設計と呼ばれるプログラムモジュールの構造化技法を開発した。構造化設計の中心はモジュールの構造図（Structure Chart）である。構造図は機能のチャンク（Chunk）を階層的に配置させたものである。この図の基本は、各チャンクの実行は閉じていなければならない、即ちチャンクの実行が終らなければ次のチャンクの実行が始まらないこと、及び各チャンク間にデータとコントロールの情報を明示することである。しかし、構造図は読むことは容易であるが、作成するには機能の分解が必要であり、そう簡単なものではない。そこでコンスタンチンは、機能分解を導出するためにプログラムグラフのアイデアを用いることにした。コンスタンチンは、このプログラムグラフをバブルチャートと呼んだ。これが構造化分析の始まりである。ゲイン（Gane）とサーソン（Sarson）は、協同してバブルチャートの拡張作業を行い、ファイルとターミネータを新たに付け加えた。また、データディクショナリ（DD）を導入した。データディクショナリには、データフロー図上で表現できない、処理の詳細説明、ファイル、レコードに関する説明を記述する。構造化分析では、データディクショナリにシステムが扱うデータの説明を場当り的に記述しており、後にシステムの安定的な特性であるデータを軽視しているとの批判を受けることになる。デマルコ（DeMarco）もゲイン／サーソンの構造化分析を普及させるとともに、データディクショナリにBNF（Backus-Naur Form）の記法を導入した。こうした関係で、ゲイン／サーソン法と呼ばれる構造化分析法やヨードン／デマルコ法と呼ばれる構造化分析法もルーツは同じであり、記号の書き方を除いて大きな違いはない[Ward91]。

1980年代に入ると、自動車や航空機へ組み込むシステム（Embeded System）の開発が多くなった。こうしたシステムは、従来の構造化分析が扱わなかった制御情報が重要になる。そのためワード（Ward）／メラー（Mellor）とハトレイ（Hatley）／ピアバイ（Pirbhai）は、ほぼ時を同じくして構造化分析のリアルタイム拡張を行った。ワード／メラー法[Ward85]では、従来のデータフロー図（DFD）にプロセスを制御するアクティビティや制御フローを付け加え、それを変換図（Transformation Schema）と呼んだ。アクティビティのロジック（ミニ仕様書）は、状態遷移図で記

述する。一方、ハトレイ／ピアバイ法[Hatley87]では、DFDの他にCFD（制御フロー図）をDFDとペアの形で設けている。また、プロセスの起動条件は制御仕様（CSPEC）に記述するようになっている。

図3.1-1に変換図の例を示す。また、図3.1-2に状態遷移図の例を示す。

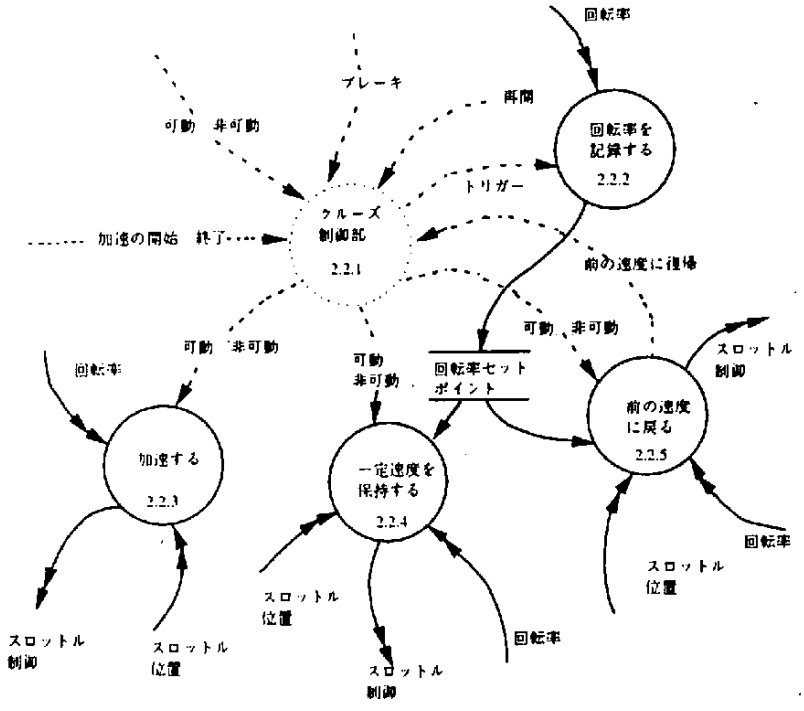


図3.1-1　変換図の例

こうした技法は、プログラムを作成したいという発想からスタートしており、まずシステムで扱う機能を求めそれを分解してモジュールを作るという機能中心の手法であった。

一方、1960年代後半になると従来のファイルシステムに代わってデータベース管理システムが登場し、データベースの構造をいかに設計するかが重要なテーマとなってきた。チェン（Chen）は、1976年にデータベースの論理設計を行う上で必要となるデータ分析の手法として、実体関連（Entity-Relationship）アプローチを発表した[Chen76]。この手法は、現実世界を実体と実体間の関連、及びその属性で示すもので、結果を実体関連図（Entity Relationship Diagram）としてまとめる。この手法は、データベース管理システムに依存しないデータベース設計方法論として発表されたが、データベース設計という小さい範囲にとどまらず、情報からの実世界モデル化技法として広く使われるようになった。図3.1-3はチェンの表記法に基づいた実体関連図であるが、その表記方法には様々なものがある。例えば、マーチン（Martin）はカラスの足の形をした記法でカーディナリティの表現を付け加えている[Martin85]。

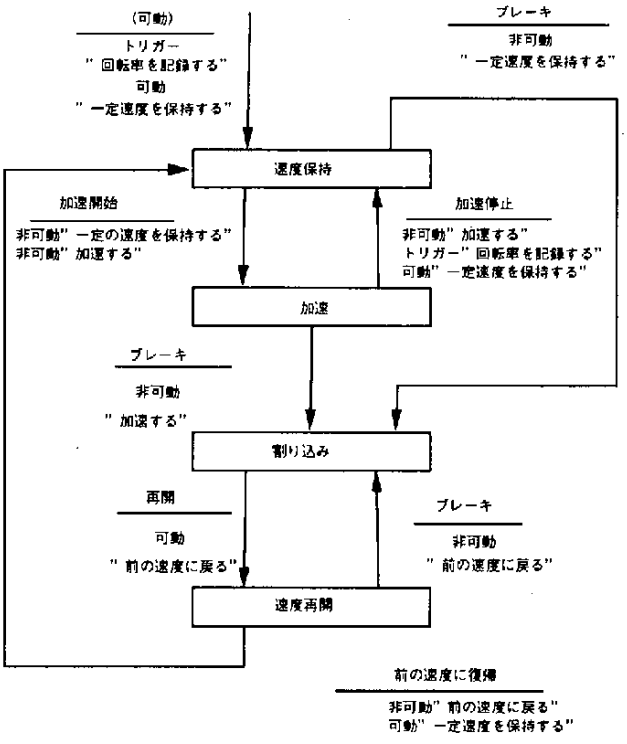


図3.1-2　状態遷移図の例

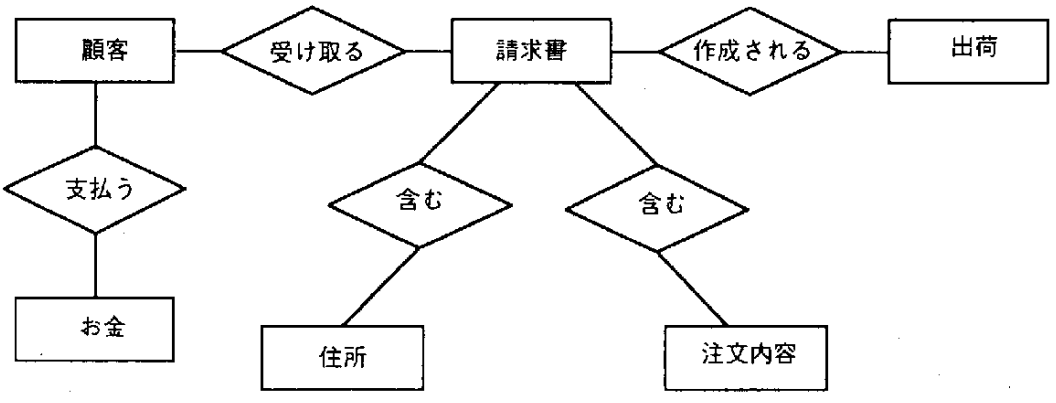


図3.1-3　実体関連図の例

(3)　オブジェクト指向分析設計で用いられる図形

1980年代の終り頃からオブジェクト指向と呼ばれる方法論が注目されるようになってきた。構造化分析／設計が機能を中心とした技法であり、実体関連技法がデータを中心としたものであったのに比べ、オブジェクト指向では実体をそのままモデル化しようとしている。即ち、データと手続きは組をなすもので、両者を組（オブジェクト）として分析／設計すべきであるという立場をとっている。

オブジェクト指向の特長として、抽象化、カプセル化、継承がある。抽象化は、複雑な世界を単純なモデルで表現し、理解の容易化を図ろうとするものである。問題の重要な側面、注目したい側面がモデル対象となる。構造化設計におけるトップダウン設計は、機能を分割していく方法であり、機能から見た抽象化である。

ジェームス マーチンらの情報工学[Martin89]で行われている、データのサブタイプ／スーパータイプといった分析も抽象化である。オブジェクト指向における抽象化もこのサブタイプ／スーパータイプの関係に近いが、機能の抽象化も同時に行われている。

オブジェクト指向ではデータと手続きが一塊となっていると述べたが、そのことをカプセル化という。オブジェクト指向のカプセル化では、手続き（メソッド）の呼出し（メッセージの送信）によってのみデータにアクセスできる。そのため、カプセル化は情報隠蔽とも呼ばれる。

オブジェクト間に包含関係を持たせ、下位オブジェクトが、上位オブジェクトの属性や振舞いの一部あるいはすべてを引き継ぐことを継承という。継承関係では、下位オブジェクトは上位オブジェクトの性質を引き継いでいるため、両者はis-aの関係であるとも言われる。なお、上位オブジェクトは下位オブジェクトから構成されているという集約関係もある。この関係は、全体対部分の関係であるからpart-ofの関係という。

今迄、オブジェクトという用語のみを用いてきたが、オブジェクト指向ではクラスと呼ばれる用語も用いている。クラスは同種のオブジェクトの集合として用いられる。

さて、オブジェクト指向での図形表現では、先に述べた、オブジェクト、抽象化、カプセル化、継承などをいかに表現するかが問題となる。

オブジェクト指向は現在発展中であり、様々な技法が開発されている。ここでは、ブーチ法[Booch91]、コード／ヨードン法[Coad90, Coad91]、OMT法[Rumbaugh91]、シュレィアー／メラー法[Shlaer88]を対比させながらどのような記法が用いられているかを見る。

(a)　オブジェクトの表現

ブーチ法では、クラスを形のない小塊（雲とも呼ばれる）で表わし、その中にクラス名を書く。クラスが持っている属性や操作はクラステンプレートとして別書きする。クラスのインスタンスであるオブジェクトは、小塊を破線の代わりに実線で囲む。ブーチ法では、フリーサブプログラムをクラスユーティリティと呼び、クラスの記号に陰影を付けて区別できる。

コード／ヨードン法では、クラスを縦長楕円で表現し、中にクラス名、属性名、メソッド名を書く。また、インスタンスを持つクラスは太線の枠の外側に細線の枠を付けることによって、インスタンスを持たないクラス（抽象クラス）と区別している。

OMT法とシュレィアー／メラー法は、ともに長方形でクラスを表わしている。OMT法では、長方

形の角を丸くすることによりクラスとインスタンスを区別している。

これら図形表現の違いを図3.1-4に示す。図からシュレィアー／メラー法は、他の技法と比較してメソッドを陽に定義しないことが分かる。

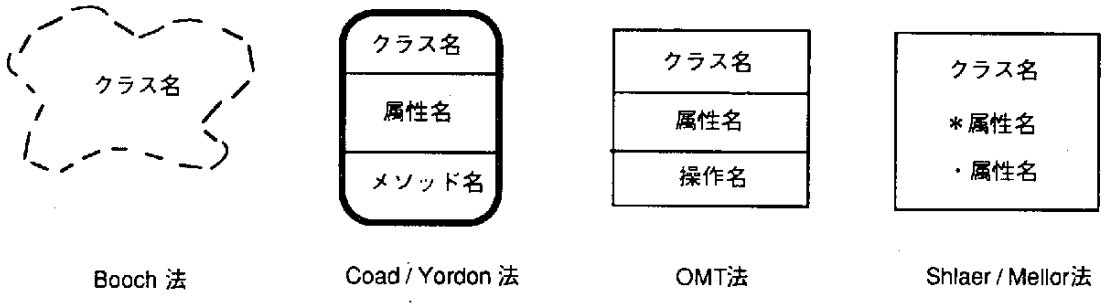


図3.1-4　クラスの図形表現

(b)　オブジェクトの静的構造表現

クラスとクラスの間には、先に述べたように、継承関係、集約関係、2項関係など様々な関係が存在する。こうした関係表現も技法により異なっている。

図3.1-5に継承関係の図形表現上の相違を示す。ブーチ法を除いてすべて同じような構造表現となっている。

2項関係は、実体関連アプローチで用いられている表現と同じになる。コード／ヨードン法は、オブジェクト間を実線で結ぶ簡素な図式表現になっているが、他の技法では、線の両端などに様々な修飾記号が書けるように工夫されている。一般に顧客というクラスと製品というクラスなどは多対多の2項関係を持っている。しかし、販売日という属性は顧客というクラスにも製品というクラスにも入らない。そうした場合、コード／ヨードン法では、新たなクラス（例えば販売というクラス）を設ける方法をとっている。しかし、OMT法では、それをリンク属性として区別し、新たな記号を設けている。図を簡素化することによりコミュニケーション能力は向上するが、厳密さに欠けることがある。この辺の問題は、技法の設計思想に係わるもので、図形表現も自ずと設計思想が反映されることになる。

(c)　オブジェクトの振舞いの表現

各オブジェクトは静的に存在するのみでなく、時間の経過とともに状態を変化させる。状態の変化を明らかにすることは、対象領域の理解、インプリメントにとって重要である。この動的振舞いの記述には、構造化分析／設計で述べた状態遷移図が多く用いられている。

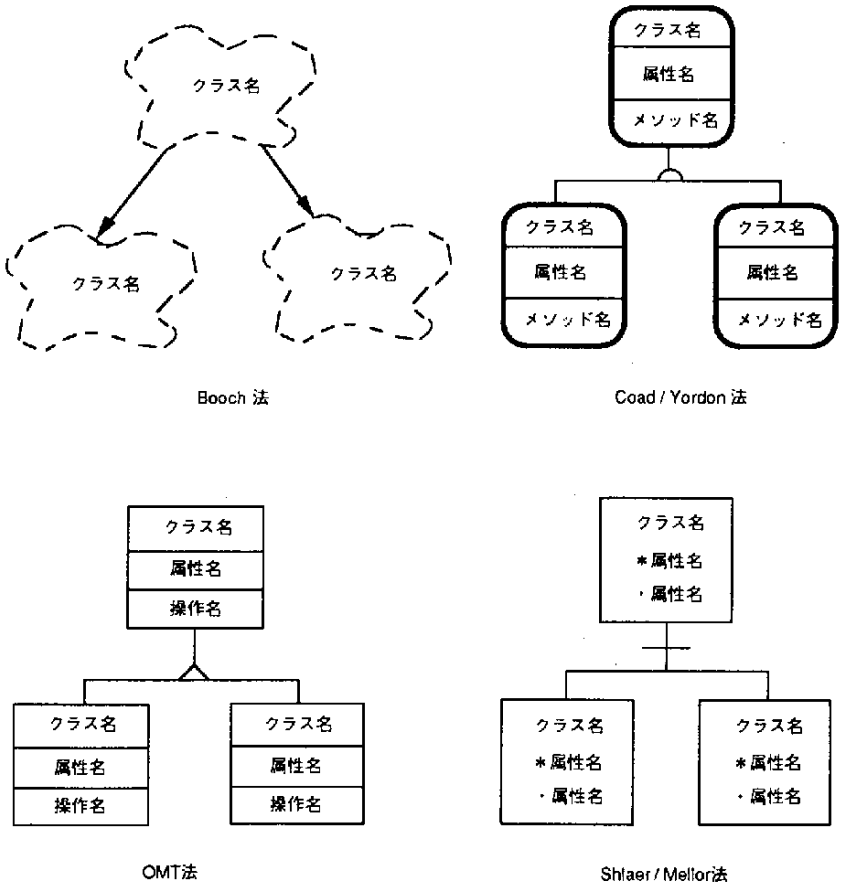


図3.1-5　継承関係の図形表現

(d)　オブジェクトの機能表現

オブジェクトが何を行うか、その機能的側面の記述には、構造化分析／設計で使われたデータフロー図が用いられる場合が多い。データフロー図で描かれたプロセスはメソッドになる。

以上、様々なオブジェクト指向の技法を図形表現の観点から眺めてきた。いずれも実体関連図を中心に、状態遷移図やデータフロー図をうまく活用していることが分かる。技法の優劣、図形表現の優劣は、対象としているシステムやその利用方法によって決定されるものと思われる。ユーザとのコミュニケーションを中心に考えるならば、コード／ヨードン法のような簡素な表現が適しているであろうし、インプリメントを重視するのであれば、ブーチ法のような記述方法が適しているであろう。

(4)　図形の相互変換

データフロー図、状態遷移図、実体関連図などを表現する図形シンボルは、技法によって少しずつ

異なっている。しかし、その構成要素（データフロー図でいうならば、プロセス、データフロー、データストア、ターミネータ）は同じであり、その相互変換は可能である。こうした図形データの交換のための標準形式として検討されているものとしてCDIFとSTLがある。

(a) CDIF

EIA（米電子産業協会）では、CASEのデータ交換のための標準化について検討を行っており、CDIF（CASE Data Interchange Format）[EIA91]と呼ばれる暫定規格を公表している。

CDIFで転送されるデータの定義は、データ、モデル、メタモデル、メタメタモデルの4階層からなっている。第1層のデータはシステムによって使われているデータそのものであり、第2層のモデルはデータフロー図や実体関連図で記述される情報である。第3層のメタモデルは、モデル情報を記述する規則をサブジェクトエリアごとに定義したものである。また、第4層のメタメタモデルは、メタモデルの構造や意味、制約を明示的に定義したものである。

CDIFの中心は、メタモデルであり、これは意味モデルと表示モデルに分かれている。意味モデルは、サブジェクトエリアの定義で、構成要素の定義（データフロー図でいえば、プロセス、データフロー、データストア、ターミネータの4つ）とその間の関係の定義である。表示モデルは、描画のための線の太さ、色などの静的な属性の定義と、箱の位置などの動的な属性の定義である。

こうした定義情報に従って、データフロー図や実体関連図などの図形仕様がASCIIコードに変換され、受取側では、図形仕様を再生成できる。

(b) STL

STL（Semantic Transfer Language）[IEEE91]は、IEEE P1175タスクフォースで検討された意味移転言語であり、1991年にIEEEのトライヤルユース標準となっている。

IEEE P1175では、実システムの情報を概念情報、局面情報、表現情報に分けて考えている。概念情報はアクション、データ、イベントなどの情報で、局面情報はいくつかの概念情報によって構成される情報である。即ち、局面情報とはデータフロー図、実体関連図などが該当する。表現情報は局面情報がいかに表現されるかを表したものである。STLは概念情報をテキストイメージで転送できるようにしたものである。

図3.1-6のデータフロー図をSTLで記述すると次のようになる（抜粋）。

```
Collection                          Diagram_A
    has label                       "STL data flow example";
    has external usage              "multi-level composite";
    has component collection        Diagram_A_upper,
```

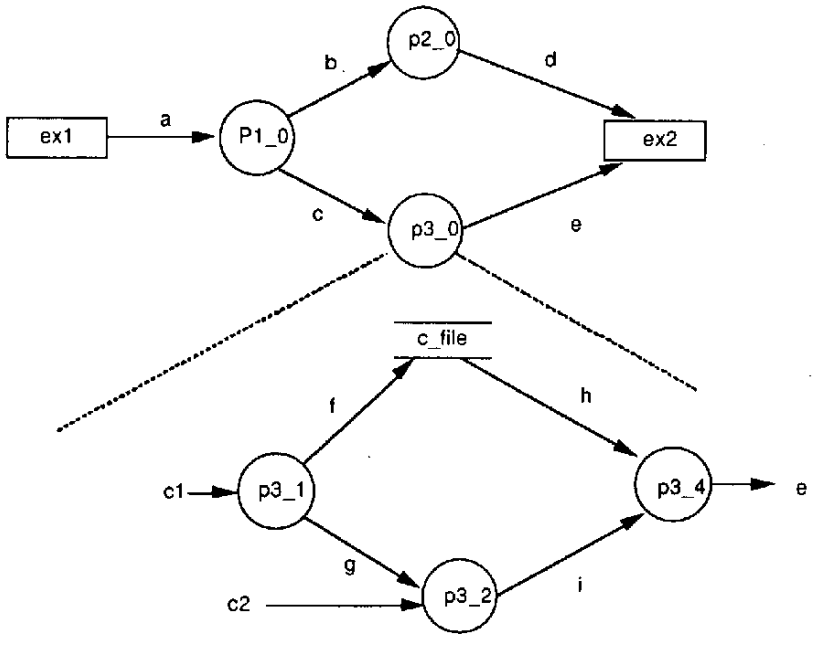


図3.1-6　STL記述のためのデータフロー図の例

	Diagram_A_lower;
Collection	Diagram_A_upper
is componet of collection	Diagram_A;
groups action	Process_p1_0,
	Process_p2_0,
	Process_p3_0,
	Process_ex1,
	Process_ex2;
groups connectionpath	Flow_a,
	Flow_b,
	Flow_c,
	Flow_d,
	Flow_e;
Action	Process_p1_0
has label	"p1_0";
is actiontype	internal;
has transform purpose	data;

has criticality	mandatory;
is connected from connectionpath	Flow_a;
is connected to connection	Flow_b,
	Flow_c;
is grouped into collection	Diagram_A_upper;
ConnectionPath	Flow_a
has label	"a";
is connectiontype	data;
connects from action	Process_ex1;
connects to action	Process_p1_0;
is grouped into connection	Diagram_A_upper;

3.1.2 コンピュータによる仕様記述支援

ソフトウェアの機能構造、動的な振舞い、手続きの記述には様々な図形仕様が用いられている。また、それらの記述は、分析、設計、製造と段階が進むにつれて精密なものになって行く。従来これらの多くの図形仕様作成は手作業を中心として行われていた。そのため、図そのものを作るのが大変な上、修正、整合性チェック、再利用には多大な労力を必要としていた。このような問題を解決するために登場してきたのがCASE（Computer-Aided Software Engineering）である。CASEは、ビットマップディスプレイを持つワークステーションが普及してきた1980年代後半に登場した。当初のCASEツールは、データフロー図や実体関連図を精密に描画する作図ツール的なものが多かったが、次第に分析から製造までの一貫した開発方法論を支援するものに発展してきている。ここでは、そうした一貫型支援ツールの代表例として米国テキサス・インスツルメンツ社のIEF（Information Engineering Facility）を取り挙げ、仕様のビジュアル表現によっていかにソフトウェアの開発がなされるかを見る。

(1) IEFの概要

IEF[TI89]は、ジェームス マーチン（James Martin）らによって開発された情報工学（IE）と呼ばれる開発方法論に基づいたツールで、開発段階に応じた計画ツールセット、分析ツールセット、設計ツールセット、構築ツールセット、インプリメンテーション・ツールセットの5つのツールセットから構成されている。各ツールセットは密に結合されており、各段階で作成される成果物はエンサイクロペディアと呼ばれるリポジトリに格納される。このエンサイクロペディアを用いて、ダイヤグラム間の関係チェック、下位ダイヤグラムの導出などが行われる。

(2) 方法論の説明

情報工学は情報戦略計画フェーズ、ビジネスエリア分析フェーズ、システム設計フェーズ、製造フェーズ、導入フェーズから成っており、各フェーズでは企業活動をデータとプロセスの観点から分析あるいは設計を行う。情報工学はBPS（ビジネスシステムプランニング）、構造化分析、構造化設計、実体関連分析などの技法を集大成したものとみることができる。また、企業戦略を支える情報システムの重要性やデータ重視の姿勢が色濃く含まれている。

(a) 情報戦略計画フェーズ

情報戦略計画フェーズは最初のフェーズであり、企業の目的、目標、重要成功要因等の分析を行う。このフェーズの作業は次の二つのサブフェーズに分けられる。

最高経営責任者による戦略計画サブフェーズには、目標と問題の分析、重要成功要因の分析、技術革新の影響分析、戦略システム計画のステップがある。

情報部門内最高責任者による分析サブフェーズには、企業の機能モデルの作成ステップがある。このステップで作成するモデル及び分析内容は、次のとおりである。

(i) 企業の全体モデルの作成

ここでは、企業の活動がどのように行われているかを調べるために部門、場所、機能、実体タイプ（データ）をリストアップし、それらの関係を明確にする。

(ii) 目標と問題の分析

目標（goal）と問題（problem)は、常に把握しておくことが必要である。そのためには、部長レベル以上の管理者にインタビューを行い企業の目標と問題を明確にする。目標は、しばしば問題の解決に関係している。また、目標と問題は関係している。

(iii) 重要成功要因分析

重要成功要因（CSF:Critical Success Factor ）は、目標を達成する上で重要となる分野を意味し、目標は最終的な到達点（end）を示す。

CSF分析を行うにはクリティカルな情報が必要であり、また、CSFはクリティカルな仮定の基に成り立っている。仮定が変われば自ずとCSFも変わることになる。CSFの遂行に当たっては、重要な意志決定が必要であり、そのために意志決定支援システムのようなものが必要となる。こうしたクリティカルな情報、仮定、決定を戦略データモデルとして具現化する。

(iv) 技術影響分析

技術革新の世界において、常に技術革新が経営にどのようなビジネスの機会を与えるか、またどのような脅威となるかを分析する。

(v) 戦略システム構想

戦略システムを考える方法論は、技術影響分析と同様である。

(vi) 企業データの分析

(i)～(v)で収集したデータを基に、機能と実体とのマトリックス、部門と実体とのマトリックス等を作成し、それらの相互関係を分析する。

(vii) 企業モデルのクラスタ化

企業と実体のマトリックスを基に、機能とそれに関係する実体のクラスタ化を図る。このクラスタ化されたものごとに次のビジネスエリア分析を行う。

(b) ビジネスエリア分析フェーズ

ビジネスエリア分析の目的はシステムを設計するのではなく、独立して開発されるシステムがお互いにフィットするかどうかを確かめるためのフレームワークを作ることである。

ビジネスエリア分析では、データモデルダイヤグラム、プロセス分解ダイヤグラム、プロセス相互依存ダイヤグラム、プロセス／データマトリックスを作成する。

(i) データモデルダイヤグラム

データのモデリングは、情報戦略計画フェーズで作成された実体関連モデルと現場で使われるドキュメント、スクリーン等のデータの分析を用いて行う。この両者を用いることにより、戦略計画に基づいたデータと現場での要求に基づいたデータを併せ持つ1つのデータモデルが作成できる。データモデルは、データの完全性、独立性等を保つために必要な正規化を行う。各フェーズとデータの関係は次のとおりである。

情報戦略計画フェーズ…… 全体的な実体／関連モデルの作成
↓ビジネスエリアごとに詳細化を行う
ビジネスエリア分析フェーズ… 完全に正規化されたデータモデルの作成
↓
システム設計フェーズ…… 完全に正規化されたデータモデルを用いたシステムの設計
↓
製造フェーズ…… 効率上の観点から必要に応じて再正規化を行う

(ii) プロセス分解ダイヤグラム

情報戦略計画フェーズで抽出した機能を詳細化する。この詳細化されたものをプロセスと呼ぶ。機能とは企業活動をグループ化したものであり、プロセスとは入力データと出力データにより識別される企業活動である。この詳細化は、木構造の階層性を持たせながら行い、基本的なプロセスが明確になるまで行う。

(iii)　プロセス相互依存ダイヤグラム

(ii)で詳細化されたプロセスは、お互いに依存関係を持っており、その関係を明確にする。このプロセス相互依存ダイヤグラムにデータを付け加えるとデータフローダイヤグラムになる。

(iv)　プロセス／データ・マトリックス

プロセス／データ・マトリックスは、先に作成したデータモデルとプロセスモデルを基に作成する。このマトリックス作成の主眼は、どのプロセスがどのデータを生成、参照、更新、削除しているかを明確にし、プロセス、データがすべて抽出されているか等をチェックすることである。

(c)　システム設計フェーズ以降

システム設計では、上記内容をさらに詳細化しコードの生成へとつなげる。

(3)　CASEによるビジュアルな開発

IEFにより実現されているシステム開発支援機能を図示すると図3.1-7のようになる。ここでは、情報工学の各開発段階ごとに支援している機能を説明する。なお、以下の説明中の①、②、…は、図3.1-7内の番号と対応している。

(a)　情報戦略計画フェーズ

情報戦略計画フェーズでは、4つのダイヤグラム作成と各ダイヤグラムで出現したオブジェクトのマトリックス作成がツール化されている。

①　組織構造図（Organizational Hierarchy Diagram）

組織構造図を作成する。この構造図で表された部門は、例えば、どの部門がどのデータを利用（生成、更新、削除）しているかを示すマトリックス等に自動展開される。

②　サブジェクトエリア図（Subject Area Diagram）

データ分析の最初のステップであり、企業活動で必要な基本的なデータの単位を図示する。このサブジェクトエリアは後に実体関連図に展開され、実体関連図はさらにデータモデルとなりDB設計につながる。

③　機能階層図（Function Hierarchy Diagram）

アクティビティ分析の最初であり、企業活動を遂行している機能を階層的に図示する。

④　機能依存図（Function Dependency Diagram）

機能は一般に依存関係を持っている。ここでは、③の機能を基にその依存図を作成する。

⑤　マトリックス（Matrix）

①～④の分析後、各々の相互作業の関係を示すマトリックスを作成する。マトリックスは約40種生成できる。

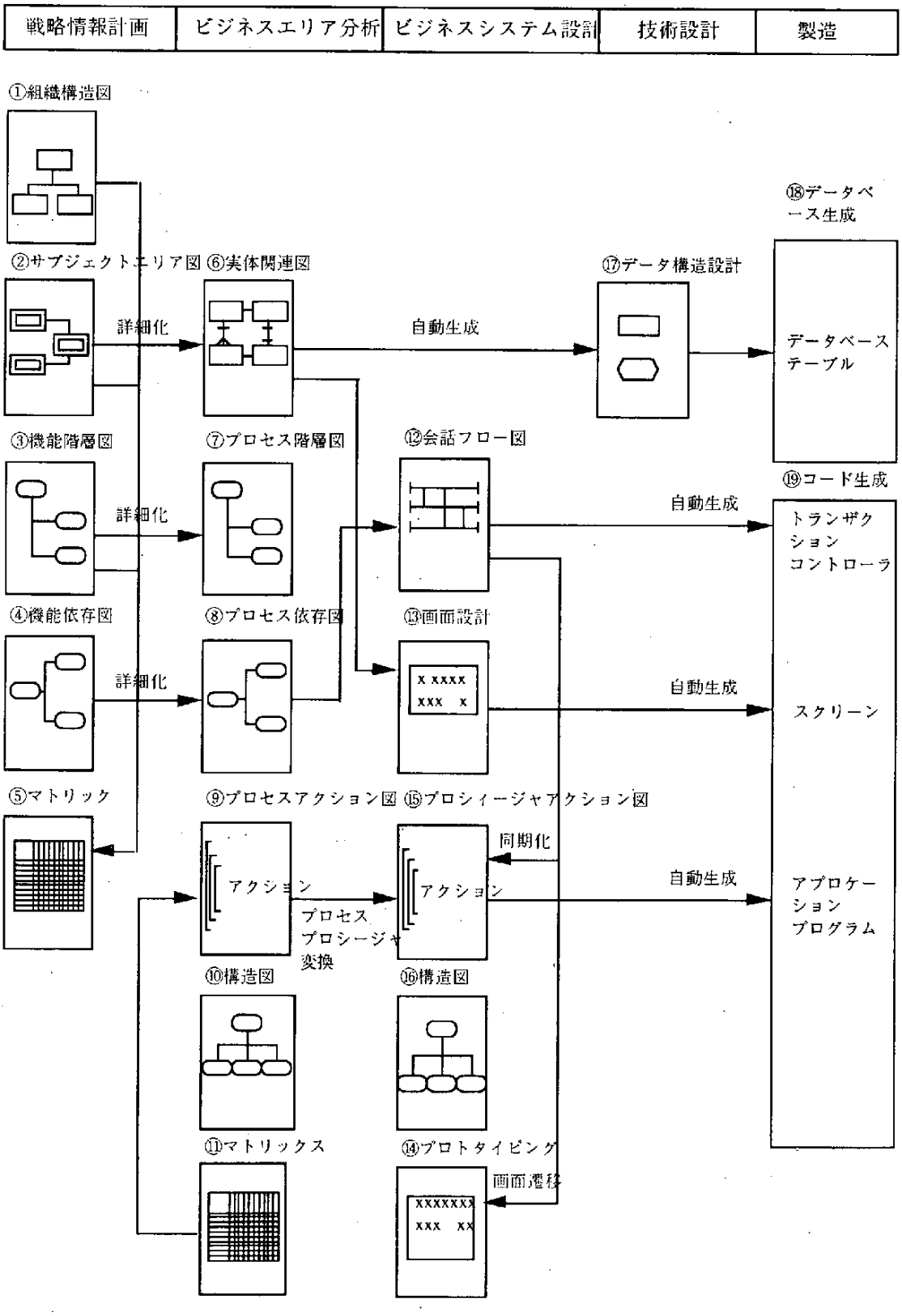


図3.1-7　システム開発支援機能

(b) ビジネスエリア分析フェーズ

⑥ 実体関連図（Entity Relationship Diagram）

②のサブジェクトエリアを詳細化し実体関連図を作成する。実体関連図では、実体と実体間の関連のみならず、1対1、1対多等のカーディナリティも示す。

⑦ プロセス階層図（Process Hierarchy Diagram）

③の機能をさらに分解する。この分解されたものをプロセスと呼ぶ。

⑧ プロセス依存図（Process Dependency Diagram）

プロセスの依存関係、トリガー、外部オブジェクトを図示する（このプロセス依存図にデータを加えれば、通常のデータフロー図となる）。

⑨ プロセスアクション図（Process Action Diagram）

アクション図は、プロセス内の処理をワーニエ風の記述法で示すものであり、基本プロセス（最下層のプロセス）に対して作成する。プロセスアクション図の多くの部分は、他のダイヤグラムやマトリックスの情報を基に自動生成される。

⑩ 構造図（Structure Chart）

プロセスアクション図内のアクションブロックの構造を図示する。

⑪ マトリックス（Matrix）

基本プロセスとデータ（エンティティタイプ）の相互作用等を示すマトリックスを作成する。

(c) ビジネスシステム設計フェーズ

⑫ 会話フロー図（Dialog Flow Diagram）

プロシージャ（一つ以上の基本プロセスが集まったもの）の遷移を図示する。これによりトランザクションコントロールプログラムが自動生成される。

⑬ 画面設計（Screen Design）

スクリーンペインティングの形式で画面を定義する。これにより画面定義体が生成される。

⑭ プロトタイピング（Prototyping）

⑫を基に画面遷移等のプロトタイピングを行うことができる。

⑮ プロシージャアクション図

プロシージャアクション図は、既に定義した情報から自動生成される。ここでは、異常処理等を付け加えることになる。

⑯ 構造図（Structure Chart）

構造図により、プロシージャと、それに含まれる基本プロセス、アクションブロックの関係が把握できる。

(d) 技術設計フェーズ

⑰ データ構造図（Data Structure Diagram）

データ構造図作成は、今迄進めてきたデータの論理設計を特定のデータベースを対象とした物理設計に変換することができる。この物理設計の大部分は自動的に行われる。ここで必要となる作業は、自動的に作成された物理設計のチューニングである。

自動設計された内容とチューニング内容の対応はIEFが持つため、チューニングによって今迄作成されたアクション図（プログラムフロー図）が影響を受けることはない。

(e) 製造フェーズ

⑱ データベース生成

データベースの定義体を生成する。

⑲ コード生成

プログラムとトランザクションコントロールプログラムを生成する。

(4) 仕様の関係付けと導出

システム化対象領域は様々な側面を持っており、単一の視点のみで抽象化することはできない。そのため情報工学では、データと機能の側面から問題領域のモデル化を図っている。また、そのモデル化は段階的詳細化の手順で行われる。各段階で作成される仕様は、前段階、後段階の仕様、同一段階における他の仕様と様々な関係を持っている。こうした各仕様間の整合性保持、仕様間の情報伝播は、CASEにとって極めて重要なことである。例えば、エンティティとプロセスの分析結果から、その関係を分析するためのマトリックスが自動生成される。このマトリックスにCRUD分析結果（C=生成、R=参照、U=更新、D=削除）を入力すると、プロセスアクション図が自動生成される。

3.1.3 仕様のハイパーメディア化

仕様のハイパーメディア化としてHyperWebとAMOREの例を示す。

(1) HyperWeb

ソフトウェア開発の各段階で作成される様々な中間成果物や最終成果物は、各々関連を持っている。例えば、プログラム構造図はいくつかのモジュールを含んでおり、各モジュールの機能は、設計ドキュメントで説明されている。また、各機能は具体的なコードとしてインプリメントされる。コードには注釈のドキュメントが添付されるかもしれない。このようにソフトウェア開発には様々な成果物が存在するわけであるが、各ドキュメントやソースコードファイルがハイパーテキストとしてリンク付

けられれば、参照、追跡が自由に行え、開発、保守の大幅な効率向上が期待できる。HyperWeb[Christel94]は成果物のハイパーメディア化を行うためのフレームワークであり、その中心はHyperWebサーバである。HyperWebサーバは、既存のツール（Epoch、FrameMakerなど）をクライアントとしてフロントエンドに持ち、バックエンドに成果物を格納管理するデータベースを持つ。データベースとしては、ヨーロッパのEspritプロジェクトで開発されたPCTEを利用している。図3.1-8にHyperWebの構成を示す。HyperWebメッセージサーバはツールの制御統合を可能にさせるもので、HyperLispインタプリタはノードやリンクの操作を行うCommon Lisp系のインタプリタである。HyperWebの特長は、PCTEというオープンな環境の上に構築されていることと、既存のツールを組み込むことができる点にある。

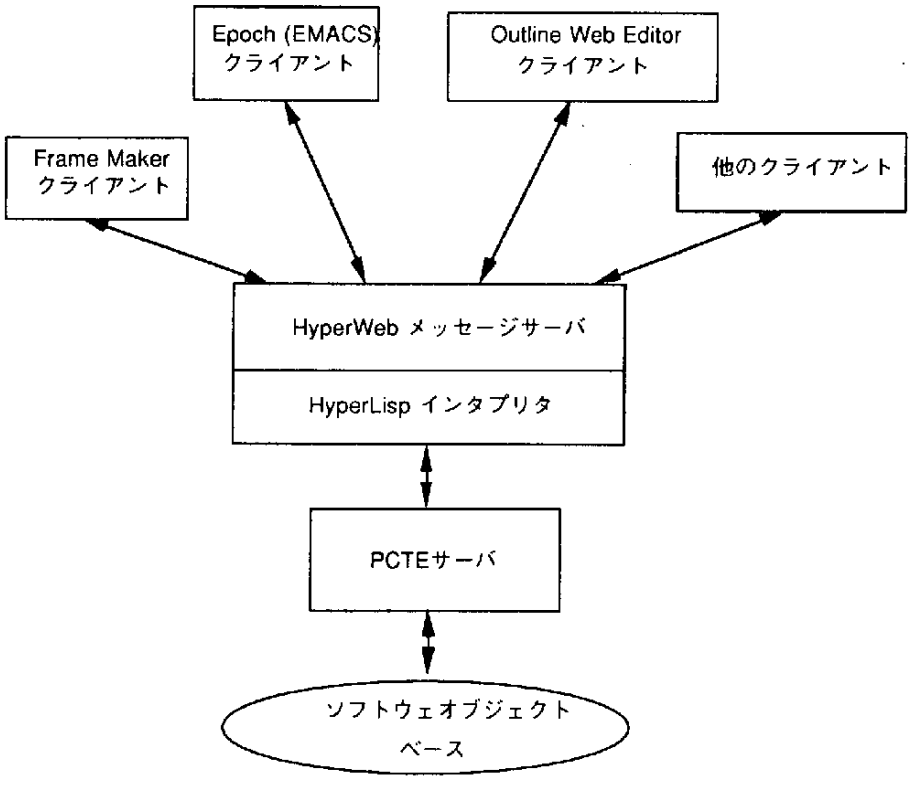


図3.1-8　HyperWebの構成

(2)　AMORE

企業の目的、目標の分析、業務規約の参照、エンドユーザとのインタビュー、技術情報の分析などを通して要求仕様の獲得が行われる。通常この要求仕様の獲得作業の結果は、フォーマルなドキュメントとして残るのみであり、なぜそのような結果になったかの論拠を示すものは残らない。その結果、要求仕様が正しく伝わらなかったり、仕様の再利用が困難になっている。こうした問題の背景には、

要求仕様獲得時点で作成されたり、集められた情報の媒体が様々でコンピュータでうまく扱えなかったことが挙げられる。しかし、コンピュータ技術の進歩により、テキスト、音声、ビデオなど様々な媒体が簡易に扱えるようになり、インタビューの模様を録画したビデオや図形情報などがそのまま扱えるようになってきた。AMORE（Advanced Multimedia Organizer for Requirements Elicitation）[Ferrans94]は、要求仕様獲得時に集められた様々な種類の情報を体系的に蓄積し、利用できるようにするシステムで、現在カーネギメロン大学でプロトタイプの研究開発が進められている。現状では、様々な可能性を検討している段階であるが、将来的には、事例ベース推論による検索、パターン認識、エージェントの導入などを考えている。

図3.1-9に、AMOREによる様々な表現形式の素材の表示例を示す。

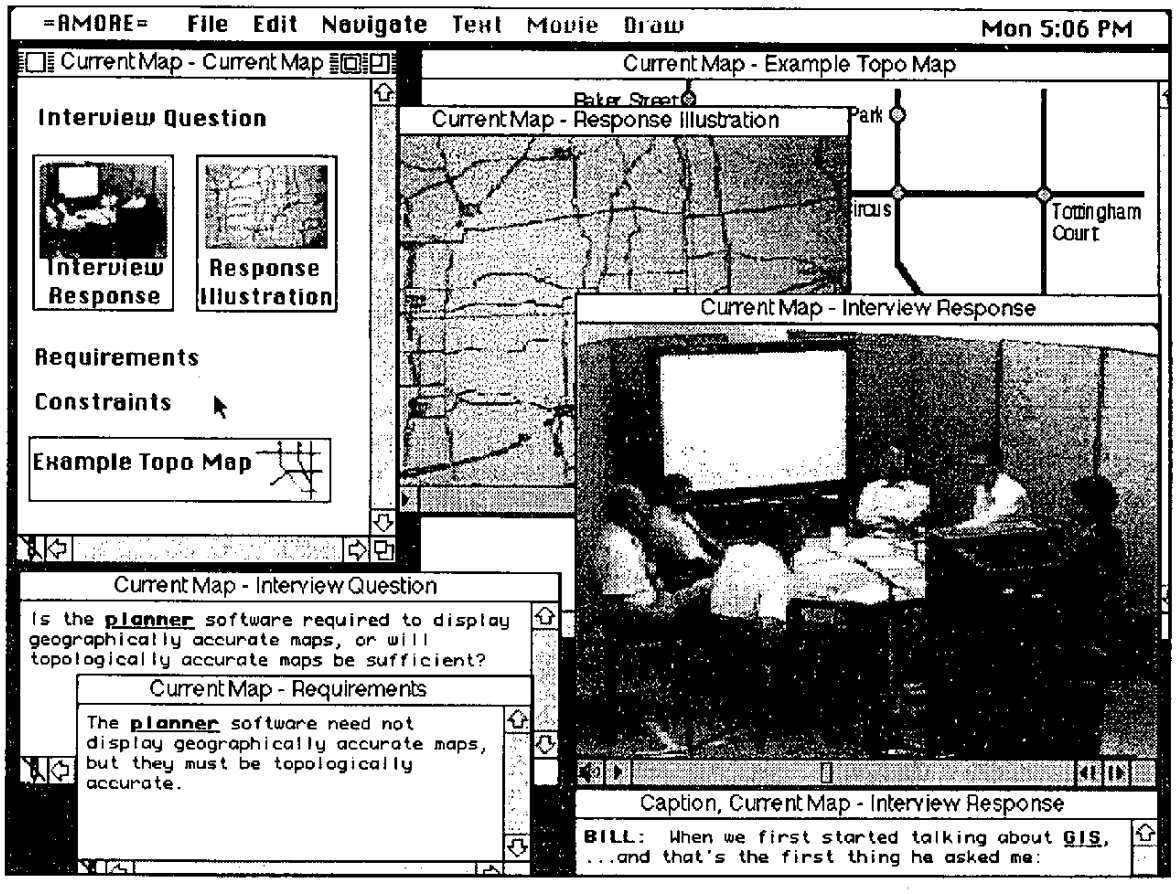


図3.1-9　AMOREによる表示例（出典：[Ferrans94]）

3.1.4　仕様の逆生成

ソフトウェア開発の各段階では、先に見てきたように様々な中間仕様が作成され、最終的にプログラムのソースコードとなる。これらの各中間生成物は、論理的に正しく、整合性がとれたものとなっている必要がある。しかし、保守時の時間的制約などから整合性が保てなくなるケースが多い。その

結果、ソースコード以外にその内容を説明するものがなくなり、機能修正が及ぼす影響範囲の分析、プログラムの再利用等が困難になってきている。こうした問題に対処するのがリバースエンジニアリングである。ソースプログラムから仕様を逆生成するツールとして様々なものが研究されているが、ここではRE-Analyzer[O'Hare94]を例に、いかに逆生成が行われるかを見る。RE-Analyzerの構成を図3.1-10に示す。

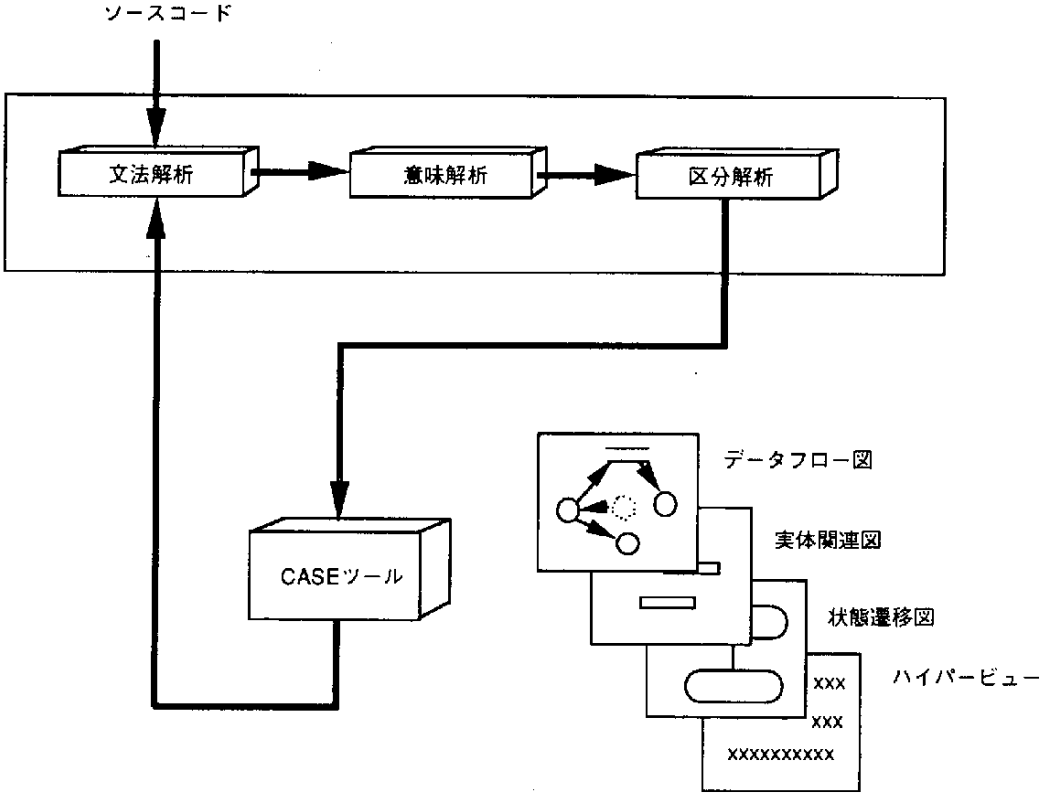


図3.1-10　RE-Analyzerの構成

RE-Analyzerに入力されたソースコードは、文法解析、意味解析、区分解析の工程を経て、等価なデータフロー図、状態遷移図になる。また、データ定義や様々なクロス参照表が作成される。

RE-Analyzerで生成される図形仕様は、初期のオートフローなどに比べて表示の階層化などが図られ、ユーザが理解しやすいように加工されているが、ソースコード以外に元データを持たないためその効果は限定されている。

ドメイン知識などを用いて、こうした問題に取り組んでいるDESIRE[Biggerstaff89,Biggerstaff94]などのシステムもあるが、いずれも研究段階で今度の発展が期待されている。

【参考文献】

[Biggerstaff89] Biggerstaff, T. J. : Design recovery for reuse and maintenance, IEEE Computer, 22, 7 pp.36-49 (1989).

[Biggerstaff94] Biggerstaff, T. J. et al. : Program understanding and the conceput assignment problem, Comm ACM, Vol.37, No.5, pp.72-83 (1994).

[Booch91] Booch G. : Object oriented design with applications, Benjamin/Cummings (1991).

[Chen76] Chen, Peter : The Entity-Relationship Model : Towards a Unified View of Data, ACM Trans. of DB, Vol.1, No.1, pp.9-36 (1976).

[Christel94] Christel M. G. et al. : A multimedia approach to reguirements capture and modeling, ICRE'94, pp.53-56 (1994).

[Coad90] Coad P. et al. : Object-oriented analysis, Prentice-Hall (1990).

[Coad91] Coad P. et al. : Object-oriented design, Prentice-Hall (1991).

[EIA91] CASE Data Interchange Format : Interim Standards, Vol. I, II, III, EIA (Electoronics Industries Association) (1991).

[Ferrans94] Ferrans J. C. et al. : HyperWeb : A framework for hypermedia-based environments, SIGSOFT'92, ACM Software engineering notes, Vol. 17, No. 5, pp.1-10 (1992).

[Hatley87] Hatley, Derek J. et al. : Strategies for Real-Time System Specification, Dorset House Publishing(1987) (立田種宏監訳：リアルタイム・システムの構造化分析, 日経BP社, (1989)).

[IEEE91] IEEE P1175 Trial-Use Standard, A Standard Reference Model for Computing System Tool Interconnections (1991).

[Martin85] Martin, James et al. : Diagramming Techniques for Analysts and Programmers, Prentice-Hall (1985).

[Martin89] Martin, James : Information Engineering, Book I, II, III, Prentice Hall (1989).

[O'Hare94] O'hare. A. B. et al. : RE-Analyzer : From source code to structured analysis, IBM systems journal, Vol. 33, No. 1, pp.110-130 (1994).

[Rumbaugh91] Rumbaugh J. et al : Object-oriented modeling and design, Prentice-Hall (1991) (羽生田監訳 : オブジェクト指向方法論OMT, トッパン (1992)).

[Shlaer88] Shlaer S. et al. : Object-oriented systems analysis, Prentice-Hall (1988) (本位田訳 : オブジェクト指向システム分析, 啓学出版 (1990)).

[TI89] IEF Technology Overview、Texas Instruments(1989).

[Ward85] Ward, Paul T. et al. : Structured Development for Real-Time Systems, Prentice-Hall (1985).

[Ward91] Ward, Paul T. : The Evolution of Structured Analysis : Part 1, American Programmer, Vol.4, No. 11 (1991).

3.2 ソフトウェアの動的視覚化

3.2.1 はじめに

効率良くソフトウェアを開発するためには、高品質なソフトウェア要求仕様が必須となる。ここでは要求の書きやすさ・読みやすさと正しさの向上を目標としたビジュアルな要求仕様化技法を紹介する。技法により、①ビジュアルな要求の記述と、②シナリオによるビジュアルな要求記述の動的視覚化が可能となる。

ソフトウェア開発の上流工程を支援するCASEツールのほとんどは、データフロー図（DFD）の描画機能を備えている[Fisher88]。DFDではデータの流れ、プロセス、ファイル、データ源泉とデータ吸収を表すために4つの図形が用いられる[DeMarco78]が、場合によっては、用意された4つの図形以外に、例えば、フローチャートのシンボルといった異なった図形や、頭に描いたイメージをそのまま表現するアイコンを用いたほうが素直に要求を表すことができる。

ビジュアルな要求仕様化技法の第一の特長は、データや制御の流れに関するソフトウェア要求を要求定義者の頭に描いたイメージにできるだけ忠実に表せることである。要求に現れる実体を任意の形状のアイコンとして定義し、実体間の関連を矢印で表し、それらをエディタ上で配置することによって、要求を仕様化する。

第二の特長は、要求仕様の誤りの検出ができることである。定義したアイコンの意味は我々が開発した要求フレームモデル[大西90]に基づいて定義される。要求フレームに基づいた要求記述の正当性検証手法については既に確立している[大西90,大西92]。

ラピッドプロトタイピングを用いたエンドユーザによるプロトタイプの使用経験は要求の確認に有効な手段であるが、第三の特長はアイコンの動作を表したシナリオを用いて、要求仕様を動的に視覚化できることである。

最初に要求言語の基礎を与える要求フレームモデルについて説明する。次にビジュアルな要求言語と仕様化技法について説明する。さらに、ビジュアルな要求言語の処理系について述べる。

3.2.2 要求フレームモデルと要求言語

開発してきた要求言語にはビジュアルな要求言語の他にも日本語要求言語があるが、これらは要求フレームモデル[大西90]に基づいている。このモデルは要求記述の枠組を与えるものであり、記述の構成要素に応じて、

名詞レベル：名詞と名詞の型

動詞・形容詞レベル：動詞・形容詞と動作概念

単文レベル：文と格フレーム

機能レベル：文章と機能フレーム

のそれぞれについての対応関係を定めるものである。

名詞は「人間」、「機能」、「データ」、「ファイル」、「制御」、「装置」のいずれかの型に分類される。

一般の文章で使われる表層の動詞は「データの流れ」、「制御の流れ」、「and木構造」、「or木構造」、「データ作成」、「レコード検索」、「レコードの更新」、「レコードの削除」、「レコードの挿入」、「ファイルの操作」の10種の深層の動作概念のいずれかに分類される。例えば、「(データを) 入力する」、「(データを) 渡す」、「表示される」といった動詞は、すべて「データの流れ」という概念に分類される。10種の概念に当てはまらない動作概念は記述者が新規に定義できる。また、大小関係などの比較を表す形容詞も6つの動作概念のいずれかに分類される。

格フレームは動作概念と必須格に対する枠組である。動作概念によって必須格は異なる。例えば、「データの流れ (DFLOW)」の必須格は動作主、源泉、目標、道具の4つであり、それぞれデータ型、機能か人間型、機能か人間型、装置型の名詞が当てはまる。このフレームを用いて必須格の抜けや格に当てはまる名詞の型誤りを検出できる。さらに、代名詞が使われたり、格が省略された場合に、文脈から用いられるべき名詞を推定できる。また、新規の動作概念を定義することもできるが、その場合は、新規概念の格構造も併せて定義しなければならない[大西92]。

日本語要求言語の場合、複文や重文も記述できるが、これらは動作概念を一つしか含まない単文に分割されてから、格フレームに基づいて解析される。

機能フレームはシステムが備えるべき一般的な性質を規定するものであり、「外部からの入力と外部への出力は、それぞれ少なくとも一つ存在する」、「作成されたり検索されて得られたデータは一度は参照されなければならない」などの10個の性質について、それらを要求記述が満たしているかをチェックするために用いられる[大西90]。機能フレームによって機能単位の抜けや矛盾を検出できる。

2つの要求言語による記述は、格フレームに基づいた共通の内部表現CRDに変換される。したがって、「データや制御の流れといったビジュアルに記述しやすい部分をVRDLで、データ構造や機能内容といった部分は日本語要求言語で」というように、用いる要求言語を切り替えて仕様化できる。

3.2.3 ビジュアルな要求言語：VRDL

VRDL (Visual Requirements Description Language) [Ohnishi94]の目標を以下に示す。

① データや制御の流れに関するソフトウェア要求を、要求定義者の頭に描いたイメージにできるだけ忠実に表せるようにする。

② ビジュアルな要求を記述した本人だけでなく、他の人間にも分かるようにする。

③ 要求フレームによる誤り検出以外に、ビジュアルな要求仕様におけるデータの流れを動的に視覚化することによって誤りを検出する。

VRDLの機能的な特長は以下に示すとおりであり、特長の1、2、3によって目標①を、特長の4によ

って目標②を、特長の5によって目標③を達成している。

1. アイコンの形状と意味を定義できる。
2. 複合的なアイコンを容易に作成できる。
3. アイコンと矢印をエディタ上で配置していくことによって要求を記述する。
4. VRDLによる記述を標準的なアイコンを用いた記述へ変換できる。
5. VRDL記述に用いられたアイコンの動作を与えることによって、記述を実行できる。

(1) アイコンの定義

DFD[DeMarco78]やSADT[Marca88]に代表される要求定義用のビジュアルな言語では、利用可能なアイコンの形状と意味は予め定まっている。DFDはデータの流れを名前付きの矢印で、機能を円で、ファイルを直線で、データの源泉と吸収を四角形で表し、図形の種類が少ないので覚えやすい。しかしながら、能大式の業務フロー図[情報処理学会80]のように30以上の多種のアイコンを使う図に慣れた人にとっては、DFDは単純化しすぎて使いにくく、アイコンの種類が少ないので名前や説明を詳細に文章などで記述しなければならない。また、使えるアイコンが限られるために、例えば、ファイルを直線でなくJISの情報処理用流れ図記号の直接アクセス記号[日本規格協会92]で表現したくてもできない。

記述者にとっては、自分のイメージにあった記号をそのまま要求記述に用いることができるならば、要求を記述しやすいし、また理解しやすい。このためには自分でアイコンの形状を定義して要求記述に用いることができるようにすればよい。一方、要求記述は記述者以外にも設計者など開発に携わる人によって参照される。他人の描いた図を理解するには、そこで使用されたアイコンの意味を的確に把握する必要がある。記述者以外の人がアイコンを別の意味にとると要求が正しく理解されない。

このためVRDLでは単にアイコンの形状を定義できるだけでなく、要求フレームモデルに対応して、要求記述に現れる名詞や名詞の型をアイコンの意味として定義できる。表3.2-1にアイコン定義例を示す。

表3.2-1 アイコン定義例

アイコンの形状	アイコンの意味
	人間型
	「ファックス」、装置型
	「メッセージ」、データ型

表3.2-1の最初のアイコンは人間型の名詞の総称を意味する。このような総称的なアイコンには、記

述に表すたびに異なる名前を名称として与えることができる。2番目のアイコンは装置型の「ファックス」を意味する。このように利用者が形状と意味の両方を明示することによって定義されるアイコンを基本アイコンと呼ぶ。基本アイコンからは複合アイコンを作成できる。

(2) 複合アイコンの作成

基本アイコンから複合アイコンを作成するために、基本アイコンに適用される操作として、``mulitiplying''と``composing''を導入する。複合アイコンの形状と意味は基本アイコンと操作から自動的に定まるので、利用者は定義しなくてよい。

``mulitiplying''は2つ以上の基本アイコンを表す複合アイコンを作成する操作である。例えば、図3.2-1のアイコンを「ファイル型で1枚のフロッピィディスク」を表す基本アイコンとし、これにmultiplyingを適用することによって図3.2-2に示す複合アイコンが得られ、その意味は「ファイル型で2枚以上のフロッピィディスク」と自動的に定まる。

図3.2-1 フロッピィディスクのアイコン

図3.2-2 フロッピィディスクのmultiply化

``composing''は2つの異なる基本アイコンを組み合わせた複合アイコンを作成する操作である。組み合わせる基本アイコンによって、得られる複合アイコンの意味は表3.2-2のように異なる。``composing''による複合アイコンの形状は2つの基本アイコンを括弧でくくったもので表す。

利用者はこれらの操作を用いることにより、容易に複合アイコンを作成できる。複合アイコンは基本アイコンと同様にビジュアルな要求記述に用いることができる。

(3) アイコンと矢印による記述

定義したアイコンと矢印を配置していくことによって、要求を記述する。アイコンと別のアイコンとの間の矢印で流れを表すが、VRDLではデータの流れを表すのに実線矢印を、制御の流れを表すの

表3.2-2　composingによる複合アイコン

基本アイコンの意味	複合アイコンの意味
A,B (同じ型の名詞)	A と B
A (機能か人間) B (装置)	B を用いて A
A (データかファイル) B (装置)	B を通して A

に破線矢印を用いている。要求フレームモデルの動作概念の中で特に流れを選んだ理由はビジュアルに表しやすいと考えたからである。流れ以外の動作を含んだ要求は日本語要求言語によって表すことができる。

例えば、「富市さんからBillさんへメッセージをファックスで送る」という要求をDFDで表すと図3.2-3のようになる。同じ要求を表3.2-1で定義したアイコンを用いてビジュアルに表したものを図3.2-4に示す。図3.2-3と比較すると、人の頭部の輪郭をアイコンとして用いることによって富市さんやBillさんが人間であることが直感的に理解できる。このように適切なアイコンを用いることで、要求をより分かりやすく、また頭の中のイメージに近い形で表すことができる。

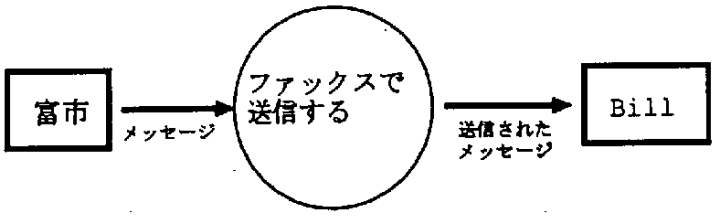


図3.2-3　DFDによる要求例

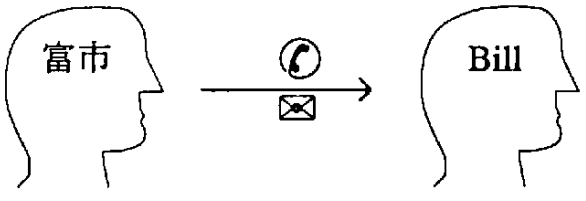


図3.2-4　ビジュアルな要求例

VRDLでは要求フレームモデルにおける動作概念の必須格にアイコンを、概念に矢印を対応させて、

要求文をビジュアルに表している。したがって、各々の要求文は格フレームを正確に反映している。同じ要求を日本語要求言語によって「富市さんがBill氏へメッセージをファックスを使って送る」と表しても同一の内部表現に変換される。

このようにして、要求は日本語でもビジュアルな言語でも表される。それぞれの要求記述が内部表現に変換されてから統合され、全体として誤りがないか機能フレームによって解析される。

(4) ビジュアルな要求記述の標準化

ビジュアルな要求記述を記述者以外の人が読む場合、例えば、表3.2-3の左上のアイコンを記述者は「ファックス」のつもりで用いたのに読者は「電話」と誤って解釈するかもしれない。

この問題は標準化されたアイコンの導入により解決できる。例えば表3.2-3のように、記述で使用される名詞や動詞に対応する標準化されたアイコンを予め用意しておき、それらを用いた記述に変換する。これにより、記述者が自分で定義したアイコンを用いても、標準化アイコンによる表現に変換されるため、読者は標準化アイコンの意味を予め知っておくことにより正しく理解できる。

表3.2-3 記述者のアイコンと標準化アイコン

意味	記述者	標準化アイコン
「ファックス」装置型		
「電話」装置型		

(5) ビジュアルな要求記述の動的視覚化

要求記述を解釈実行することによって動的視覚化し、利用者に記述が正しく書かれているかどうかを確認できる。ここでの動的視覚化とは具体的にはVRDLによる記述からデータフローに関する記述を抽出し、さらにそこで用いられている動作可能なアイコンに対して、動作記述（scenario）を利用者が与えることによって、データフローの様子をアニメーションとして表示させることを指す。

このため動作を記述する言語を開発した。この言語は以下の項目を記述できる。

- 動作させるアイコンの指定
- アイコンを表示する位置の指定
- アイコンの表示開始
- アイコンの表示終了
- アイコン表示速度の変更

アイコンの動作は、

1. アイコンの移動

2. 少しずつ異なる複数アイコンの表示の切り替え

によって実現している。例えば、「データが渡される」という動作を、源泉格から目標格のアイコンに向かって、データを表すアイコンを座標を少しずつずらしながら表示したり消去したりすることを繰り返すことによって実現する。また、「電話が鳴る」という動作を、受話器が少し持ち上がったアイコンと平常の電話のアイコンとを切り替えて表示することによって実現する。さらに、アイコンの移動と異なる複数のアイコンの表示の切り替えを組み合わせることによって、複雑な動作を実現する。

3.2.4 ビジュアルな要求記述の処理系

提案したビジュアルな要求仕様化技法の有効性を確認するためにプロトタイプを試作した。プロトタイプは、

・ビジュアルな要求の記述と解析系

・ビジュアルな要求記述の動的視覚化系

に分けられる。

(1) アイコンの定義

基本アイコンの形状については、X Window Systemのbitmapコマンドといった既存のツールを用いて描いたり、スキャナなどの装置を用いて、bitmapデータとして定義する。プロトタイプではアイコンの縦横のサイズは固定としている。アイコンデータの格納されたファイルとその意味をアイコン辞書に登録することにより基本アイコンを定義する。現在のところ複合アイコンはサポートしていない。

(2) ビジュアルな要求の記述と解析系

次に、定義したアイコンをエディタ上に配置する。図3.2-5にビジュアルな要求記述エディタの画面を示す。画面の左には、予め定義した名詞のアイコンが型ごとに分類されて表示される。最初に動作概念としてデータ又は制御の流れをマウスで選択すると、その動作概念の格構造に基づき、必須格に相当する名詞アイコンを次々と選択するように促され、マウスでアイコンとその配置位置を指定しながら、記述を進めていく。記述中の1つの実線矢印が1つのデータの流れに対応し、矢印の始点はデータの流れの源泉格に、終点は目標格に相当する。矢印の途中に置かれるアイコンは流れるデータと装置に相当する。これら4つの必須格に相当する名詞アイコンと1つの実線矢印でもって、1つのDFLOW文が表される。

例えば、表3.2-4のようにアイコンが定義されているならば、記述の最左の部分は「分析者は顧客に電話で資料を送る」ということを表す。各々の文は格フレームに基づいており、日本語要求言語と共通の内部表現に変換される。さらに次々とアイコンや矢印を配置していくことによって、複数の文に

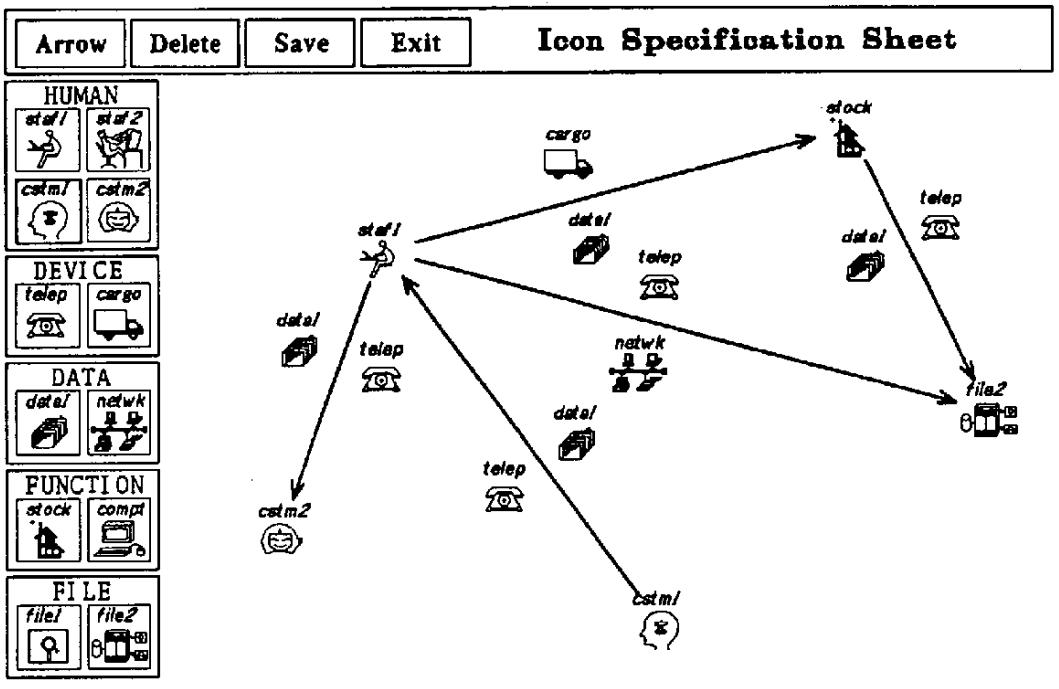


図3.2-5　ビジュアルな要求記述エディタ画面

表3.2-4　記述中のアイコン定義

アイコンの形状	意味	
	名称	名詞の型
	顧客	人間
	電話	装置
	資料	データ
	分析者	人間

相当する記述を表すことができる。

同じ要求をDFDで表すことはできるが、その場合はバブルや矢印に付けるラベルに細かい説明が必要となる。本手法では、記述者が自分で定義したアイコンを自分で指定した位置に配置しながら、頭の中のイメージにより近い形で要求をビジュアルに記述できる。

(3)　ビジュアルな要求記述の動的視覚化

VRDLによる記述において、データの流れ（DFLOW）の源泉格と目標格は動作しないと仮定し、

データに相当する動作主格と道具格に対してそれらの動作記述を与える。動作記述中のアイコンは、VRDL記述で用いたアイコンと動作用に新たに定義したアイコンが利用できる。動作記述では、どのアイコンを、どの位置に、どの程度の時間間隔で、表示・消去するかが記述される。この動作記述を解釈実行することにより、源泉格から目標格へのデータの流れが一文ずつ逐次アニメーションとして表示される。

図3.2-5に対する動作記述の一部を図3.2-6に示す。この例では、図3.2-7に示す電話に対してビジュアルな要求記述で使われたアイコンと、図3.2-8に示すアニメーション表示のために新たに定義したアイコンの2つを切り替えながら表示することによって、電話が鳴るという動作をアニメーションで実現している。

また、動作記述でアイコンの移動の始点、終点、時間間隔を指定することによって、そのアイコンの移動をアニメーションとして表示できる。図3.2-9～3.2-12はアニメーションのスナップショットを示したものである。アニメーション化に際して、1つのDFLOW文は10面のスナップショットに対応させている。

動作記述言語を解析し、動作記述を実行するプロトタイプを作成した。このプロトタイプは動作記述をC言語とX Window Systemによるプログラムに変換し、これをUNIXワークステーション上のX Window環境下で動作させている。「電話が鳴る」、「プリンタに出力される」といった頻繁に記述されるような装置の動作については、ライブラリ化して陽に指定しなくてもすむようにしている。

現在は動作記述を用意しなくてもすむように技法を研究中である。具体的には、状態遷移で表された動的な振舞いのモデル（動的モデル）が明らかになっている場合は、動的モデルにおける状態遷移を順序付けすることによって動作記述を自動生成する。

3.2.5 関連研究

関連した研究として、[Hsia88]ではシナリオに基づいたプロトタイピングと画面ベースのシナリオ生成手法が提案されている。生成されるのはシナリオに基づいたディスプレイ画面のシーケンスであり、これによって人間と計算機のやりとりが正しいかどうかを確認するものである。我々の技法では確認の対象はデータの流れに関する要求であり、流れはアニメーション表示されて、その向きや順序が正しいかどうかを確認する。

[Chang89]ではSIL-ICONと名付けられたビジュアル言語コンパイラが提案されており、アイコンの構文と意味、並びに言語の文法を定義できる。我々の技法ではアイコンの意味は要求フレームに基づいており、複合アイコンの意味は自動的に定まる。

[St-Denis90]ではアイコンのアニメーション化によるプロトタイピング手法が提案されている。アイコンは初期状態と終了状態の形状が定められ、初期状態から終了状態までの動きは垂直方向か水平

```
tel_anime(ACT src, ACT goal, ACT data)
{
    ACT tel1, tel2;
    int i;
    ICON icon1, icon2;
    LINE line;

    icon1 = ''telep1'';
    icon2 = ''telep2'';

    tel1 = {icon1, icon2};
    tel2 = {icon1, icon2};

    line = LINESTYLE1 ;

    locate(tel1, 50, 50);
    locate(tel2, 450, 50);

    actfor(i=0;i<11;++i){
        locate(data, 50 + 40*i, -50);
        if(i-i/2*2 == 0){
            display(tel1[0], ON);
            display(tel2[1], ON);
        }
        else {
             display(tel1[1], ON);
             display(tel2[0], ON);
        }
        display(line, ON);
        display(data, ON);
        display(src,  ON);
        display(goal, ON);
    }
    display(tel1, OFF);
    display(tel2, OFF);
    display(line, OFF);
}
```

図3.2-6　動作記述（部分）

図3.2-7　電話アイコン

図3.2-8　電話アイコン（ringing）

方向の移動か色の塗り潰ししかない。我々の技法ではアイコンの動きは任意の方向への移動と、形状の異なる複数のアイコンの切り替え表示によって実現され、しかも動作速度を指定できるため多彩な動きが表現できる。

3.2.6　おわりに

ビジュアルな要求仕様化技法とそれに基づいた処理系について紹介した。DFDのような記法に比べると、頭の中のイメージにより近く仕様化でき、VRDLと日本語要求言語とを場合に応じて使い分けることによって、要求記述の書きやすさや読みやすさが向上すると考えている。さらにビジュアルな要求記述の動的視覚化によってデータフロー要求の正しさの確認ができる。

我々は要求定義のための環境としてCARD（Computer Aided Requirements Definition）を開発している[Ohnishi93]。CARDは要求記述解析系、要求記述精製系、設計支援系から構成されているが、要求記述解析にはVRDLのエディタの他に日本語要求言語解析系があり、要求は日本語かビジュアルな言語のいずれかにより記述できる。したがって日本語で書きたい部分とビジュアルに書きたい部分を切り分けて、より書きやすい言語を用いて記述できる。どちらの言語による要求記述も、解析の結果として同一の内部表現に変換されて、要求記述精製系や設計支援系で利用できる。ビジュアルな要求記述の実行系のCARD環境への統合は今後の課題である。

また、VRDLのエディタの改善点として以下のものが挙げられる。

・曖昧さの解消：新たにアイコンを配置する際に、過去に配置したアイコンに重なる場合や、矢印どうしが近過ぎると、配置したアイコンが2つの矢印のどちらの格に相当するかが（人間にとっては）曖昧になる場合が生じており、指定した位置から重なりや曖昧さが生じる場合を排除して配置する工夫が必要である。

・アイコンサイズの変更：アイコンの大きさは同じとしているが、視認性の向上のためには、場合

に応じて大きさを変化させる工夫も必要である。

・アイコンのカラー化：同じく視認性の向上のためにアイコンをカラー表示する。

・構造的な記述：複雑な要求のためにDFDのような構造的な記述をサポートする。

・複合アイコンのサポート。

・標準アイコン：ファイルやディスプレイの形状のアイコンなど、標準的なアイコンをシステム側で予め用意する。

これらの問題点の解決と動作記述の作成支援手法の確立、並びにソフトウェア開発への適用を今後進めていきたいと考えている。

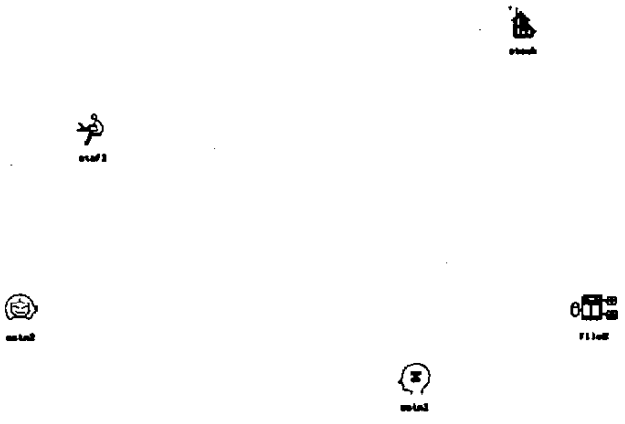


図3.2-9　アニメーションのスナップショット(1)

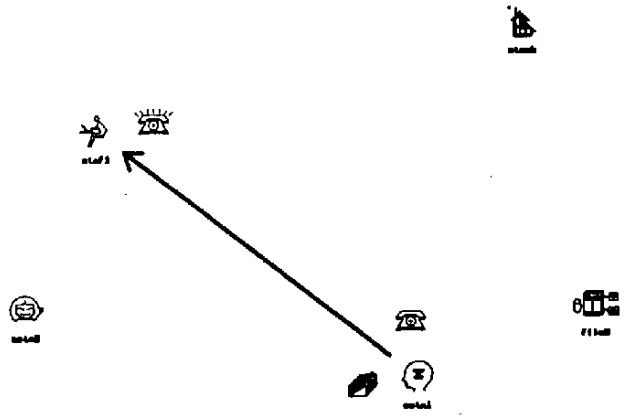


図3.2-10　アニメーションのスナップショット(2)

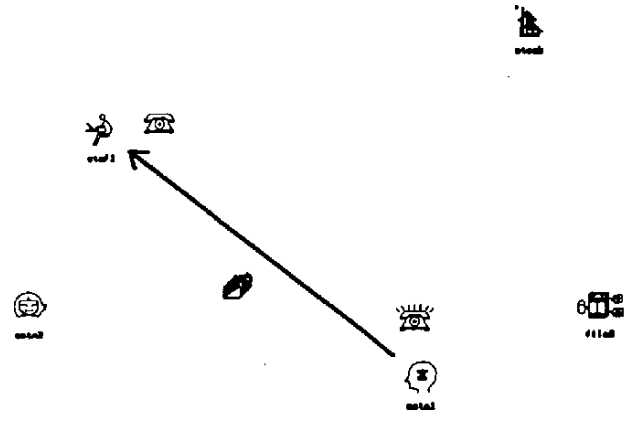


図3.2-11　アニメーションのスナップショット(3)

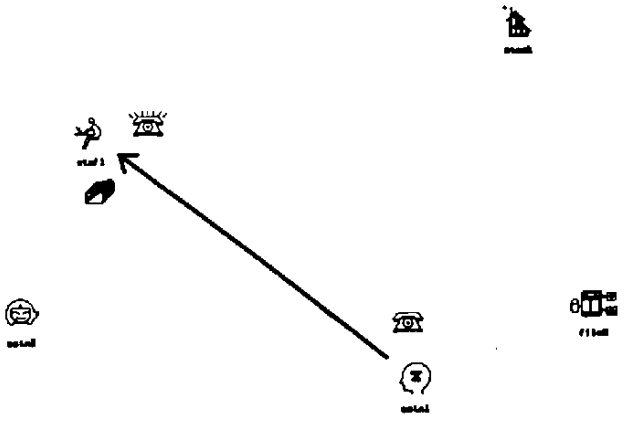


図3.2-12　アニメーションのスナップショット(4)

【参考文献】

[Chang89] Chang, S.K.: A Visual Language Compiler, IEEE Trans. Softw. Engnr. Vol.15, No.5, pp.506-525(1989).

[Davis90] Davis, A.M.: The Analysis and specification of systems and Software Requirements, IEEE Tutorial System and Software Engineering, (Thayer, R. H. and Dorfman, M. eds.), IEEE-CS Press, pp.119-144(1990).

[DeMarco78] DeMarco, T.: Structured Analysis and System Specification, p.352, Prentice-Hall (1978).

[Fisher88] Fisher, A.S.: CASE Using Software Development Tools, P.287, John Wiley (1988).

[Hsia88] Hsia, P. et al.: Screen-Based Scenario Generator: a tool for scenario-based prototyping, Proc. IEEE HICSS,Vol.2, pp.455-461(1988).

[情報処理学会80] 情報処理学会編: 情報処理ハンドブック, p.1166, オーム社,(1980).

[日本規格協会92] 日本規格協会編: JISハンドブック情報処理 ソフトウェア・符号編, p.1699, JIS X 0121(1986), (1992).

[Marca88] Marca D.A. et al.: Structured Analysis and Design Technique, p.393, McGraw-Hill Book(1988).

[大西90] 大西　淳他: 要求フレームに基づいた要求仕様化技法, 情報処理学会論文誌31巻2号, pp.175-181(1990).

[大西92] 大西　淳: 要求定義のためのコミュニケーションモデル, 情報処理学会論文誌33巻8号, pp.1064-1071(1992).

[Ohnishi93] Ohnishi, A. et al.: CARD: A Software Requirements Definition Environment, Proc. IEEE Requirements Engineering (ISRE), pp.90-93(1993).

[Ohnishi94] Ohnishi, A.: A Visual Software Requirements Definition Method, Proc. IEEE Requirements Engineering (ICRE'94), pp.194-201(1994).

[St-Denis90] St-Denis, R.: Specification by example using graphical animation and a production system, Proc. IEEE HICSS , Vol.2, pp.237-246(1990).

3.3 視覚的開発環境

3.3.1 はじめに

3.1節、3.2節では、ソフトウェアあるいは仕様そのものの視覚化について述べた。本節では、エディタやデバッガなどの開発ツールを中心に、視覚的環境を提供している例や協同作業環境を提供している例を示し、その動向について述べる。

本節では、まず第一に仕様の作成から基本設計までを扱う上流工程を支援する環境について述べる。次に、ソフトウェアのコーディング（製造）や検査に相当する中下流工程を支援する環境について述べ、最後に、版管理などの開発管理を支援する視覚的環境について述べる。

3.3.2 設計を支援する視覚的開発環境

(1) 概論

ソフトウェアの要求定義から基本設計に至る上流工程では、開発対象のソフトウェアについての諸定義、すなわち仕様を作成し、さらにその仕様についての共通の認識を、開発者と利用者の間及び開発者間で形成することがそもそもの目的である。この目的を達成するため、要求仕様書、基本設計仕様書など様々な文書を作成し、読み合わせを行い、記録に残している。

したがって、上流工程での視覚的支援環境では、以下のような支援アプローチが考えられる。

(a) 発想支援

ソフトウェアの仕様をより充実したものにするため、仕様定義の段階で自由で広範な発想が行えるような効果を狙う。具体的には、議論を活性化する効果を持つ会議支援ツール等が考えられる。

(b) 表現支援

作成した仕様をより分かりやすく表現する手法を提供し、利用者と開発者、あるいは開発者間での意図の伝達を円滑にしたり、これらの人々の間の意識のギャップを早期に発見するような効果を狙う。3.1節や3.2節で述べた仕様の視覚化はこの例である。

(c) 文書作成支援

文書の作成作業そのものの負荷を軽減するような効果を狙う。例えば、共同文書作成を支援するグループウェアや、発想支援の結果を文書として自動的に記録する機能などには、このような効果があると考えられる。

以下に示す個々の事例は、上記の支援要素を複合して持っていると考えられる。なお、3.1節、3.2

節で既に述べた例や、ソフトウェア開発以外にも広く応用可能な共同文書作成環境等については、本節ではその説明を省略する。

(2)　ワークフローシステム

ワークフロー（Workflow）システムでは、開発対象のシステムの構成やシステム利用者の役割分担を、業務の流れに合わせて記述する。ワークフローの記述はビジュアルなチャートで行われることが多く、仕様の表現支援の効果が期待できる。また、ビジュアルなチャートを自由に変更して様々な仕様例を検討できることから、発想支援的な効果も持ち合わせていると考えてもよいだろう。なお、多くの場合、ワークフローシステムでは開発者というより利用者側での利用（ユーザ側のSE等）を想定しており、エンドユーザによるシステム開発が可能となっている。

以下に例を挙げて説明する。

(a)　Action Workflow

Action Workflow [Medina-Mora92,Medina-Mora93]は、商品化されているワークフローシステムの代表例の1つである。Action Workflowでは、図3.3-1に示すような基本ループの組合せによって、ワークフローを表現する。すなわち、すべての協調作業をCustomer（作業依頼者）とPerformer（作業実行者）の間のProposal（提案）、Agreement（同意）、Performance（実行）、Satisfaction（満足）の4つのフェーズからなるループとしてモデル化している。

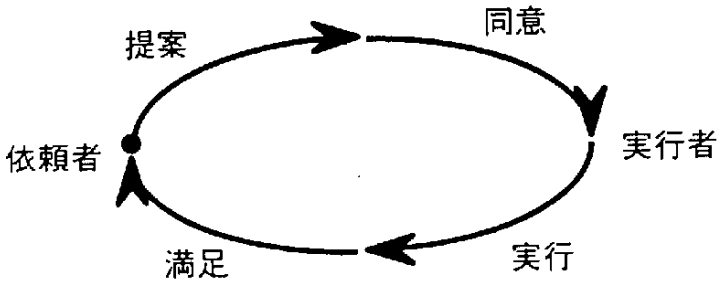


図3.3-1　Action Workflowの基本ループ

提案：作業依頼者が作業実行者に対して作業の内容を示す。

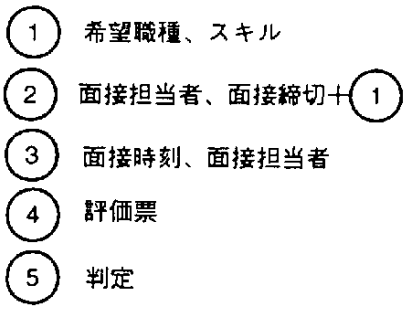
同意：作業実行者が、示された作業の内容を理解し承諾する。

実行：作業実行者が作業依頼者に対して作業の結果を示す。

満足：作業依頼者が作業の結果を見て評価する。

このループを複数組み合わせることによって、複雑なワークフローも表現できる。4つのフェーズのそれぞれを、また別のループで細分化して示すこともできる。具体的な業務をワークフローで示し

た例を図3.3-2に示す。これは、会社の人事部門による採用面接の手順を示したものである。図中の丸で囲んだ数字はフォーム等のデータを表現している。また、右下のループでは実行フェーズが複数の矢印によって表現されているが、これは面接担当者が複数いることを示している。



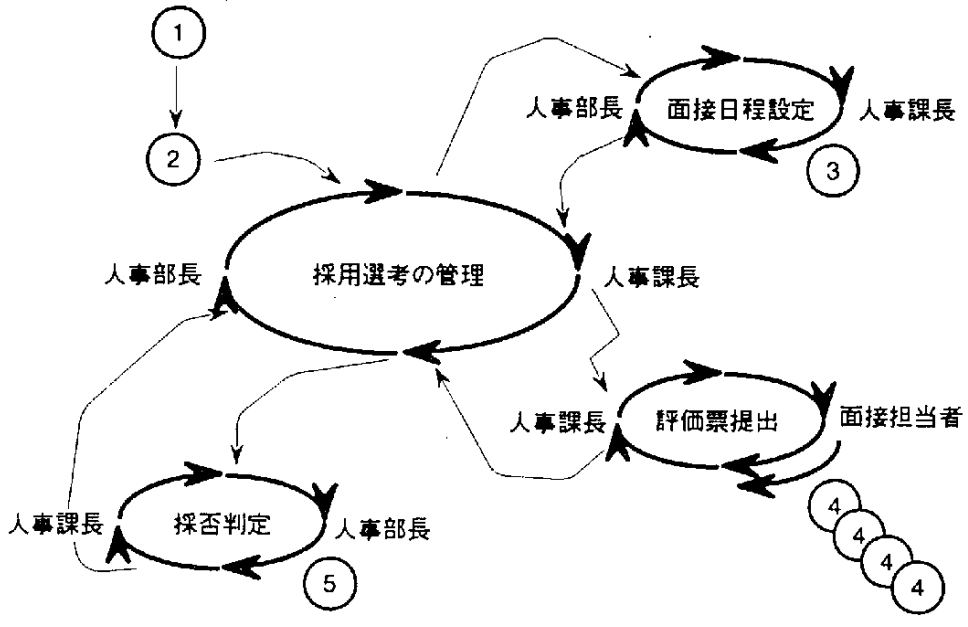


図3.3-2　Action Workflowによる採用面接のワークフロー（出典：[Medina-Mora93]）

(b)　Regatta

Regatta[Swenson93]では、プラン（plan）と呼ばれる単位でワークフローを部品化する。各プランは、ステージ（stage）と呼ばれるノードのネットワークになっている。各ステージは、さらにサブプランを用いて階層的に定義することが可能である。プランの例を図3.3-3に示す。これは、ソフトウェアのバグ報告に対する検査の手順を示したものである。

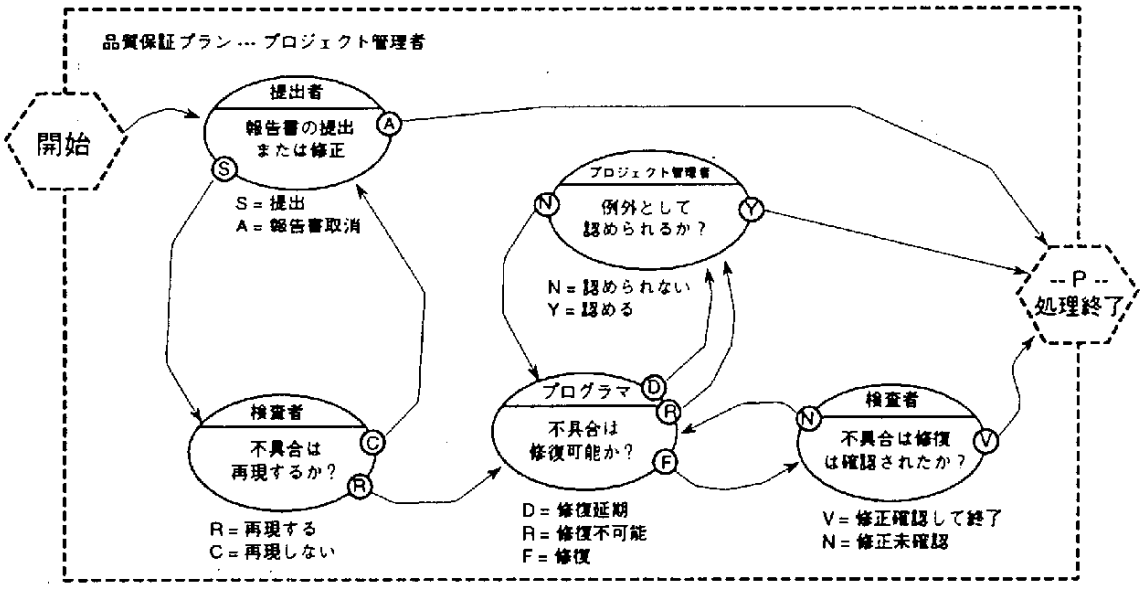


図3.3-3　Regattaにおけるチャートの例（出典：[Swenson93]）

プランは六角形の端子（1つの入口と複数の出口）を持つ破線の長方形によって表現されている。ここで示した例は、ソフトウェアの検査に関する手順を表現したものである。各楕円がステージであり、楕円の上側に作業担当者（個人名あるいは役割名）が、下側に作業内容あるいは判断基準が示されている。ステージは複数の小円で表された端子を持ち、それぞれが出口となる。

ステージはさらに別のプランによって詳細化可能である。例えば、図3.3-3の左下のステージを詳細化するには、次のようなサブプランの型紙を作り、内部を再び複数のステージによって図3.3-4のように構成すればよい。

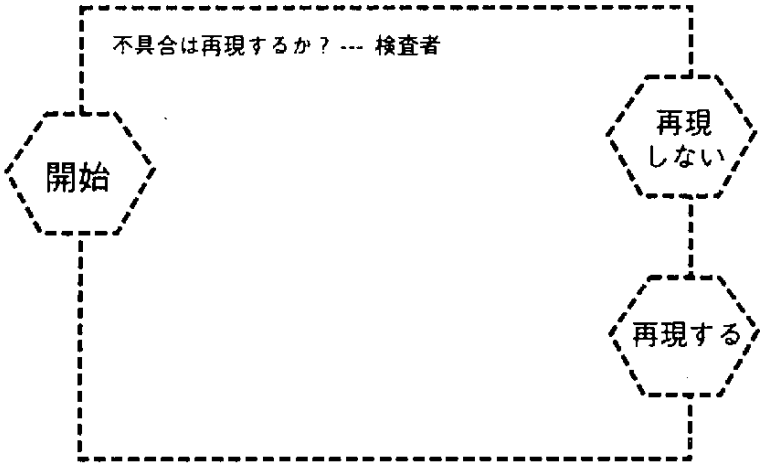


図3.3-4　サブプランの型紙の例（出典：[Swenson93]）

(c)　め組・育組

マルチメディア帳票インタフェースを持つメールシステム「め組」では、ルールによる自動ルーティングによってワークフローを実現する。そのルーティングと各業務担当者での処理等をビジュアルに定義するツールは育組と呼ばれている[Tarumi94]。

図3.3-5は育組によるワークフローチャートの例である。グラフの各ノードは、業務に関わる人（役割）を表現している。また、ノード間を結ぶ矢印は、書類の流れを示し、矢印の近くには書類の名称が書かれている。

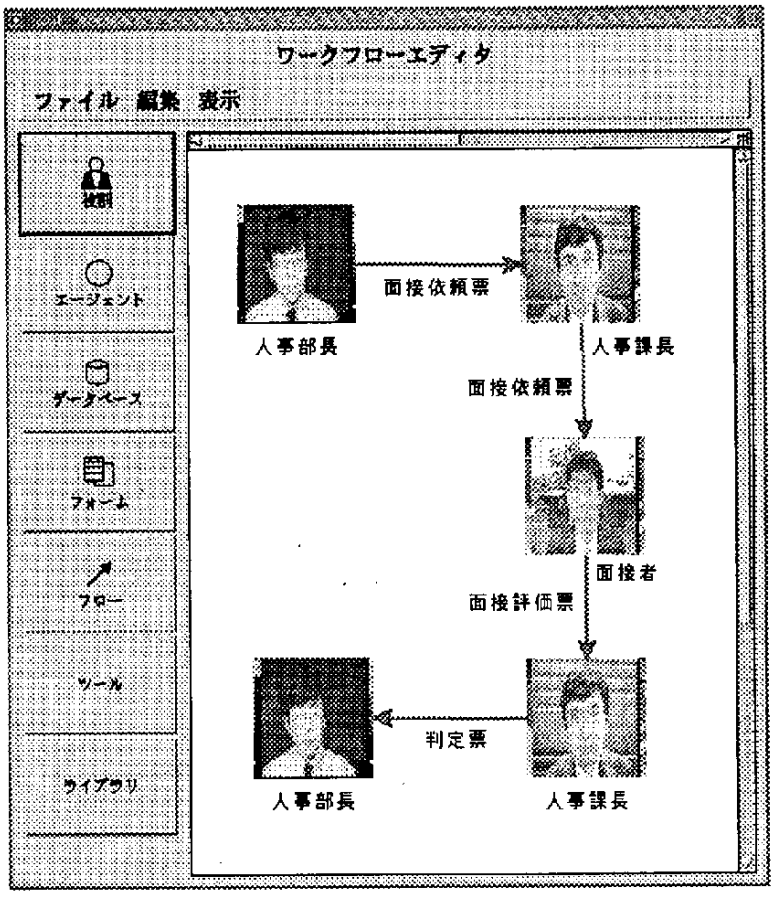


図3.3-5　育組によるワークフロー記述例

(3)　Design Rationale

Design Rationaleとは、設計結果を得るに到った履歴や理由のことである。Design Rationaleを記録することの最大の狙いは、設計の結果に到るまでの様々な議論をすべて残して、後々のソフトウェア保守に役立てることであるが、視覚的環境の立場で見ると、チャートの視覚的表現が発想支援や表現支援に役立っていると考えられる。

多少古典的であるが、最も有名な例としてgIBIS[Conklin88]がある。このシステムでは、問題点（Issue）、問題に対する回答案（Position）、及びそれに対する議論（Argument）という3種類のノードを、木構造状に展開したハイパーテキストを共有しながら構築していくことで、設計の議論の過程を記録する。実用の結果、問題点の整理、未解決問題の忘却防止、開発会議の効率化等に効果があったとされている[Yakemovic90]。

(4)　その他

COMICS[仲谷90]は、システムの構造を劇場の構造と対比させ（劇場モデル）、システムをソフトウェアモジュールやハードウェアらが演じる一連のドラマと見なし、そのドラマをワークステーション上で表現するツールである。開発チーム内でソフトウェア仕様のイメージを徹底させるための意図伝達システムであり、COMICSは仕様の表現支援の効果を狙ったものである。

システムの動き、アルゴリズム、開発進捗等を記述したシナリオに従って仕様書の図が点滅したり紙芝居的に動作し、仕様の説明が容易にできる。その画面の例を図3.3-6に示す。COMICSには、

・設計者の自由な視点からのシステム構造記述ができる。
・シナリオによるダイナミックな表現ができる。
・COMICS上で設計できる。
・履歴の再現ができる。

等の効果がある。

リアルタイム型のグループウェアを用いた上流工程の会議に関する研究もある。例えば海谷らは、議事録の品質の向上や蒸し返し議論の減少などの効果を実験によって確認している[海谷95]。

(5)　まとめ

上流工程の視覚的開発環境では、場合によっては従来よりも多くのドキュメントの作成が強いられる場合がある。例えば、Design Rationaleの記録システムでは、従来記述されていなかった設計理由の部分まで記録を残す必要がある。またCOMICSでは、シナリオの記述という作業を余分に行わなければならない。しかしながら、実際の現場では納期に追われるあまり、仕様そのものの記録さえろくに残されず、口頭で伝えられたり、細部は開発者の頭の中にあったり、曖昧なまま下流工程に進んだりしているのが実情である。このような状況では、いくら良いツールでも余分な工数を割いてまで利用することは困難である。今後は文書作成支援機能や、下流工程と連携して後からの修正が円滑に行えるような（言い換えれば、スパイラルモデルのような開発プロセスにも対応できるような）機能の充実が期待される。

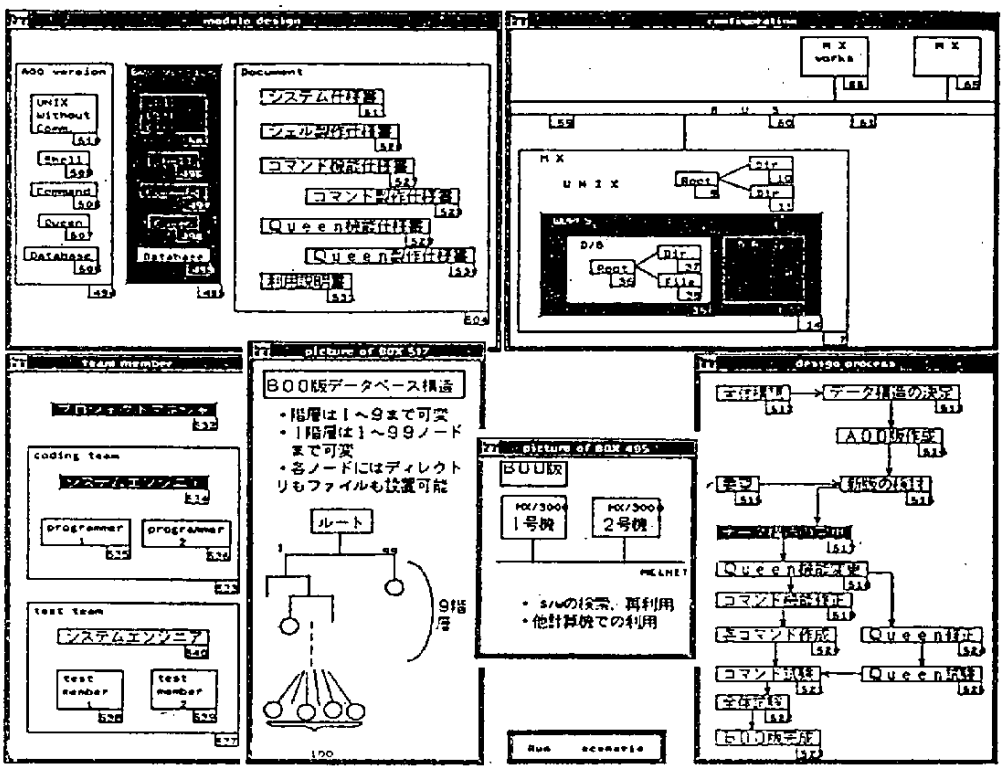


図3.3-6　COMICSの画面例（出典：[仲谷90]）

3.3.3　製造、検査を支援する視覚的開発環境

(1)　概論

本項で対象とするのは、プログラム作成、テストラン、デバッグ、性能測定などの製造、検査に係わる開発作業を、古典的なタイプライタ環境ではなくグラフィクスを用いた環境で行えるようにした支援環境であるが、グラフィクスを用いたエディタやデバッガについては80年代から存在している。そこで、ここでは特に最近の傾向として、複数の作業者による協同作業の支援を中心に述べることにする。なお、3次元グラフィクス等の、より高度のグラフィクスを用いた事例もまた最近の傾向であるが、これについては4章で述べるのでここでは省略する。

(2)　共同エディタ

コンピュータネットワークを介して、複数の作業者が一つの文書を編集することを可能にするようなエディタについて、最近研究されている。例えば、ミシガン大学で開発されたDistEdit[Knister90]は、エディタを改造して文書を共有しながら編集できるようにしたツールキットであり、共同編集エディタの開発を容易にする。例えば、ソフトウェア開発者が用いているemacsやvi等のエディタをマ

ルチユーザ版に改造することが可能である。

MEdit[Dewan93]は、Flecseというシステムの一部として開発されたマルチユーザエディタである。DistEditでは編集結果が即座に他のユーザにも反映されるのに対し、MEditでは編集結果の波及はすぐには行われない。その代わり、編集しているプログラムの関数単位や行単位の変更管理を行っており、ロッキング機能や複数のユーザによる変更を事後にマージする機能等が提供されている。ソフトウェアの分散開発のように必ずしも十分な通信容量が確保できず、また分担が比較的はっきりしている場合には有効であろう。

以上はテキストエディタの例であるが、グラフィックエディタの例としてはRendezvous[Brinck93]がある。RendezvousはDistEditと同様に、共同エディタを構成するためのツールキットである。これは、ユーザインタフェースの共同編集に応用されている（注：RendezvousはBellcoreの商標）。

(3)　視覚的デバッガ

最近の視覚的デバッガの例として、ParaGraph[Heath91]について述べる。このシステムは、並列マシンの並列プログラムの動作を可視化するものである。実行ログを元にして、25種類もの様々な表示を行うことができる。例えば、各プロセッサの稼働率、ロードバランス、プロセッサ間のメッセージのキューイング状況、通信量、クリティカルパス等をグラフを用いて表示する。その表示の一例を図3.3-7に示す。

ParaGraphは表示の種類が多いうえ、新たな表示方法の追加も容易であるのが特徴である。通常の開発環境（X Window）で利用できるので、4章で述べるような3次元グラフィクスシステムと比較すると実用的であると考えられるが、画面の広さの制約から同時に表示できるプロセッサの数が増やせない等の問題点があるとされている。

デバッグのための協同作業環境を提供した例としては、上記Flecseに含まれるMDebug[Dewan93]というツールがある。これはシングルユーザのデバッガに遠隔マルチユーザのインタフェースを与えたもので、例えば、あるユーザがステップ動作を実行する様子を、遠隔の別のユーザが監視することができる。デバッグ作業においては、専門的なノウハウを持つ開発者が初級開発者のデバッグを手伝うと効果的である場合が多く、このようなツールは役に立つだろう。ただ、MDebugはテキストベースであり、視覚性という点ではまだ発展の余地がある。

(4)　レビュー支援

プログラムコードのレビューは、複数の開発者で行うべき作業であり、グループウェア的な視点からの支援が期待されている。例えばICICLE[Brothers90]は、ペーパーレスのレビュー作業を行うためのコメント共有機能及び編集機能、さらには変数等の検索機能などを有している。

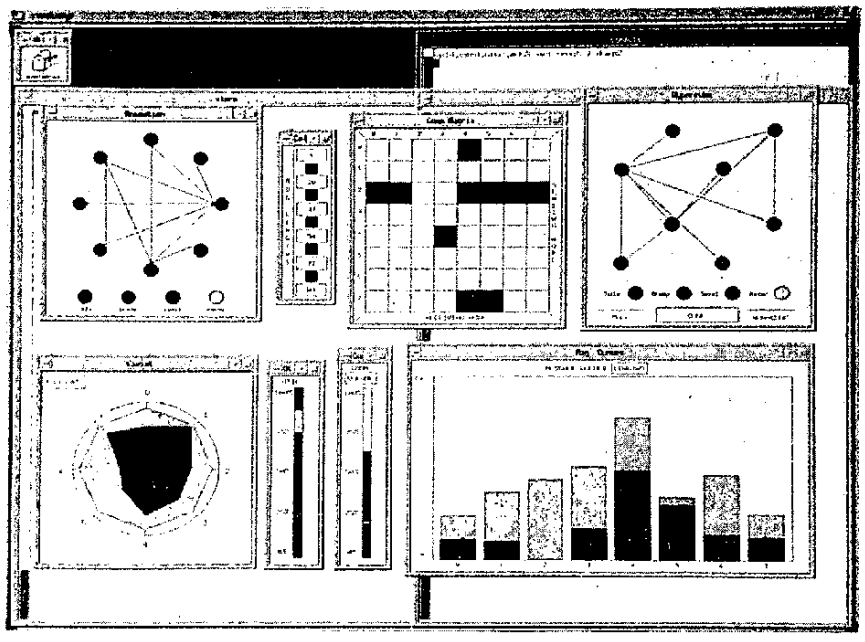


図3.3-7　ParaGraphの表示の一例（出典：[Heath91]）

さらに進化した形は、ハワイ大学のCSRS[Johnson93]に見ることができる。CSRSはマルチユーザ分散ハイパーテキストの環境で動作するシステムで、ユーザはプログラムに対するコメントを分散共有ハイパーテキストとして構築することができる。また、以下の機能も提供されている。

・集めたコメントから統計的データを生成する機能

・LaTeXによるレポート生成機能

・レビュー後の宿題のスケジューリング機能

・版管理システムSCCSとの連携

(5)　まとめ

製造、検査の工程では、上流工程とは異なり、共有している情報（プログラムや実行結果）は極めて具体的であり、開発者間でその認識について誤解が生じることは少ないが、その反面、扱う情報は非常に細かく大量なものになる。そのため、ParaGraphのように多数の表示方式を提供したり、CSRSに見られるように統計的データの生成機能を提供したりすることで、大局的な情報を開発者に提供する機能が提供されている。現在のワークステーションのグラフィクス機能はまだ不十分であり、十分な表示が行えるとはいえないが、最近、高度のグラフィクスが利用できるワークステーションが安価になってきており、今後の進化が期待できる。

一方、共同エディタの分野では、高速通信回線の普及により、遠隔リアルタイムの共同編集環境が、現実的なものに近づきつつある。環境が十分整えば、Rendezvousのようなグラフィックエディタの分野でも共同編集が現場での実用になるだろう。

3.3.4 開発管理を支援する視覚的開発環境

(1) 概論

本項では、開発管理者に対して提供される視覚的開発環境について述べる。

ソフトウェア開発の作業手順、すなわち開発プロセスについて、細粒度の管理をすることが重要であるとされている。管理者は開発計画に基づいて開発プロセスを設計し、それを末端の開発担当者まで十分に徹底させる必要がある。開発プロセスは頻繁に変更されるものである。そのため、開発プロセスを設計する管理者にとって、それが言語で記述されていると全体を把握しにくく、変更が困難になる。よって視覚的な開発プロセス記述・管理環境を提供することが重要である。以下に、事例を紹介する。

(2) ワークフローシステム

3.3.2で述べたワークフローシステムは、開発対象システムの記述だけではなく、開発プロセスの記述にも応用できる。実際、図3.3-3で示したチャートは保守工程の開発プロセスを記述したものである。ワークフローシステムには、通常、プロセスの進捗表示機能が備えられているので、開発管理システムとしては便利である。ワークフローシステムは、事務処理分野での普及が先行しているが、今後、ソフトウェア開発分野でも普及が期待されている。

(3) はこにわ

ワークフローシステムは、汎用的なプロセス記述システムであるといえるが、ソフトウェア開発に特化したプロセス記述システムも数多く研究されている。「はこにわ」[飯田93,飯田94]はその一例である。図3.3-8に「はこにわ」によるプロセスの表示例を示す。「はこにわ」は、複数の開発者の作業を自動化・誘導し、進捗データを自動的に収集する。

(4) まとめ

開発プロセスの視覚的記述についての事例を述べたが、実用という点で見るとまだこれからである。主な理由は開発部隊で統一した環境を整えるのが難しいことであろう。開発プロセスを徹底するためには、一人一台のマシン環境は必須である。一人一台のワークステーションを合言葉にして、多くの現場で開発環境の整備がこれまで進められてきた。最近ではダウンサイジングが進み、一人一台のパ

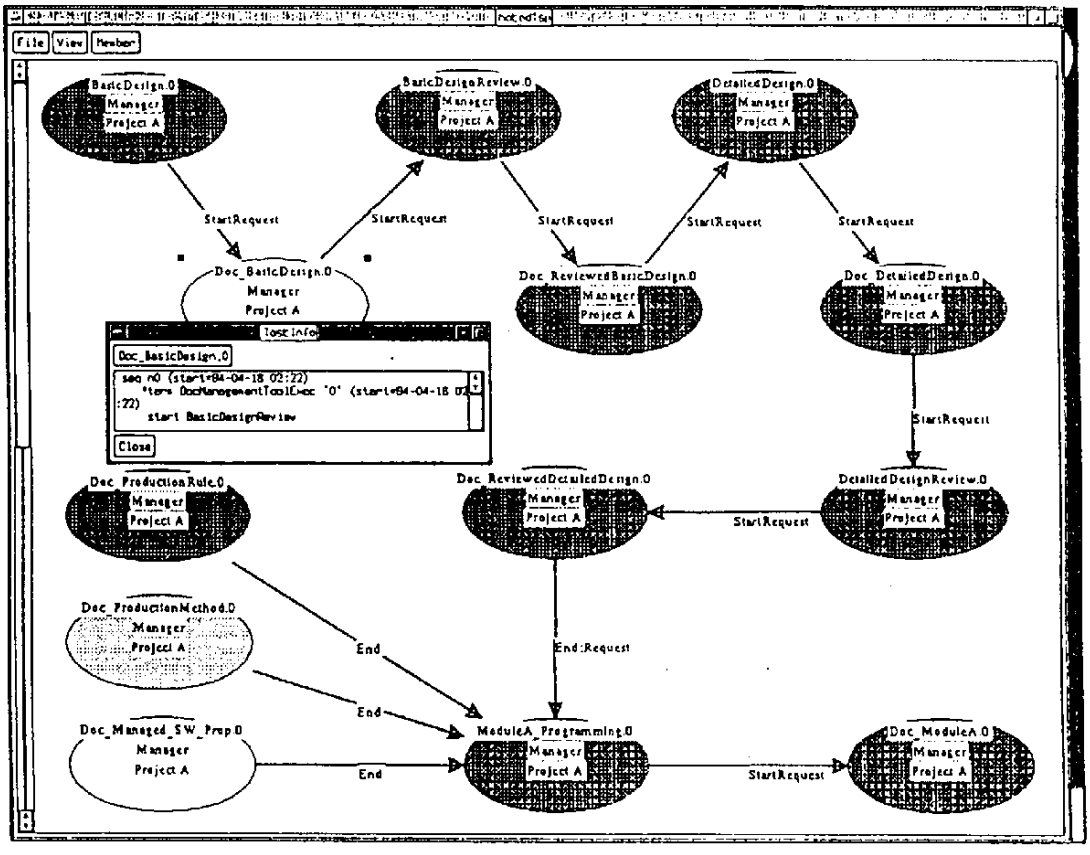


図3.3-8　はこにわの表示例（出典：[飯田94]）

ーソナルコンピュータという環境も求められている。

環境の整備に伴い、今後はプロセス記述だけではなく、開発進捗や様々な実績データを管理者に分かりやすく表示する機能も含めて、環境の進化が期待されている。

【参考文献】

[Brinck93] Brinck, T. and Hill, R.D.: Building Shared Graphical Editors Using the Abstraction-Link-View Architecture, Proceedings of the Third European Conference on Computer-Supported Cooperative Work, Kluwer Academic, pp.311-324 (1993).

[Brothers90] Brothers, L., et al.: ICICLE: Groupware For Code Inspection, CSCW'90 Proceedings, ACM (1990), pp.169-181.

[Conklin88] Conklin, J., and Begeman, M.L.: gIBIS: A Hypertext Tool for Exploratory Policy Discussion, CSCW'88 Proceedings, ACM, pp.140-152 (1988).

[Dewan93] Dewan, P and Riedl, J: Toward Computer-Supported Concurrent Software Engineering,

IEEE Computer, Vol.26, No.1 pp.17-27 (1993).

[Heath91] Heath, M.T. and Etheridge, J.A.: Visualizing the Performance of Parallel Programs, IEEE Software, Vol. 8, No.5, pp.29-39 (1991).

[飯田93] 飯田 元, 他: ソフトウェア協調開発プロセスのモデル化とそれに基づく開発支援システムの試作, 情報処理学会論文誌, Vol.34, No.11 pp.2213-2222 (1993).

[飯田94] 飯田 元, 他: プロセス記述・実行システムを用いた CMM と ISO9000-3の比較の試み, 情報処理学会ソフトウェアプロセスシンポジウム論文集, pp. 123-132 (1994).

[Johnson93] Johnson, P.M. and Tjahjono, D.: Improving Software Quality through Computer Supported Collaborative Review, Proceedings of the Third European Conference on Computer-Supported Cooperative Work, Kluwer Academic, pp.61-76 (1993).

[海谷95] 海谷治彦, 他: ソフトウェアの要求獲得を支援する対面式会議システムに関する一考察, 情報処理学会グループウェア研究会, 95-GW-9, pp.75-80 (1995).

[Knister90] Knister, M. and Prakash, A.: DistEdit: A Distributed Toolkit for Supporting Multiple Group Editors, CSCW'90 Proceedings, ACM, pp.343-355 (1990).

[Medina-Mora92] Medina-Mora, R., Winograd, T., et al.: The Action Workflow Approach to Workflow Management Technology, Proceedings of CSCW'92, ACM, pp.309-327 (1992).

[Medina-Mora93] Medina-Mora, R., Winograd, T., et al.: The Action Workflow Approach to Workflow Management Technology, The Information Society, Vol.9, pp.391-404 (1993).

[仲谷90] 仲谷美江, 他: 劇場モデルに基づいたソフトウェア意図伝達支援ツール COMICS, 情報処理学会論文誌, Vol.31, No.1, pp.124-135 (1990).

[Swenson93] Swenson,K.D.: Visual Support for Reengineering Work Processes, Proc.of Conference on Organizational Computing Systems, ACM, pp.130-141 (1993).

[Tarumi94] Tarumi,H. et al.: IKUMI: A Groupware Development Support System with Visual Environment, in Advanced Information Systems Engineering (Proceedings of CAiSE*94), Wijers, G. et al. (Eds), Lecture Notes in Computer Science Vol. 811, Springer-Verlag, pp.66-79 (1994).

[Yakemovic90] Yakemovic, K.C.B. and Conklin, E.J. : Report on a Development Project Use of an Issue-Based Information System, CSCW'90 Proceedings, ACM, pp.105-118 (1990).

4.　情報の視覚化とアクセス

4. 情報の視覚化とアクセス

4.1 3次元情報視覚化

近年のハードウェア技術の進歩に伴い、低価格高性能な3次元グラフィクスワークステーションが一般に普及し始めている。従来はCAD（Computer Aided Design）やScientific Visualizationといった特殊な分野でしか使用されることのなかった3次元コンピュータグラフィクス（CG）が比較的手軽に利用できるようになってきた。

この3次元CGの新たな応用分野に情報視覚化（Information Visualization）がある。情報視覚化は、情報を単なる文字の羅列でなく図として表示することにより、人間の情報に対する理解をより早く、より深くすることを目的としている。例えば、情報システムにおいて頻繁に利用される階層構造それ自体は単なる論理関係に過ぎず決して木の形をしているわけではない。しかし、これを木として表示することにより情報の全体構造が明白になる。もちろん、こうした図は以前から紙に描画されてきたが、ビットマップディスプレイを利用した2次元情報視覚化によって、図の直接操作（Direct Manipulation）[shneiderman83]による情報の検索や変更、すなわち図を通じての情報とのインタラクションが可能となった。そして現在、2次元グラフィクスに代わる新たな出力メディアとして3次元グラフィクスの可能性が研究されている。これを3次元情報視覚化、あるいは情報とのインタラクションを強調してインタラクティブ3次元情報視覚化と呼ぶ。

本節はインタラクティブ3次元情報視覚化（以下、3次元視覚化と略）についての解説である。本節では、まずこれまでの研究事例について紹介し、それらが明らかにした3次元視覚化の利点と問題点をまとめる。次に、3次元視覚化を実用化する上で重要な研究課題について述べ、これまでに提案された幾つかの解決手法を紹介する。最後に、3次元視覚化を利用したアプリケーションを作成する際に考慮すべき重要な点についてまとめる。

4.1.1 研究事例

(1) SemNet

3次元情報視覚化の先駆的研究は、米国MCC（Microelectronics and Computer Technology Corporation）で開発されたSemNet[semnet]である。SemNetはPrologで記述された大規模知識ベースの効率的検索及び保守に、知識要素をノード、知識要素間の関係をリンクとするグラフ表現を用いた3次元視覚化を採用した。

SemNetは大量のノードの視覚化を可能にするとともに、2次元グラフにおいて一般に生じるリンクどうしの交差を回避した。しかしその一方で、SemNetは3次元視覚化が持つ次のような本質的問題に直面することとなった。

(a) 知識要素の配置

知識要素をランダムに配置すると、得られるグラフは複雑な巨大迷路のようになり、ユーザはこのグラフからなんら有益な情報を得ることはできない。単純な3次元化は巨大な知識ベースを巨大な3次元グラフにマップするだけなのである。

これに対しSemNetは以下の配置手法を用いて、大規模知識ベースになんらかの「形」を与えようとした。

・マッピング関数

知識要素をその属性に基づきxyz軸に割り当てる。

・多次元尺度法

多次元尺度法のKruskalのアルゴリズム[kruskal64a,kruskal64b]を用いて知識要素を配置する。

・発見的手法

中心化手法、焼き鈍し法という2つの発見的手法を用いて、リンク接続された要素どうしが近くに配置されるようにする。

・ユーザによる配置

マウスを用いて特定の知識要素を希望する位置に動かす。

(b) 表示図素数の削減

3次元の採用によって2次元に比べはるかに多くの知識要素を表示できるようになった。しかし、表示図素数の増大は2つの問題を引き起こした。第1はグラフィクスの表示速度の低下である。3次元グラフィクスでは、画面の更新速度は描画されているポリゴン数にほぼ比例して増大する。したがって、静止した図を見るのには問題ないが、3次元オブジェクトや視点のスムーズな移動が要求される場合にはこれがネックとなる。第2は人間の認知負荷が高まることである。人間は図から一度に多くの情報を獲得できるが、表示される情報量が多すぎると図は逆に人間の認知を妨げる。

これに対しSemNetは次の手法により表示情報量の削減を行った。

・空間クラスタリングに基づくFisheye View

空間クラスタリングとはCADで利用されるオクトツリーの考え方を利用したもので、SemNetの表示領域である3次元立方空間を8個の部分立方体に分割し、得られた部分立方体をさらに8分割する。次に各知識要素をそれらが含まれる部分立方体の子ノードとした木を作成する。そして、現在ユーザが存在する部分空間を着目点として後述するFisheye View[furnas86]を適用し、ある閾値以下のクラスタに含まれるノードは消去し、単にクラスタの存在を示すノードだけを指標として表示した。この手法はユーザの着目点があるクラスタは詳細に、それ以外は指標だけを表示できるが、着目点がクラスタ間を移動する際に表示の不連続な変化が起こり、ユーザの認知を著

しく妨げるという問題点がある。

・リンクの表示

巨大知識ベースを視覚化した場合、膨大なノード数もさることながら、表示すべきリンク数の多いことが問題となる。これらは後方に存在するノードの表示を妨害するとともに、ノード同様画面更新速度を低下させるからである。これに対しSemNetでは、リンクの始点ノードが表示されている場合にだけリンクを表示するという戦略を用いて表示するリンク数の削減を行った。

(c) 位置の制御

第3の問題点は3次元空間における視点の移動（Navigation）である。SemNetは複雑な3次元空間での視点移動に次の手法を用意した。

・相対移動

ユーザの視線に沿った前後移動、及びロール、ピッチ、ヨーの回転を行う。しかし、この手法ではユーザは3次元空間において容易に迷子になり、かつ移動速度が遅い。

・絶対移動

xy平面とyz平面への2つの2次元投影図を用意し、各平面図上において自分の位置を示すマーカを動かすことで3次元空間における絶対移動を行う。この手法の問題は、正確な位置への移動が困難なことと、3次元空間を2つの2次元図に投影することによって生じるユーザの認知負荷の増大である。

・テレポーテーション

最近たどった知識要素の履歴を保持し、これらを選択することで以前いた位置に瞬時に戻ることができる。

・超空間移動

現在選択されている知識要素にリンク接続されている要素を一時的にその近くに配置し表示する。この機能が解除されるか別のノードが選択されると知識要素は元の位置に戻り、別のノードが選択された場合、視点もそのノードに従って移動する。

これらの3次元空間視点移動により明らかとなった重要な問題点の1つは、3次元空間における迷子問題である。3次元空間を自由に動き回らせると、ユーザは容易に迷子となる。Poltrockはこの迷子問題がSemNetにおける最大の問題であったと述べている[poltrock]。また、この問題に関しては[darken]でも詳細に述べられている。

結果的にSemNetが明らかにした3次元の利点は、大規模データの表示とリンク交差の回避であるが、SemNetの真の意義は大規模知識ベースの視覚化を通じて、①3次元空間での配置、②表示図素数の増加、③3次元空間とのインタラクション、という3次元視覚化が持つ本質的な問題点を明らかにした点

である。

(2) Information Visualizer

Information Visualizer[card91, robertson91, mackinlay91]はXerox PARCで行われている情報視覚化プロジェクトである。Information Visualizerはオフィスワークにおける情報を対象としている。特に階層データと線形データという代表的な情報構造のための視覚化システムが作成され、前者をCone Tree、後者をPerspective Wallという。

(a) Cone Tree

Cone Tree [robertson91]はUNIXディレクトリに代表される階層データを3次元の木として表示するシステムである。木の各レベルにおいて子ノードは親ノードを頂点とする円錐の底面円周上に配置される（図4.1-1)。2次元表示では大規模な木は容易に表示領域から溢れてしまうが、Cone Treeはより大きな階層データを画面溢れを起こすことなく表示することができる（ただし、[robertson91]の中で述べているようにアスペクト比がほぼ一定になるという表現には問題がある。これに関する議論は[koike93e]でなされている)。つまりCone Treeは、3次元の奥行き方向を利用することによる表示領域の効果的利用を示したわけである。

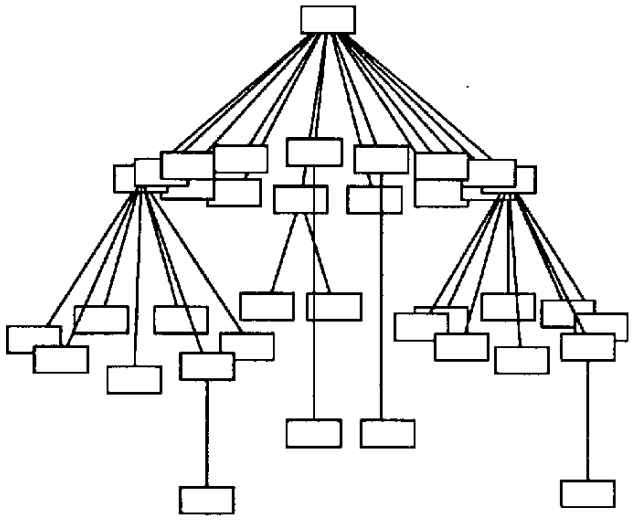


図4.1-1 Cone Treeによる階層データの3次元配置

ユーザが任意のノードをマウスで選択すると、ルートからそのノードへのパス上に存在する全ノードが最前面に一直線に並ぶように、各円錐は並行して回転する。しかも、後述するようにこの回転時間はユーザの認知を妨げないような速度で行われる。表示ノード数が増えた場合には、着目していな

い部分木の円錐だけを表示して個々のノードを隠したり、あるいはマウス操作によって不必要な部分木を隠す（pruning）こともできる。

(b) Perspective Wall

Perspective Wall[mackinlay91]は、プロジェクト管理データに代表される線形データを3次元の壁上に表示するシステムである。この3次元の壁は途中2カ所で折れ曲がり、中央部分は計算機画面と平行だが、左右部分はそれぞれ端が画面奥行き方向に遠ざかっている（図4.1-2）。この壁に線形データを表示すると、中央の壁に表示されるデータはその詳細を見ることができ、一方、左右の壁に表示されるデータはその存在だけを把握することができる。従来の2次元視覚化で大規模な線形データを見る場合、着目点近傍を詳細に見ようとすると全体構造が見えなくなり、逆に、全体を見ようとすると個々の詳細が見えなくなる。これに対しPerspective Wallは、3次元グラフィクスの透視投影図法（Perspective View）を利用することによって、局所的詳細と大局的概略を統合した表示を可能とした。

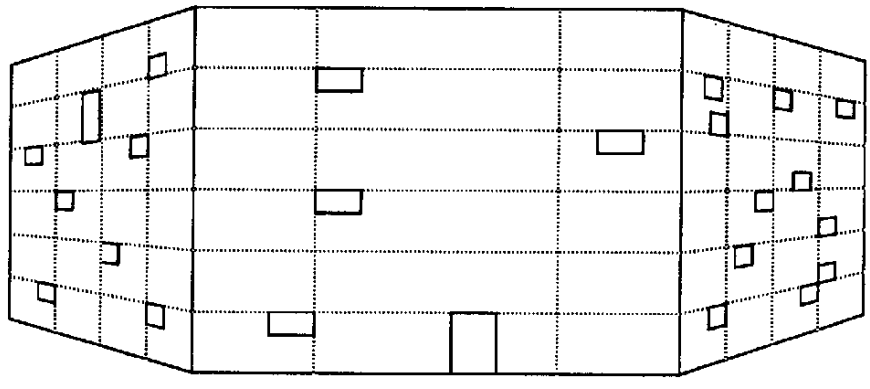


図4.1-2 Perspective Wallによる線形データの3次元配置

ユーザがあるデータを選択すると、そのデータが画面中心にくるように線形データ全体が移動し、常に着目データを中心とした視点が得られる。またCone Tree同様、この移動はユーザの認知を妨げない速度で行われる。

(c) Cognitive Coprocessor

Cone TreeとPerspective Wallが強調したもう1つの点はアニメーションによるユーザの認知負荷の低減である。ユーザが着目点を移動する場合、現在提示されている図から新しい図へ急に変えると、ユーザは両者の整合性をとるために認知的負荷を強いられる。逆に変化が遅すぎると、ユーザは不必

要に長時間待たされることになる。彼らはこれに対しCognitive Coprocessor[robertson89]というアーキテクチャを提案し、ユーザに認知負荷をかけない範囲の時間で表示の切り替えを行った。

Information Visualizerは階層データと線形データに対するそれぞれ特殊化したデザインのシステムを作成した。したがって、SemNetで生じた配置問題を気にする必要がなかった。さらに、3次元オブジェクトとユーザ視点の相対位置は基本的に変わらないため、3次元空間での視点移動問題をも気にする必要がなかった。

Information Visualizerプロジェクトは現在も引き続き進行中で、3次元を利用したドキュメント・ブラウジング[robertson93]、表のブラウジング[rao94]、カレンダ[card94, Robertson94]への応用が行われている。

(3) VOGUE

VOGUE[koike91]は東京大学と東京電力の仮想現実感プロジェクト[myoi91]において開発された3次元ソフトウェア視覚化システムである。VOGUEはソフトウェア開発で利用される様々な図のうち、互いに関連を持った2つの2次元図を1つの3次元図として統合する枠組を提案した（図4.1-3）。

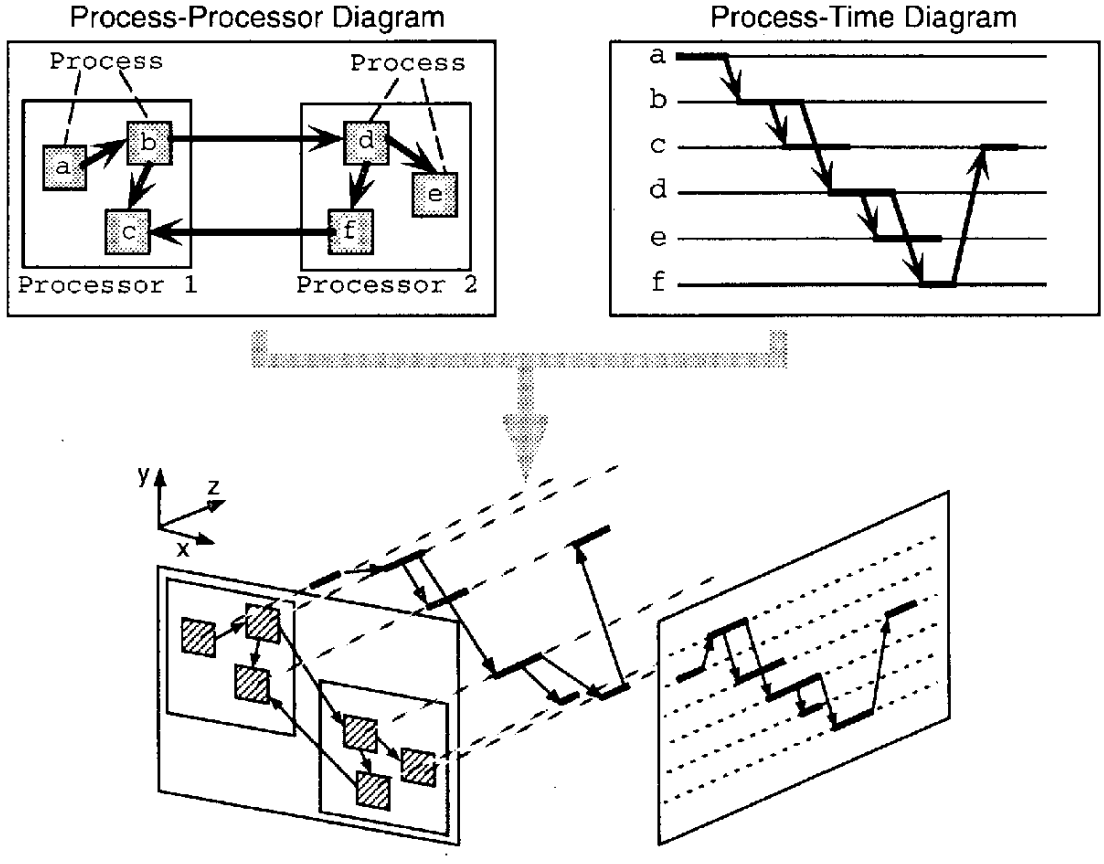


図4.1-3 VOGUEにおける3次元視覚化の枠組

プロセス間関係図とプロセス-時間関係図を1つの3次元図として記述することができる。この統合された3次元図の意義は次の2点である。

・モジュール構造を持った並列プロセスの時間変化のように、その表示に基本的に3次元が必要な情報の記述が可能である。

・2つの2次元図間の関連性を理解するためのユーザの認知的負荷が低減される。

VOGUEはこの枠組を次の3つのアプリケーションに適用した。

(a) 並列プロセスのモニタ

第1の応用例は並列プロセスのモニタである。階層的モジュール構造を持つプロセス間関係図をxy平面に記述し、z軸を時間軸にとった（図4.1-3)。従来のプロセス-時間関連図に比べプロセス関係記述のための自由度が1増加した結果、①大規模プロセスの表示、②プロセス-プロセッサ間関係の記述、③局所的詳細情報と大局的概略の統合表示、が可能となるとともに、④プロセス間通信の視認性が向上することを、電力制御用ソフトウェアのトレースデータの視覚化、自律分散型ロボットアーム[ikei2]におけるメッセージパッシングの視覚化を用いて示した。

(b) オブジェクト指向言語のクラスライブラリ

第2の応用例はオブジェクト指向言語のクラスライブラリの視覚化である。クラス階層をxy平面に木として表示し、各メソッドはそれが属するクラスと同一のxy座標を持ち、かつ同一名を持つメソッドは同一z座標を持つように配置した。ユーザは1つの3次元視覚表現でクラス階層とメソッド継承が同時に把握できるため、あるインスタンスにメッセージを送った場合、どのメソッドが実際に起動されるかを視覚的に迅速に同定できる。被験者実験の結果、Smalltalk[smalltalk]のシステムブラウザ、Prograph[prograph]の図形的2次元ブラウザに比べ、より短い時間でメソッド探索課題を達成することが示された[koike92b]。

(c) バージョン管理・モジュール管理

第3の応用例はバージョン情報・モジュール情報の視覚化である。VOGUEでは、各ファイルのバージョン履歴をz軸を時間軸にとった木として表現し、xy平面から見たとき同一モジュールのファイルが近くになるように配置した。そして、1つのメジャーリリースを構成するバージョンどうしをリンクで接続することによって、特定のメジャーリリースを構成する各ファイルのバージョンが視覚的に把握できる。これに対し、一般のCASE（Computer Aided Software Engineering）システムではバージョン履歴とモジュール構造がそれぞれ別の2次元図として視覚化されるため、複数ファイルの異なるバージョン間の関係を管理できない。

VOGUEでのインタラクションは基本的にマウスとキーボードで行われ、視点の移動にはGUIツールのスライダを用いる手法とテンキーを用いる手法がある。また、VOGUEは仮想現実感インタフェースをも試験的に使用し[myoi91]、3次元視覚化システムの仮想現実感インタフェースとの整合性の高さを示した。図形数削減手法としては、①オブジェクト指向データベースが保持する要素のクラス情報に基づく方法、②後述するFractal View[koike92a]を用いた手法が特徴的である。

VOGUEの意義は、SemNet、Information Visualizerにおいて曖昧であった3番目の座標軸の積極的利用法を提案した点である[koike93b]。例えばCone Treeでは、1つの視覚表現中に含まれる関係は階層構造だけであり、したがって視点を変更しても得られる情報は基本的に同じである。これに対しVOGUEでは、2つの異なる関係が1つの視覚表現の中に（互いに他を邪魔することなく）埋め込まれており、ユーザはより少ない認知的負荷でより多くの情報を得ることができる。現在VOGUEプロジェクトは電気通信大学において引き続き行われており、並列プログラミング言語Lindaの視覚化[takada93]、オブジェクト指向言語C++の視覚化[kobayashi93]、UNIXのバージョン管理ツールSCCSの視覚化[chu93]が行われている。また、前述したInformation Visualizerでも、最近ではVOGUEと同様の発想の3次元統合を試みている[Mackinlay94]。

(4) その他のシステム

以上の先駆的な研究を基に、最近では多くの3次元視覚化システムが開発されている。ここでは、それらについて簡単に紹介する。

(a) FSN

FSN[fsn]はSilicon Graphics社で開発されたUNIXファイルシステムの3次元ナビゲーションシステムである。FSNはUNIXディレクトリを水平面上に2次元の木として描画し、ユーザは飛行機から地面を見るようにUNIXディレクトリを見る（図4.1-4）。Perspective Wall同様、透視投影図法を利用しているため、着目しているディレクトリに関する情報は詳細に見ることができ、より深いディレクトリはその概略だけを把握することができる。また、高さ方向をファイルサイズ等の付加的情報の表示に利用している。なお、FSNはftp.sgi.comからanonymous ftpが可能である。

(b) n-Vision

Feinerらのn-Vision[feiner90]は、3次元空間内に多次元データ(X1, X2, X3)の視覚化を行うシステムで、3次元直交座標系の中にもう1つの小さな3次元直交座標系が表示されている。外側の座標系が(X1, X2, X3)に対するグラフを表示し、内側のグラフは原点を(X1, X2, X3)とし、残りの(X1, X2, X3)に対するグラフを表す。そして、ユーザがDataGloveを用いて内側の座標系を動かすと(X1, X2, X3)が変

化し、それに伴い内側の座標系に表示されるグラフが変化する。

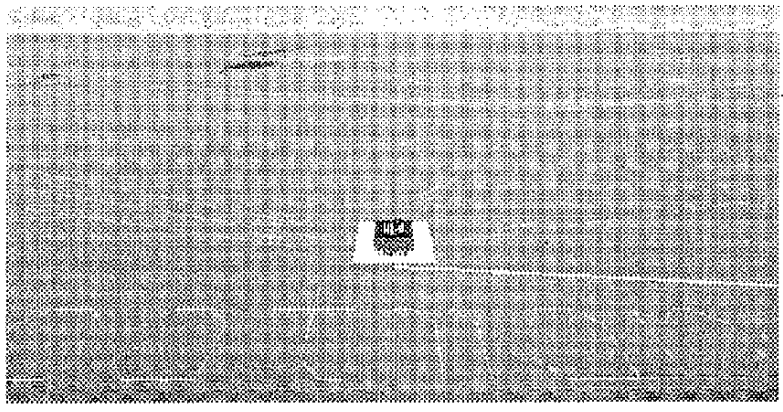


図4.1-4　FSN

(c)　Information Cube

Information Cube[rekimoto93]は、UNIXディレクトリに代表される階層構造を3次元の入れ子状の立方体として視覚化するシステムである（図4.1-5)。各立方体はディレクトリを表し、子ディレクトリの立方体は親ディレクトリの立方体に包含されている。そして、3次元グラフィクスの透明度（Transparency）を利用することにより、各立方体は内部が透けて見える。この透明度は内側の立方体ほど低くなるように制御されているため、ある程度の深さまで中の情報を見ることができるとともに、必要以上に深いディレクトリは隠され、図の複雑さを増さない効果が得られる。Cone Tree、FSNのように階層構造を木として表示した場合、画面からの溢れを避けることができないが、Information Cubeでは着目したディレクトリから下は必ず一定の範囲に収まる。

インタラクションにはアルバータ大学の仮想現実感研究で開発されたMR Toolkit [shaw93]を利用している。ユーザはDataGloveを装着し、グラフィクス内の仮想の手から発する光線で立方体を選択し、立方体に接近する。この接近においてユーザと着目立方体の距離は、着目立方体のスケールに比例するようになっているため、常に着目立方体を中心とした視点が得られる。

こうした入れ子の箱を用いた階層構造の視覚化は[lieberman89, koike91]にも見られるが、Information Cubeは箱とのインタラクションの点において完成度が高い。

(d)　Pad、Pad++

Pad[Perlin93]及びその後継版であるPad++[Bederson94]は、ズームによるナビゲーションを特徴とする情報システムである。Pad++は3次元視覚化システムではないが、通常の2次元インタフェースに、ズームによる連続的な拡大操作を加えた点が注目できる。これは3次元視覚化における遠近感の利用

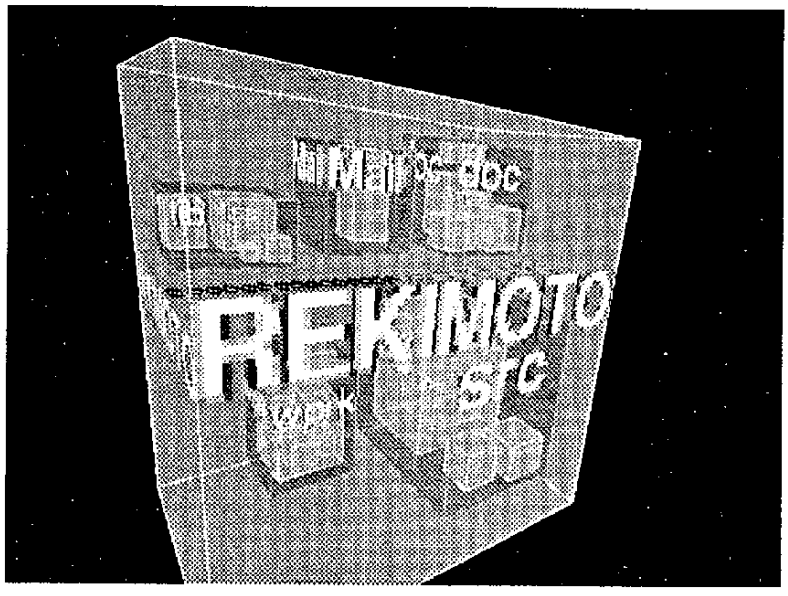


図4.1-5　InformationCube

と共通の利点を持つ。年間カレンダをズームしていくと各月の情報が表示され、さらにズームしていくと特定の日の情報が表示されるようになる、といった具合に、ズーム操作は情報空間にアクセスするためのメタファと考えることができる。情報アクセスの手段としてズームを取り入れた研究には、他にMITのLiebermanのmetascopeシステム[Lieberman94]、Galaxy of News [Rennison94]などがある。

(e)　PLUM

Brown大学のPLUM[reiss93]は、3次元視覚化のためのツールキットである。ノード、リンクといったプリミティブがオブジェクトとして定義され、3次元視覚表現を比較的簡単に作成することができる。ただし、既存の3次元システムの機能をツール化したにとどまり、新しい概念等の提案はない。

(f)　プログラム視覚化システム

プログラム視覚化システムとは、プログラムの静的、あるいは動的状態を図として表示することで、プログラムに対するプログラマの理解を促進するシステムである。このプログラム視覚化システムにおいても3次元の概念を取り入れたものが幾つか開発されている。

MITのLiebermanはLispのS式を階層的な3次元の箱として表示した[lieberman89]。S式の評価に従い、現在評価中のS式に対応する箱がアニメーションとともに拡大する。ただし、焦点はアニメーションにあり、3次元を利用する必然性はない。

Zeus[zeus]はDECのBrownらが開発中のアルゴリズムアニメーションシステムである。Zeusは以前

彼が開発したBALSA[brown88]の枠組を引き継いでいるが、最近、3次元の概念を導入しはじめた[brown93]。例えば、BALSAではアルゴリズムの多面的な振舞いをマルチウィンドウを使って表示していたが、Zeusではこれらの図を1つの3次元図として表示している。

同様に、アルゴリズムアニメーションシステムTango[tango]を開発したジョージア工科大学のStaskoらは3次元アルゴリズムアニメーションシステムPolka3D[stasko93]を開発し、アルゴリズムを3次元的なアニメーションで表示している。

CoxとRomanによるPavane System[pavane]は並行プログラミングのための視覚化システムで、3次元の概念を取り入れている。ただし、3次元の利用法は明確に述べられていない。

これらのプログラム視覚化システムは、基本的にプログラムの動作を見ることに焦点があるため、図とのインタラクションはほとんど考慮されていない。

4.1.2 3次元視覚化の研究課題

前項では、これまでの研究が明らかにした3次元の利点及び問題点を各研究ごとにまとめた。本項では、これらの問題点のうち特に重要と思われる3次元インタラクション技術と表示情報量の削減について、より詳しく解説するとともにこれまでに提案されている解決手法をまとめる。

(1) 3次元インタラクション技術

情報視覚化とScientific Visualizationとの決定的な相違点は、視覚化された情報とのインタラクションである。Scientific Visualizationの場合、どちらかというとデータの静的解析に重点を置いているが、情報視覚化の場合、単に見るだけでなく3次元オブジェクトの直接操作による情報へのアクセス・変更にも重点を置いている。

このインタラクション技術には以下の2つの操作が含まれる。

- オブジェクトの位置・姿勢等の操作（Manipulation）
- ユーザ視点の制御（Navigation）

MacDrawライクなインタフェースを持つ2次元インタラクティブシステムの場合、オブジェクトの位置に対してx、yの2自由度と、姿勢に関して表示平面に垂直な軸に対する回転の1自由度の計3自由度が操作できれば一般には十分である。また、ユーザ視点の変更に関しても縦横への画面スクロールとズームイン・アウトの計3自由度だけ考えれば十分である。そして、これらはすべてマウスによる操作として実現されている。

これに対し3次元の場合、単に1自由度の増加という以上に問題は複雑である。これは3次元グラフィクスにおける複雑な座標系に起因する[foley]。3次元グラフィクスには絶対座標系を持つグローバルな世界、個々の3次元オブジェクト、これらを映すカメラという3つの概念があり、オブジェクトと

カメラはそれぞれ相対座標系を持つ。したがって、オブジェクトの位置に関して絶対座標系でのx、y、zの3自由度、姿勢に関してオブジェクト相対座標でのロール、ピッチ、ヨーの3自由度の計6自由度を指定する必要がある。また、ユーザ視点に関しては、3次元カメラの絶対座標と相対座標における姿勢の6自由度の他に、カメラが現在参照している位置の絶対座標3自由度の計9自由度の指定が必要である（実際にはカメラの姿勢は法線ベクトルで表され、(1,0,0)等の値に固定することが多い。その場合は、カメラ位置と参照位置の計6自由度の指定が必要となる）。

こうした3次元におけるオブジェクトの位置・姿勢とユーザ視点の指定法としては、従来の2次元マウスを利用する手法と特殊な入力デバイスを利用する手法がある。

(a) 2次元マウスによるインタラクション

先に述べた視覚化システムの多くが2次元マウスを利用している。この理由は、これらのシステムが普通のワークステーション上での使用を前提とし、特殊な入出力デバイスの使用を避けたためである。ただし、2次元マウスでは自由度が不足しているため必然的に何らかの工夫が必要となる。

(i) 3次元オブジェクトの移動・回転

Chenらは3次元オブジェクトの回転に関し、①普通のGUIスライダを用いる手法（Sliders）、②回転オブジェクト上にスライダをオーバーラップさせる手法（Overlapping Sliders）、③カーソルがオブジェクト内にある時はXY方向、オブジェクト外にある時はZ方向の回転をする手法（Continuous XY+Z）、④回転するオブジェクトを仮想的なトラックボールと見立てる手法（Virtual Sphere）、という4種類の手法による被験者実験を行い、③と④が同等の成績を収めることを示した[chen88]。

同様に、Houdeは3次元オブジェクトの移動と回転に関する被験者実験を行い、操作の際にオブジェクトの回りにバウンディングボックスを表示することが有効であることを確かめた。また、ナレーティブハンドルと呼ばれる、手の形をした絵が物体の回転方向の正しい把握に有効であると述べている[houde]。

これに対し、HerndonらのInteractive Shadows[herndon92]は、3次元の物体に対して機械図面の正面図、側面図、立面図に相当する「影」を表示し、この影をマウスで操作することで3次元物体の操作を行うユニークな試みである。しかもこの影は単なる黒い影ではなく、ワイヤーフレーム、投影図といった様々な表示が可能である。

(ii) 視点の移動

最も単純な方法は、3次元カメラの6自由度に対してそれぞれスライダ等のGUI部品を対応させ、これらを操作することで視点を変更する手法である。ただし、パラメータが多すぎるため操作が非常に

複雑となり、ユーザの求める視点を得るのは難しい。

これに対し、SemNetの相対移動、FSNの視点移動等は、マウスの微小移動に応じて3次元カメラの6自由度の複数を同時に変更する手法である。

また、実際の視覚化システムでは、アプリケーションに応じた視点の移動方式が採用されることが多い。Information VisualizerのPoint of Interest方式[mackinlay90]は、ユーザが着目点を選択すると着目オブジェクトの法線ベクトル方向に視点が移動する。その移動速度は着目点との距離の対数関数になっており、距離が大きいときは速く、小さいときは遅いスピードで移動するように制御されている。またVOGUEやInformation Cubeでのオブジェクトへのフォーカスは、オブジェクトの縮尺率とオブジェクト・カメラ間の距離とが一定の比率になるように行われる。

この他に、Silicon Graphics社の3次元グラフィクスツールキットIRIS Inventorでは、ツール自体が視点変更とオブジェクトの位置・姿勢・属性変更のインタフェースを予め用意している。したがって、IRIS Inventorで開発されたアプリケーションはこの機能を利用できるようになっている。

(b) 特殊デバイスによるインタラクション

2次元マウスを利用する代わりに、特殊なデバイスを使用する方法である。一般にこれらはオブジェクト操作と視点操作の両方に使用できる。

CADではオブジェクトの回転等に従来からダイヤルボックスが利用されてきた。しかし、オブジェクトの選択等にはマウスを必要とし、結果としてユーザは各デバイス間を渡り歩かなければならず操作が煩雑となる。

SpaceBallはSilicon Graphics社のワークステーションで利用できる入力デバイスで、球形をしたハンドルの中にあるストレインゲージによって利用者の力の向きを検出できる。このため、3次元オブジェクトの並進移動及び回転が1つのデバイスで可能である。さらに、ハンドルにはボタンがありオブジェクトの選択もこれでできる。しかし、マウスのように迅速なオブジェクト選択や位置指定が困難である。

ローラーマウス[venolia]は従来のマウスのボタンの両側に2つのローラーを付けたものである。従来のマウスによって画面縦横方向、ローラーによって画面奥行き方向の指定を行う。

Polhemusセンサは磁気を利用して位置・姿勢の6自由度の検出が可能である。特にDataGlove[zimmerman87]とともに利用されることが多く、現在、仮想現実感研究で盛んに利用されている。ただし、正確な位置の設定が困難であるとともに、操作中は常に手を空中に置いておかなければならない。

Polhemusセンサ等の位置センサとボタンを組み合わせたデバイスは（マウスとのアナロジで）「バット」（こうもり）と総称される（図4.1-6)。DataGloveと違いジェスチャ認識などの用途には不向き

だが、装着の手間がいらないなどの利点も多い。例えば、3次元CADシステムなどの入力デバイスとして用いられている[Liang93, Shaw94]。

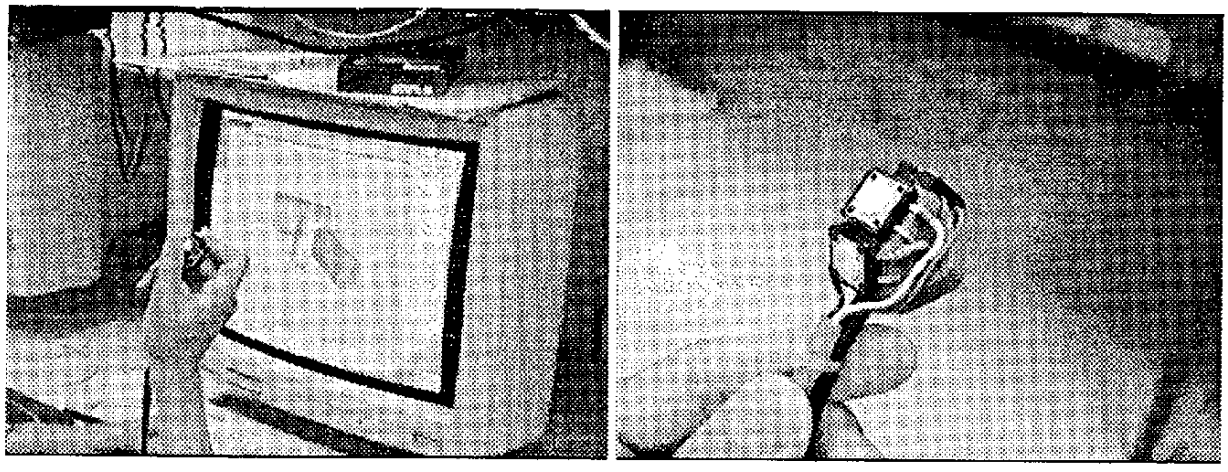


図4.1-6　Batの例（アルバータ大）

(2)　表示情報量の削減

(a)　Small Screen Problem

一般に、図は文字に比べより広い表示領域を必要とする。特に多量の情報を扱う情報視覚化においては、現在の高々20インチ程度のディスプレイにすべての情報を表示することは困難である。これはSmall Screen Problemと呼ばれ[henderson86]、情報視覚化における本質的な問題の1つである。

これに対する1つの解決策は、表示領域を物理的にあるいは仮想的に拡大することである。前者としては、MITで開発されたDataLand[bolt]が有名である。DataLandは大型ビデオプロジェクタを利用し、壁一面を表示領域として利用した。最近の仮想現実感研究でも大型ビデオプロジェクタが利用されている[amari93, takemura92]。これに対しRooms[henderson86]における「部屋」へのタスクの分類や、頭部搭載型ディスプレイ（HMD）を用いた仮想現実感研究[feiner91]、そして3次元視覚化における奥行き方向の利用は視覚化領域の仮想的拡大である。同じ大きさのディスプレイ（あるいはウィンドウ）を使っていても、我々ははるかに多くの情報を表示することができるようになった。

しかしながら、SemNetが明らかにしたように、表示領域の拡大による大量情報の表示は次の副作用を生じる。

・表示速度の低下

・ユーザの認知負荷の増大

前者はグラフィクスハードウェアの進歩とともに改善されるように思われる。しかし、グラフィクスが速くなれば、個々の情報の描画に複雑なポリゴンやTexture Mapping等の利用が要求され始め、

結果として速度不足は慢性化する恐れがある。さらに後者に至っては、いかに計算機技術が進歩しても決して解決されない問題である。

つまり、情報視覚化において重要なことは、いかに多くの情報を表示するかではなく、いかに必要な情報だけを表示するかなのである。

(b) 表示情報の削減手法

提示情報量の制御手法は、その特徴からセマンティックアプローチとシンタクティックアプローチの2つに分類することができる。

(i) セマンティックアプローチ

セマンティックアプローチとは、対象とする情報に関するデータベースを持ち、このデータベースを利用することによって必要な情報とそうでない情報とを区別する手法である。

① 知識工学的手法

個々の情報はもちろん、対象とする領域やユーザに関する知識を利用し、ユーザの意図を理解して必要な情報を提示する。しかし、ユーザの真の意図を把握することは非常に難しい上、あるアプリケーションにおける知識は他での応用がきかない。また、推論モジュールが大きくなりやすい。

② オブジェクト属性の利用

個々の情報の属性を利用し、ユーザの指示に応じて必要な情報を表示する。例えばVOGUEでは対象とする情報はオブジェクト指向データベースに登録され管理される。このクラス情報を利用することで、指定クラスに属する要素だけ、あるいは指定クラスとそのサブクラスに属する要素だけの表示・消去が可能である。

(ii) シンタクティックアプローチ

シンタクティックアプローチとは、対象とする情報を数学的グラフとして捉え、ノード間の距離、つまりノード間のリンク数に基づき表示・非表示を行う手法である。個々の情報が持つ意味を一切考慮しないので、必ずしもユーザの意図を反映しないが、実現法が比較的単純であるとともに、応用範囲が広い。

① 着目点からの距離

着目点からある一定の距離内の要素は表示し、それ以外は消去する。これは、着目点付近は見る可能性が高いという仮定に基づいている。古くはLispの省略表示がこの手法を利用している。

3次元視覚化では、遠近感（透視変形）により、同じ大きさのものでも、着目点に近い情報ほどスクリーン上で大きな面積を占める。これは情報量の制御と人間の持つ3次元空間の知覚能力を自然に

結合していると考えることができる。さらにCGの世界では、遠近感の演出にフォグ（遠くのものほど霞んでいく）などの技法も併用している。これらの技術を3次元視覚化に応用することも可能であろう。

CGからの技術の転用という観点では「半透明表示」による情報量の制御が最近注目されている。例えばウィンドウが半透明であれば、ウィンドウの後ろ側にある情報もそれなりに読みとることができる。現実世界と異なり、透明度を自由に設定することができるので、ユーザに提示すべき情報量の度合いを調節することもできる。情報視覚化に半透明の考え方を取り入れている研究としては、Information Cube[Rekimoto93]、Staplesのウィンドウシステム[Staples93]、Translucent Patches[Kramer94]、macroscope[Lieberman94]などがある。

② Fisheye Views

FurnasのFisheye Viewsは着目点からの距離と、対象とする情報構造の潜在的重要度（a-priori importance）を考慮することにより、着目点近傍を詳細に表示しつつ遠方の重要な点をも表示する手法である。この手法はSemNetで利用され、最近ではグラフレイアウト[sarkar92]や焦点からの距離に応じたアイコンの変更[fairchild93]等にも利用されている。

③ Fractal Views

前2者はあるノードでの分岐数の大小に関係なく同じ重要度を与えるため、ある一定閾値に対し着目点付近の分岐数が少なければより少ないノードが表示され、分岐数が多ければより多くのノードが表示される。これに対しFractal Views[koike92a]は、着目点からの距離をそのままノードの重要度とするのでなく、この計算にフラクタル関数を利用した。その結果、分岐数が少ない場合はより遠くのノードまで表示し、分岐数が多い場合はより近いノードだけ表示することができるとともに、表示ノード数が統計的にほぼ同程度となるという特徴を持つ。Fractal Viewsは大規模階層構造視覚化[koike93a]、Lispプリティプリンタ[koike94a]等に利用されている。

現在、こうした手法が実験的に利用されているが、決定的な解決手段はない。実際には、幾つかの手法を複合して利用することになる。

4.1.3 議論

前項において、3次元視覚化の研究課題を述べるとともに現在までに提案されている幾つかの手法に関してまとめた。本項では、3次元視覚化を適用するに当たって重要な点に関してより一般的な立場から述べてみたい。

3次元視覚化システムを作成するに当たり、まず第1に考慮しなければならないのは3次元の必要性である。2次元で十分なものをあえて3次元にする必要はない。Cone Treeによる巨大階層構造の視覚化は明らかに2次元視覚化の限界を越えている。またPerspective Wallの透視投影図法による局所詳細

と大局文脈の統合も3次元の概念によって可能となった。さらに、VOGUEの2次元図の統合は明らかに従来の2次元視覚化では不可能であった。こうした3次元の必要性に関する考察なしに作成されたシステムは、デモンストレーションとしてはいいかもしれないが、実用性は皆無に等しい。

第2に、効果的な3次元視覚化のためにはデータの量がある程度多くなければならない。例えば、ノード数10個程度の階層構造をCone Treeで3次元表示しても無意味だが、ノード数が数100個となると3次元表示の効果が現れる。

この量に関する議論は、3次元視覚化システムに限らず視覚化システム全体にあてはまる本質的な問題である。例えば、初期のVisual Programmingシステム[glinert84]は図形システムの発展に大きく貢献したが、実用に供されているものは少ない。普通のプログラミング言語で書いた方が速いからである。これに対し、Scientific Visualizationの成功は、1つ1つ解析するにはあまりに膨大な科学的データを視覚化することによって、その解析が容易になったためである。従来の情報システムでは、扱う情報量がまだ視覚化を必要としない程度であったかもしれない。しかし、今後の超並列・超分散システムにおける情報処理は視覚化なしには無理であろう。そして、その膨大な量の情報の視覚化には2次元では不十分である。

第3に重要な点は、膨大な量の情報を扱いながら、いかにして図を簡素化するかである。例えば高分子構造モデリングシステム[koide88]では、電子軌道を含めたリアルな表示より、結合を表すリンクだけのスケルトン表示と特定のパターンだけを強調する表示の方が便利なことが多い。同様に、情報視覚化システムにおいても全情報を表示するのではなく、必要な部分だけを抽象化して表示する手法を持つ必要がある。

最後に、3次元インタラクション技術の問題がある。我々は3次元世界に生活していながら、普段の移動は2次元的であり、空間的に視点を移動するのには慣れていない。従来の研究からも分かるように、ユーザを自由に動き回らせて任意の視点から見させるより、対象とする情報の特徴に応じた視点をシステム側が予め準備し、ユーザをそこへ誘導する手法の方が効果的である。

4.1.4 おわりに

本節は、これまでに開発されたインタラクティブ3次元視覚化システムの紹介を通じて、3次元視覚化の2次元視覚化に対する主だった利点をまとめるとともに、現在の研究課題について述べた。

皮肉なことに、情報処理の専門家には視覚化システムに懐疑的な人が多い。この理由は、従来の2次元視覚化システムで扱う程度の情報は、彼ら専門家にとって視覚化するまでもないか、手書きの図で十分だったからである。しかし、情報量のさらなる増大に伴い視覚化を利用しない解析手法は早晩限界に達するであろう。また、3次元グラフィクスの表現力は、2次元グラフィクスと異なり明らかに紙の表現力を越えている。それ故にCADや分子モデリングシステムの成功があった。以上の理由から、

3次元情報視覚化システムは必要不可欠なツールとなるであろう。

また、本節では取り上げることができなかったが、現在注目を集めているTexture Mapping、Morphingといった新しいCG技術の情報視覚化への応用も興味深い研究テーマであろう。

【参考文献】

[amari93] Amari, H., Nagumo, T., Okada, M., Hirose, M., and Ishii, T.: A Virtual Reality Application for Software Visualization, Proceedings of IEEE VRAIS'93, 1993.

[Bederson94] Benjamin B. Bederson, James D. Hollan, "Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics", UIST'94, pp.17-26., 1994.

[bolt] Bolt, R. A.: The Human Interface, Lifetime Learning Publications, 198.

[brown88] Brown, M. H.: Algorithm Animation, MIT Press, Cambridge, MA, 1988.

[brown93] Brown, M. H. and Najork, M.: Algorithm Animation Using 3D Interactive Graphics, Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'93), 1993.

[card91] Card, S. K., Robertson, G. G., and Mackinlay, J. D.: The Information Visualizer, An Information Workspace, Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'91), ACM Press, 1991, pp. 181-188.

[card94] Card, S. K., Pirolli, P., and Mackinlay, J. D.: The Cost-of-knowledge Characteristic Function: Display Evaluation for Direct-walk Dynamic Information Visualization, Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'94), ACM Press, 1994, pp. 238-244.

[chen88] Chen, M., Mountford, S. J., and Sellen, A.: A Study in Interactive 3-D Rotation Using 2-D Control Devices, Computer Graphics, Vol. 22,No. 4(1988), pp. 121-129.

[chu93] 朱慧珠, 小池英樹: 3次元視覚化によるバージョン管理とモジュール管理の統合, インタラクティブシステムとソフトウェアI: 日本ソフトウェア科学会WISS'93(竹内彰一(編)), 近代科学社, 1994.

[darken] Darken, R. P. and Siberta, J. L.: A Toolset for Navigation in Virtual Environments, Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'93), ACM Press, 1993.

[fairchild93] Fairchild, K. M., Serra, L., Hern, N., Beng, L., and Leong, A. T.: Dynamic Fisheye Information Visualizations, Virtual Reality Systems, Academic Press, 1993, pp. 161-177.

[feiner90] Feiner, S. and Beshers, C.: Worlds within Worlds ¥/ Metaphors for Exploring n-Dimensional Virtual Worlds, Proceedings of the ACM SIGGRAPH Symposium on User Interface

Software and Technology (UIST'90), ACM Press, 1990, pp. 76-83.

[feiner91] Feiner, S. and Shamash, A.: Hybrid User Interfaces: Breeding Virtually Bigger Interfaces for Physically Smaller Computers, Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'91), ACM Press, 1991, pp. 9-17.

[foley] Foley, J. D.: Interfaces for Advanced Computing, Scientific American, Vol. 257,No. 4(1987), pp. 126-135.

[fsn] Silicon Graphics, Inc: {FSN: File System Navigator}, 1992.online manual.

[furnas86] Furnas, G. W.: Generalized Fisheye Views, Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'86), ACM Press, 1986, pp. 16-23.

[glinert84] Glinert, E. P. and Tanimoto, S. L.: Pict : An Interactive Graphical Programming Environment, IEEE Computer, Vol. 17,No. 11(1984), pp. 7-25.

[henderson86] Henderson, Jr., D. A. and Card, S. K.: Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-based Graphical User Interface, ACM Trans. on Graphics, Vol. 5,No. 3(1986), pp. 211-143.

[herndon92] Herndon, K. P., Zeleznik, R. C., Robbins, D. C., Conner, D. B., Snibbe, S. S.,and van Dam, A.: Interactive Shadows, Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'92), 1992, pp. 1-6.

[houde] Houde, S.: Interactive Design of an Interface for Easy 3-D Direct Manipulation, Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'92), 1992, pp. 135-142.

[ikei2] Ikei, Y., Hirose, M., and Ishii, T.: Fault Tolerant Design of Holonic Manipulator, Proc. MSET21: The International Conference on Manufacturing Systems and Environment, JSME, 1990, pp. 117-122.

[kobayashi93] 小林裕一, 小池英樹: オブジェクト指向言語C++のクラスライブラリ視覚化に関する研究, インタラクティブシステムとソフトウェアI: 日本ソフトウェア科学会WISS'93(竹内彰一(編)), 近代科学社, 1994.

[koide88] 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).

[koike91] 小池英樹: 3次元視覚化表示のソフトウェア工学への応用, 博士論文, 東京大学大学院工学系研究科情報工学専攻, 1991.

[koike92a] 小池英樹, 石井威望: フラクタルの概念に基づく提示情報量制御手法, 情報処理学会論文誌, Vol. 33,No. 2(1992), pp. 101-109.

[koike92b] 小池英樹, 石井威望: 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, 情報処

理学会論文誌, Vol. 33,No. 6(1992), pp. 778-790.

[koike93a] Koike, H.: Fractal Views: A Fractal-Based Method for Controlling Information Display, ACM Transaction on Information Systems.to appear.

[koike93b] Koike, H.: The Role of Another Spatial Dimension in Software Visualization, ACM Transaction on Information Systems, Vol. 11,No. 3(1993).

[koike93e] Koike, H. and Yoshihara, H.: Fractal Approaches for Visualizing Huge Hierarchies, Proceedings of 1993 IEEE/CS Symposium on Visual Languages (VL'93), IEEE CS Press, 1993, pp. 55-60.

[koike94a] 小池英樹: フラクタルの概念に基づく提示情報量制御手法Fractal ViewのLispプリンタへの応用, 情報処理学会論文誌.to appear.

[Kramer94] Axel Kramer, "Translucent Patches", UIST'94, pp.121-130, 1994.

[kruskal64a] Kruskal, J. B.: Multidimensional Scaling by Optimizing Goodness of Fit to a On-metric Hypothesis, Psychometrika, Vol. 29(1964), pp. 1-27.

[kruskal64b] Kruskal, J. B.: Non-metric Multidimensional Scaling: A Numerical Method, Psychometrika, Vol. 29(1964), pp. 28-42.

[Liang93] Jiandong Liand and Mark Green, Geometric modeling using six degrees of freedom input devices, in 3rd International Conference on CAD and Computer Graphics, pp.217-222, 1993.

[lieberman89] Lieberman, H.: A Three-Dimensional Representation for Program Execution, Proceedings of 1989 IEEE Workshop on Visual Languages, IEEE CS Press, 1989.

[Lieberman94] Henry Lieberman, "Powers of Ten Thosand: Navigating in Large Information Spaces", UIST'94, pp.15-16, 1994.

[mackinlay90] Mackinlay, J. D., Card, S. K., and Robertson, G. G.: Rapid Controlled Movement through a Virtual 3D Workspace, SIGGRAPH'93 Conference Proceedings, 1990, pp. 171-176.

[mackinlay91] Mackinlay, J. D., Robertson, G. G., and Card, S. K.: The Perspective Wall: Detail and Context Smoothly Integrated, Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'91), ACM Press, 1991, pp. 173-179.

[Mackinlay94] Jock D. Mackinlay, George G. Robertson, and Robert DeLine, "Developing Calendar Visualizers for the Information Visualizer", UIST'94, pp.109-118, 1994.

[myoi91] Myoi, T., Amari, H., Inamura, K., Koike, H., Kuzuoka, H., Hirose, M., Ishii, T., and Hayashi, T.: A Method of Large-Scale Control System Design Aided by System Visualization Technology, Trans. of IEE Japan, Vol. 111-C,No. 5(1991), pp. 194-201.(in Japanese).

[pavane] Cox, K. C. and Roman, G.-C.: Visualizing Concurrent Computations, Proceedings of 1991

IEEE Workshop on Visual Languages, IEEE CS Press, 1991, pp. 18-24.

[poltrock] Poltrock, S. E.: 1992.private communication.

[prograph] Cox, P. T. and Pietrzykowski, T.: Using a Pictorial Representation to Combine Dataflow and Object-Orientation in a Language Independent Programming Mechanism, Proceedings International Computer Science Conference, IEEE, 1988.

[rao94] Rao, R. and Card, S. K.: The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus+Context Visualization for Tabular Information, Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'94), 1994, pp. 318-322.

[reiss93] Reiss, S. P.: A Framework for Abstract 3D Visualization, Proceedings of 1993 IEEE/CS Symposium on Visual Languages(VL'93), 1993, pp. 108-115.

[rekimoto93] Rekimoto, J. and Green, M.: Information Cube, インタラクティブシステムとソフトウェアI: 日本ソフトウェア科学会WISS'93(竹内彰一(編)), 近代科学社, 1994.

[robertson89] Robertson, G. G., Card, S. K., and Mackinlay, J. D.: The Cognitive Coprocessor Architecture for Interactive User Interfaces, Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology (UIST'89), 1989, pp. 10-18.

[robertson91] Robertson, G. G., Mackinlay, J. D., and Card, S. K.: Cone Trees: Animated 3D Visualizations of Hierarchical Information, Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'91), ACM Press, 1991, pp. 189-194.

[robertson93] Robertson, G. G. and Mackinlay, J. D.: The Document Lens, Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'93), ACM Press, 1993.

[sarkar92] Sarkar, M. and Brown, M. H.: Graphical Fisheye Views of Graphs, Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'92), ACM Press, 1992, pp. 83-91.

[semnet] Fairchild, K. M., Poltrock, S. E., and Furnas, G. W.: SemNet: Three-Dimensional Graphic Representation of Large Knowledge Bases, Cognitive Science And Its Applications For Human-Computer Interaction(Guindon, R.(ed.)), Lawrence Erlbaum Associates, 1988, pp. 201-233.

[shaw93] Shaw, C., Green, M., Liang, J., and Sun, Y.: Decoupled Simulation in Virtual Reality with the MR Toolkit, ACM Transaction on Information Systems, Vol. 11,No. 3(1993).

[Shaw94] Chris Shaw and Mark Green, Two-Handed Polygonal Surface Design, UIST'94, pp.205-212, 1994.

[shneiderman83] Shneiderman, B.: Direct Manipulation: A Step Beyond Programming Languages, IEEE Computer, Vol. 16,No. 8(1983), pp. 57-69.

[smalltalk] Goldberg, A. and Robson, D.: Smalltalk-80: The Language and its Implementation, Xerox,

1983.

[Staples93] Loretta Staples, "Representation in Virtual Space: Visual Convention in the Graphical User Inteface", INTERCHI'93, pp.348-354, 1993.

[stasko93] Stasko, J. T. and F.Wehrli, J.: Three-dimensional Computation Visualization, Proceedings of 1993 IEEE/CS Symposium on Visual Languages (VL'93), 1993.

[takada93] 高田哲司, 小池英樹: 並列言語Lindaのプログラムの実行状態の視覚化, インタラクティブシステムとソフトウェアI: 日本ソフトウェア科学会WISS'93(竹内彰一(編)), 近代科学社, 1994.

[takemura92] Takemura, H. and Kishino, F.: Cooperative Work Environment using Virtual Workspace, Proceedings of ACM 1992 Conference on Computer-Supported Cooperative Work, 1992, pp. 226-232.

[tango] Stasko, J. T.: TANGO: A Framework and System for Algorithm Animation, Computer, Vol. 23,No. 9(1990), pp. 27-39.

[venolia] Velolia, D.: Facile 3D Direct Manipulation, Proceedings of INTERCHI'93, 1993, pp. 31-36.

[zeus] Brown, M. H.: Zeus: A System for Algorithm Animation and Multi-View Editing, 1991 IEEE Workshop on Visual Languages, October 1991, pp. 4-9.

[zimmerman87] Zimmerman, T., Lanier, J., Blanchard, C., Bryson, S., and Harvill, Y.: A Hand Gesture Interface Device, Proceedings of the ACM Conference on Human Factors in Computing Systems and Graphics Interface (CHI + GI 1987), ACM Press, 1987, pp. 189-192.

4.2 3次元を用いた情報アクセス

コンピュータを利用した情報検索のユーザインタフェースは、利用できる端末の変化とともに変わってきた。初期のユーザインタフェースは、端末からコマンドを入力し結果が端末に表示されるというもので、タイプライタ型の端末が使われることが多く、表示はラインで制御されていた。DIALOG、JOIS等の代表的な商用データベースサービスがこれに相当する。これらを利用するためには、当然のことながら用意されたコマンドがどのようなものか知っている必要があり、かつシステムごとにコマンドの名称、体系が異なるため、ユーザの負荷は高かった。検索の専門家は別にして、たまにしかデータベースサービスを利用しない一般ユーザにとって、使いやすいものとはいえない。標準データベース言語としてSQLが登場し、多くのDBMSがこれに対応するようになったが、SQL自体が決してエンドユーザにとって使いやすいといえるものではない。このようなインタフェースを1次元ユーザインタフェースと呼ぶことができよう。

パソコンの性能がよくなるにつれ、ユーザインタフェースに画面を使うものが多くなった。いわば、2次元ユーザインタフェースである。初期にはユーザにむき出しであったコマンドを、画面をかぶせることにより、ユーザの目に触れないようにすることで、一般ユーザに親しみやすさを感じさせ、抵抗感を減じている。代表的なものがQBE（Query By Example）であり、表示例に真似て属性に制約を与えることにより検索を行う。また、パソコン上で動作する多くのDBMSは検索画面、結果表示画面等をユーザが簡単に作成できるようになっている。

上記のユーザインタフェースは、1次元、2次元の差はあるものの、ともに属性に対する制約を与えるという方法を採っている。ユーザがスキーマを熟知していて、データベースを使ってなすべきことがはっきりしている定型的作業の場合はこれで十分であろう。一方、今まで未知の何か新しいテーマに関する情報を探すといったような場合、ユーザ自身が自分が本当は何を探しているのかはっきりとは分かっていないことが多く見受けられる。また、通常、探しているものの概念をキーワード等にマッピングして検索するわけであるが、ユーザとデータベース作成者との間では、同一文字列を持ったキーワードであってもそれに含まれている概念が一致するとは限らない。結局のところ、個々のオブジェクトの内容を見て、自分が探しているものはこれであったということが初めて分かることが多い。このような場合の検索の方法として、オブジェクトを空間に配置してユーザはそれを見ながら探索的に検索するという方法がいくつか提案されている。

オブジェクトを配置する空間としては、2次元を使う方法と3次元を使う方法がある。人は3次元空間で生活しているわけだが、日常生活では地面のような2次元平面をまず基準として、これにそれと垂直な方向の要素を付け加えることが多く、3次元の3つの方向を同時に等価に扱うことはまれである。この意味で、2＋α次元が人の認識方法に合っているのかもしれない。ところで、オブジェクトを配置する場合、その個数が多くなればなるほど大きな空間が必要となる。現実にあるディスプレイは限

られた空間であり、そこにより多くのオブジェクトを配置しようとすれば、2次元より3次元のほうが有利である。しかし、3次元空間をナビゲートする方法が問題となる。

以下に、3次元を使った情報アクセスの事例として、Bead、LyberWorld、GUI of P/FDMを紹介する。2次元の情報アクセスの例としてはPad、Pad++、3次元で扱ったものとしては、Cone tree、VOGUE、Information Cube等があるが、4.1節で紹介されているのでここでは触れない。

(1) Bead

従来の情報検索システムでは、ユーザは専用の質問言語又は自然言語を使って検索を行う。ユーザがこれらの方法で的確に自分の要求を表現できなかったり、あるいはそもそも自分が何を探しているのか本当は明確に分かっていない場合、これらの方法は適当とはいえない。そこで、データベースのコーパス全体を表示することで、何が有効かを大まかに把握し、徐々に質問を改良していく方法が試みられた。Bead[Charmers92, Charmers93]は、ドキュメントを空間上の点として表示し、ドキュメントの主題の類似度を空間上の点の幾何学的距離に反映させて表示させる。これによって、各ドキュメントの個々のドキュメントとしての役割（タイトル、著者名等）と、ドキュメント集合の中での要素としての役割を視覚化することを狙ったものである。初期のバージョンでは表示空間に3次元を、改良版では2.1次元の表示空間を使っている。

N個のドキュメントの集合D = {A, B, C,...}に対応するN個の粒子の集合P = {a, b, c,...}を考える。予め付けられたキーワードリストを基に2つのドキュメントA、 Bの距離D(A, B)を定義する。D(A, B)は、AとBのキーワードリストが完全に一致したとき0、全く異なったとき1となり、キーワードリストの部分的一致の状態に応じて、0と1の間の値をとる。一方、2つの粒子a、 bの距離d(a, b)は表示空間上での幾何学的距離とする。つまり、D(A, B)は2つのドキュメントがとるべき「望ましい」距離であり、d(a, b)は表示空間上でのある時点での2つの粒子の距離である。表示空間は、減衰スプリングモデルに従う引力／斥力が働く空間であり、2つの粒子にはD(A, B) - d(a, b)に線形比例する力が働く。この空間にランダムに配置されたN個の粒子は、それぞれ上記の物理法則に従って運動し、やがて粒子系のエネルギーが最小となったとき安定する。このときの粒子の位置が求める配置である。

これはN体問題であり、O(N**2)の複雑さをもち収束がきわめて悪い。このため、階層的空間分割法という近似法を使って、複雑さをO(NlogN)に落としている。さらに、初期状態をN個の粒子がランダムに配置された状態として開始すると効率が悪いため、初期状態は粒子が少数の状態から始め、何回かの粒子系の配置計算の後、新しい粒子を系へ追加するということを繰り返すという方法を使って効率を上げている。

この結果、3次元空間にはドキュメントの点によって構成された「雲」が見えることになる（図4.2-1)。この表示方法には、①ドキュメントの点は「雲」の内部にも配置され見にくい、②表示構造が複

雑になりがち、③基準となるものあるいはランドマーク的特徴がないといった欠点がある。このためユーザは、自分がどこにいるか、どちらを向いているか分かりにくくナビゲートが難しい、またドキュメントのコーパスのメンタルモデルを作りにくい。

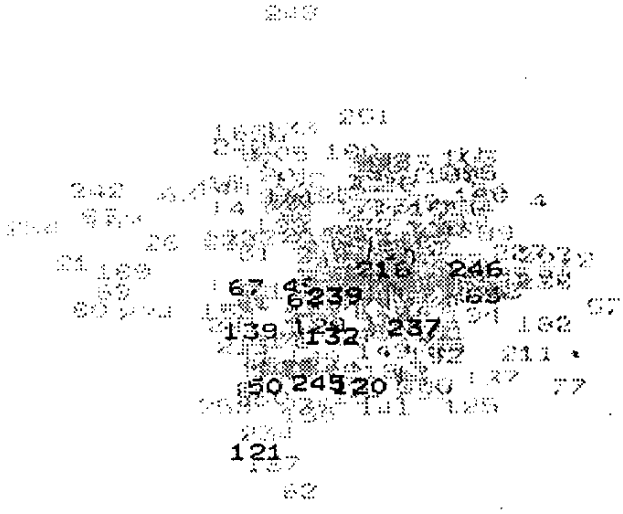


図4.2-1　Beadの表示（3次元）（出典：[Charmers93]）

「infomation retrieval」という語によるサーチによってマッチしたドキュメント番号がハイライト表示されている。ドキュメントは大部分中央のクラスタにあるが、サーチにマッチしたドキュメントはコーパス全体に渡って散らばっている。雲の内部が見にくいことや全体の複雑さがコーパスの理解を妨げている。

改良版では、この欠点を克服するため2.1次元のランドスケープモデルを使っている。人の日常生活は3次元空間で営まれているが、3方向に等質な3次元空間ではなく、垂直方向よりも水平方向により大きな広がりを持つと考えられ、完全な3次元よりも2.1次元のほうが人の感覚に合うと思われる。ランドスケープモデルでは、3次元表示におけるドキュメント間の相対距離の正確さは犠牲にし、アクセスしやすさとなじみやすさに重点をおいている。

ドキュメントの位置は主題に一貫性が見られるように決められ、個々のドキュメントは色付きのマーカで表される。ランドスケープには、いくつかのマーカ密度の高い部分と低い部分ができる。これによって、ランドスケープ全体はいくつかのクラスタに分かれていることが視覚的に分かる。各クラスタの中心部、密度の高い部分は主題に関連の強いドキュメントが位置し、周辺部には関連の低いドキュメントが位置する（図4.2-2）。

ランドスケープ全体はコーパス全体のビューを与え、その中の密度の濃淡はランドマーク、人の視点が向きやすい領域、クラスタ間の境界等を与える。このようなめりはりのある表示によって「景観」が生まれコーパスのメンタルモデルを作りやすくなる。

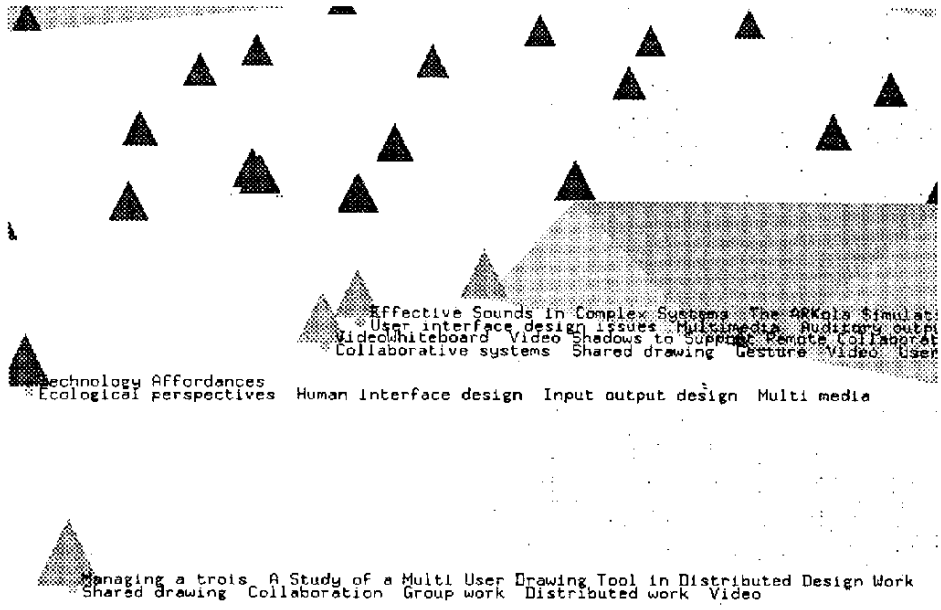


図4.2-2　Beadの表示（2.1次元）（出典：[Charmers93]）

「collaborative」のいう語によるサーチ後、コーパスの中心部にある「谷」の近くにあるドキュメントがハイライト表示されている。マウスを使ったブラウジングによって、マッチしなかったドキュメントのタイトル（Technology Affordances）が表示されている。このドキュメントは、マッチした論文の一つ（Effective Sounds in Complex Systems）の近くにあり、二つとも著者がともにBill Gaverである。

(2)　LyberWorld

LyberWorld[Hemmje94]は、フルテキスト検索システムINQUERYのユーザインタフェースのプロトタイプとして開発された。データベースの内容を3次元に視覚化し、インタラクティブに操作することにより情報を探索的に検索することを狙っている。データモデルとして、ドキュメントとタームをノードとし、タームのドキュメントに対する寄与度をドキュメントとターム間のリンクの長さで表すネットワークモデルを使っている（寄与度は、INQUERYによって計算される）。要求される機能としては、ナビゲーション（空間移動と方向付け）機能、情報アイテム（ドキュメントとターム）が適切かどうかの判断を助ける機能、ドキュメントのクラスタリング機能、サーチが十分かどうか、これ以上検索を続ける必要があるかどうかの判断を助ける機能、ドキュメントの詳細を表示する機能がある。ナビゲーション機能は、NavigationConesによって、他の機能はRelevanceSpheresによって実現されている。

NavigationConesは、ドキュメントとタームのネットワーク構造を階層ツリー構造に変換して表示する。リンクの長さは情報アイテム間の寄与度に無関係に等しく表される。一つのタームが複数のド

キュメントに関係する場合は別々のノードとして表示し、hidden pathでそれらが同一のタームであることを示す。一般にネットワーク構造を3次元空間上のグラフとして表示する場合、リンクの交差を避けるとかノードの位置決め等のレイアウト上の複雑な問題が生じ、システムの計算負荷が高いだけでなく、これを見るユーザの認知負荷も高いが、ツリー構造ではこれらの負荷が小さくなる。NavigationConesでは、データベースの全ツリーを一度に作るのではなく、ユーザがタームを指定することにより、そのタームを親ノード、そのタームが関係するドキュメントを子ノードとするconeが作られ表示される（図4.2-3)。このとき、画面手前に表示された子ノードが親ノードに対して最も大きい寄与度を持つものであり、画面向こう側にいくに従い寄与度が小さくなる。ここで、ユーザが子ノードの一つであるドキュメントを選択すると、そのドキュメントを親ノード、ドキュメントに関係するタームを子ノードとするconeが作られ表示される。続いて別の兄弟のノード（この場合ドキュメント）を選択することも、子ノード（この場合ターム）を選択することもできる。また親ノードに遡ることもできる。

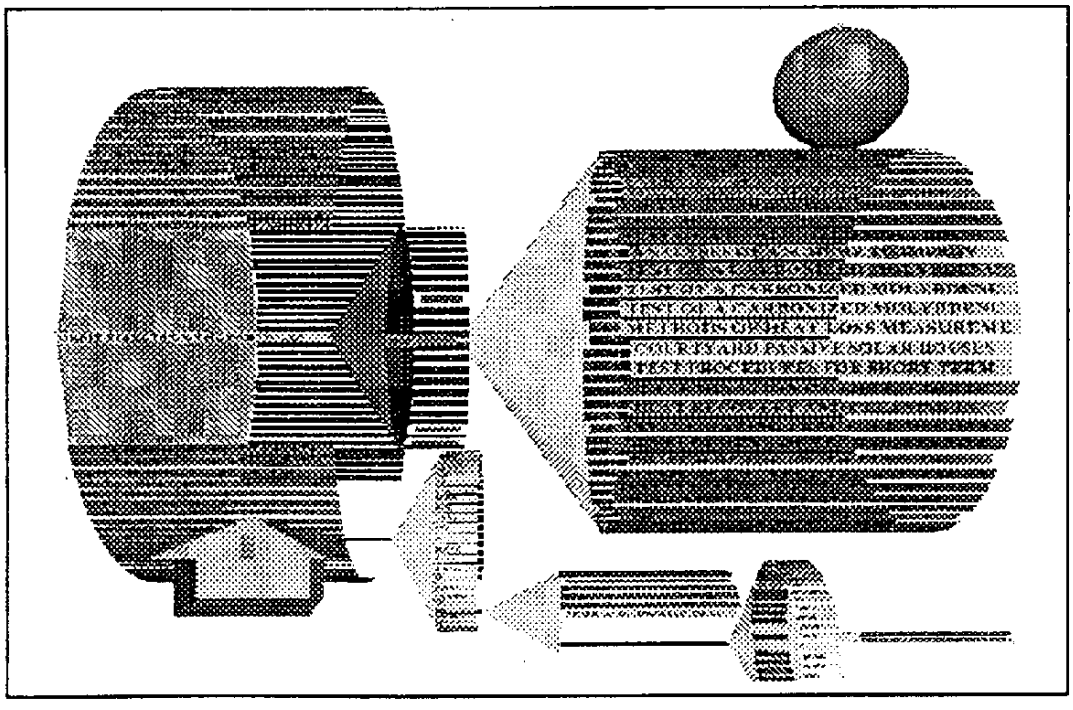


図4.2-3　NavigationConesの例（出典：[Hemmje94])

RelevanceSpheresは、NavigationConesで得られたドキュメントが検索結果として適切かどうかの判断を助けるツールである。球面上にタームを等距離に配置し、各タームのドキュメントに対する寄与度に応じてドキュメントを球内に配置する。例えば、タームA、B、C、Dに対するドキュメント

docの位置は次のように決まる。タームを互いに等距離を保つよう球面上に配置し、ドキュメントdocを球の中心に置き、球の中心を始点として各ターム方向の向きを持ち、長さがタームの寄与度となるベクタRA、RB、RC、RDを考える。次にベクタRAの始点を球の中心に置き、4つのベクタの和RA+RB+RC+RDをとる。終点がドキュメントdocの配置されるべき位置である。これによって、あるドキュメントに対しどのタームが最も寄与度が高いかを球面上のタームと球内のドキュメントの幾何学的配置から知ることができる（図4.2-4）。

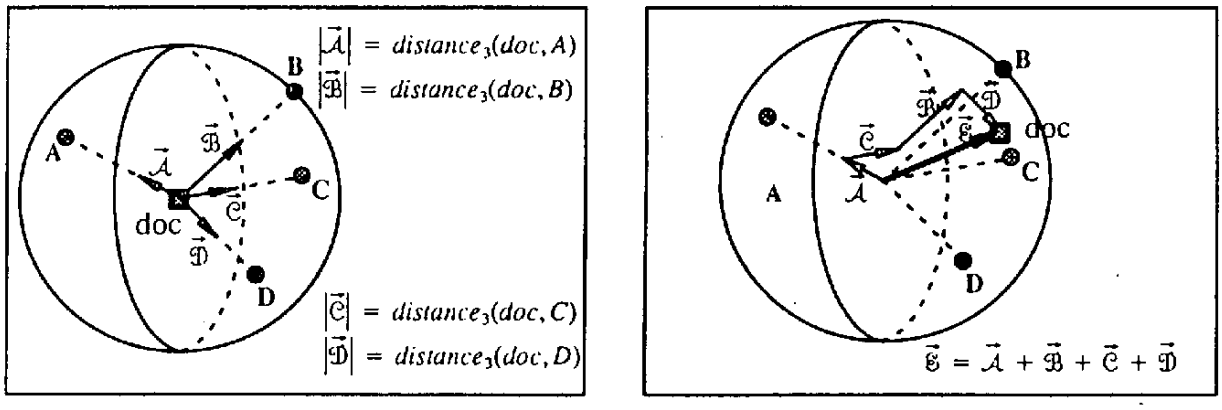


図4.2-4　RelevanceSpheresにおけるドキュメントの位置決め（出典：[Hemmje94]）

深さ方向の位置は分かりにくいが、球全体を回転させることによって見やすくできる。

ドキュメントのクラスタリングはタームを球面上で移動させることにより行う。3つのタームA、B、Cに関係するドキュメントを集める場合は次のようにする。球面上のタームA、B、Cと球の中心で作られる三角錐（ただし、A、B、Cで作られる面は球面）を考えると、この三角錐の内部に配置されたドキュメントはこれらのタームに関係するドキュメントである。タームA、B、Cの位置を移動させると、ドキュメントもそれに応じて移動し、より幾何学的に見やすくすることができる。

三角錐中にタームAまたはBまたはCの寄与度が他のドキュメントに比べ極めて高いドキュメントが一つ（ないし複数）あり、他の多くのドキュメントの寄与度が低い場合、寄与度の高いドキュメントは球面のすぐ近くに表示され、他のドキュメントは球の中心近くにまとまって表示されることになり、球の中心近くに表示されたドキュメント間の相違はほとんど分からない。このような場合、寄与度の極めて高いドキュメントを球外に出してしまう、つまり一時的にドキュメントの位置計算の対象から外すことにより、他のドキュメントだけで再配置を行うことができる。

この他、タームの寄与度を変化させてみる機能、ドキュメントがタームの寄与度が低いクラスタと高いクラスタに分かれた場合、同心球面で低いクラスタを分離する機能がある。

ドキュメントの詳細は、InformationRoomというツールによって表示させることができる。

(3) GUI of P/FDM

GUI of P/FDM[Boyle93]はP/FDMという蛋白質を扱ったオブジェクト指向型データベースのグラフィカルユーザインタフェースである。

起動するとP/FDMのスキーマが3次元表示される。エンティティは立方体、エンティティ間の関係は実線で表示される。エンティティが階層関係にある場合は関係は太い実線で表示され、階層下位のエンティティは上位のエンティティより幾何学的に下のレベルに表示される。ただし、スキーマ表示を真上から見たとき、階層上位のエンティティによって下位のエンティティが隠されないように配置される。これにより、3次元の操作に慣れないユーザは2次元で操作することができる。

質問は3次元表示されたスキーマのエンティティに質問を付加することにより行う（図4.2-5）。エンティティを選択すると質問用ウィンドウがポップアップし、属性に制約を与えることで「部分質問」を作成する。他のエンティティに質問を付加したい場合は表示されたスキーマをナビゲートして、エンティティへの「部分質問」を与える。与えたすべての「部分質問」を組み合わせたものが全体の質問となる。

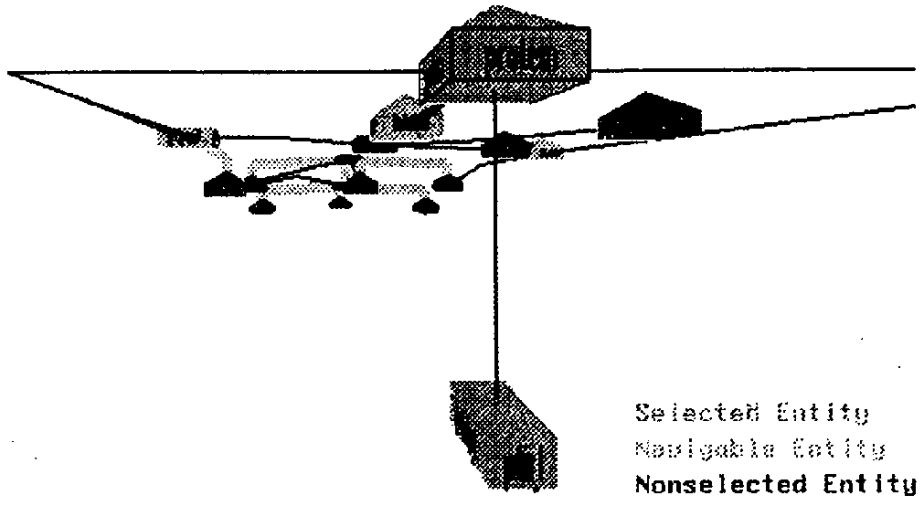


図4.2-5 スキーマの3次元表示（出典：[Boyle93]）

結果の表示は3次元でなされる。質問結果として得られた蛋白質は一つ一つが立方体で表示され、結果全体は3次元の迷路状に表示される（図4.2-6）。

(4) ZoomFinder

(a) 概要

ZoomFinderは、人が実世界でものを探すのと同様な方法で計算機内の情報を探すことを実現する

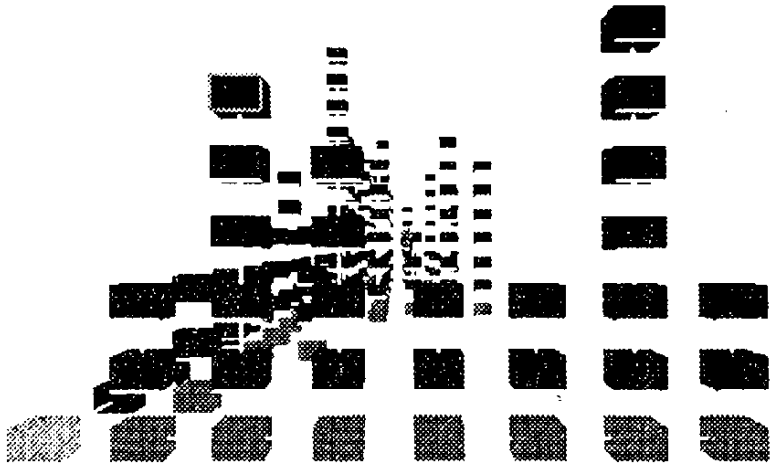


図4.2-6　結果表示例（出典：[Boyle93]）

ことを狙ったプロトタイプシステムであり、HMDやデータグローブのような特別な機器は使用せず、入力装置としては、マウスとキーボードのみという一般のオフィスに設置されているワークステーションやパソコンの機器構成で操作されることを前提にしている。

取り扱う情報構造としては、とりあえず最も分かりやすい階層構造を取り扱うこととし、その典型的な例であるUNIXディレクトリをデータとして使った。親ディレクトリを円錐の頂点に、子ディレクトリを円錐底面の円周上に等間隔に配置し、親ディレクトリと子ディレクトリを線分で結んでいわゆる「cone」を作る。ディレクトリの階層に従って再帰的にconeを作り、ディレクトリ階層全体としてはいわゆる「cone tree」を作ることにより、階層構造を表示した。全体を一度に表示するかユーザの指定した部分のみを表示するかについては、日常世界同様、他のものの陰に隠されて見えないこともあるが、ともかく存在はしていることからすべて表示することとした（実際には、ルートディレクトリをトップノードとした計算機システムの全ディレクトリ、全ファイルを対象とするのではなく、ユーザが指定したディレクトリをトップノードとして、その配下のディレクトリ、ファイルを対象としている。もちろん、ルートディレクトリを指定することはできる）。

(b)　ものの探索

人がものを探す状況はいくつかに分けることができる。①探す対象に関しては、それが明確な場合と不明確な場合がある。明確な場合とは、探す本人がそれが何であるかが分かっている場合である。不明確な場合とは、探す本人にとって分かったつもりでも実はよく分かっていない場合である。②対象の存在に関しては、実在することが明確な場合／実在するかどうか不明確な場合、③対象の場所に関しては、それがどこにあるか明確な場合／不明確な場合、④対象に関する情報に関しては、情報を

豊富に持っている場合／わずかしか持っていない場合等がある。

このような状況の違いによって、探すために採られる行動も異なってくる。探しているものがはっきりしていてどこにあるか分かっている場合は、まっすぐそこに行き、手前に何かある等で目的のものに手が届かない場合は邪魔なものを動かしてやったり、引き出しや箱に入っていれば開けて中身を取り出す。探しているものははっきりしているがどこにあるか分からない場合は、まず、全体を見渡し、ありそうだと目星をつけたところに移動する。もしこのとき探索のための何らかの手がかりがあれば、探索の効率を上げることができる。手がかりとしては、探しているものに対する知識、探索空間の構造に関する知識等が考えられる。なければ、端からもう一方の端まで順に探したり、手当たり次第に調べることになる。探しているものが分かったつもりで本当は分かっていない場合は、探索行動としては本当に分かっている場合と同様であるが、探していると思い込んでいたものの詳細を調べてこれは実は違うということが分かり、別のところを探す。

このようなものの探索で見られる人の動作には、全体を見渡す、ある限られた場所に目をやる、人が移動する、別の方向から見る、ものを動かす、詳細を調べる等の基本動作がある。ものの配置に規則性があったり、目星を付ける手段があれば探索の効率は上がる。

(c) 要求される機能

(i) 全体構造の表示

実世界では、ものは人が認識するか否かによらず物理的実態として存在している（そうではなく、認識されたもののみが存在するという立場もあろうが）。計算機内の情報も同様である。これらを表示する方法として、最初にすべてを一度に表示してやる方法となんらかの基準にあったもののみを表示してやる方法が考えられるが、すべてを一度に表示する方法を採用した。

(ii) 視点の移動

人が移動したり、別の方向から見たりする動作に対応する。移動に関しては、画面に平行な面上での水平、垂直方向の移動と画面に垂直な方向 の移動を、方向の変更についてはcone treeの性質を考慮して、ルートノードを通るconeの軸を中心とする周回移動を採用した。モデルは固定されたままである。

ルートノードを通るconeの軸を中心にモデル全体を回転させても同様の効果が得られるが、実世界でものを探すときには、対象全体を動かすということはまれであるので、モデルではなく視点の周回移動を採用した。

(iii) 拡大／縮小

限られた場所に目をやる動作に対応する。拡大はモデルに近づき、さらにモデルの内部に入っていく動作に、縮小はモデルから遠ざかる動作に対応する。

(iv) ノードの一時消去

別のものの陰に隠れたものを見るために、邪魔しているものを動かす動作に対応する。

(v) ノード名によるハイライティング

ものへの名前の付け方には種々の方法があるが、最も一般的に行われるのは内容を表す名前を付ける方法であろう。結果として内容が似通っていれば似た名前が付けられることになる。目星を付ける手段として名前によるハイライティングを採用した。

(vi) 日付によるフィルタリング

最近話題の「超整理法」でも日付だけによる整理を推奨しているが、オブジェクトをいつごろ操作したかというのは人の記憶に残りやすい。このことから、操作日付によるフィルタリングを目星を付ける手段として採用した。

(d) 実現したプロトタイプ

操作例をとおして実現方法を示す。

① 最初に某ディレクトリより下の階層構造の概観を見る（図4.2-7)。

② ZoomInしてみる（図4.2-8)。

③ 日付によるフィルタリングを行う（図4.2-9)。

④ 名前によるハイライティングを行う。該当部分の色が変わる（図4.2-10)。

⑤ 場所を変えてZoomInとZoomOutを行う（図4.2-11)。

(e) システム構成

(i) ハードウェア

SUN SparcStation10GS(メモリ：64Mb）上に作成した。キーボード、マウス以外の機器は付いていない。

(ii) ソフトウェア

SUNOS5.3+X11R5+Motif1.2.2の上にアプリケーションを作成した。Xサーバは、OSにバンドルされているものではなく、X11R5に含まれているサンプルサーバにカラー表示を可能とするパッチをあてたもの（いわゆるXsun24）を使っている。3Dグラフィクスライブラリは PEXlibを使った。

(f) 考察

あまり多く実験したわけではないが、ZoomFinderの利用をとおして以下のことが明らかになった。

(i) ZoomIn/ZoomOutの効果

ZoomInは、単にモデルの一部を拡大するというものではなく、モデルの中にまで視点を移動する

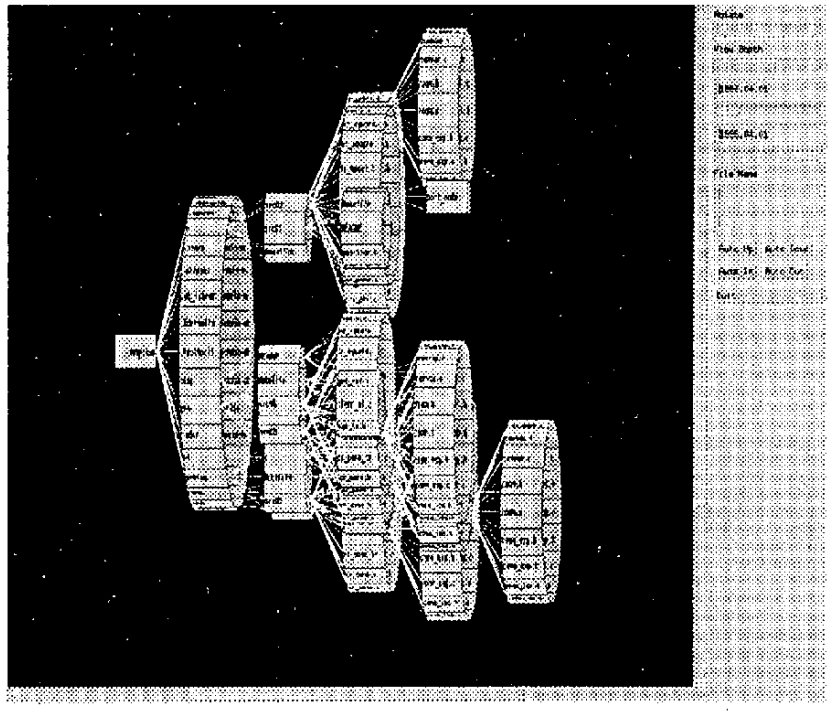


図4.2-7　某ディレクトリの階層構造の概観

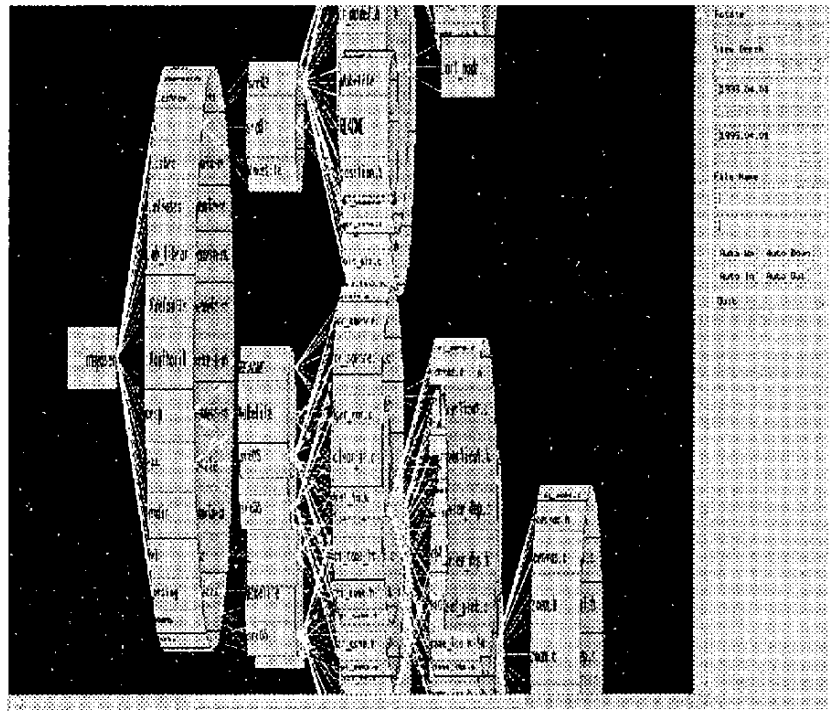


図4.2-8　ZoomInしたところ

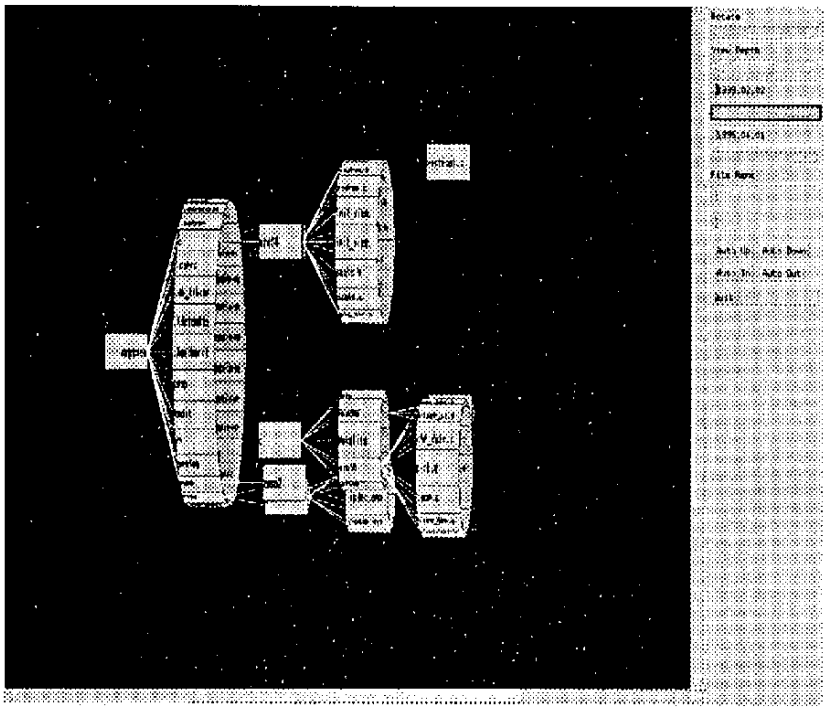


図4.2-9　日付によるフィルタリングを行ったところ

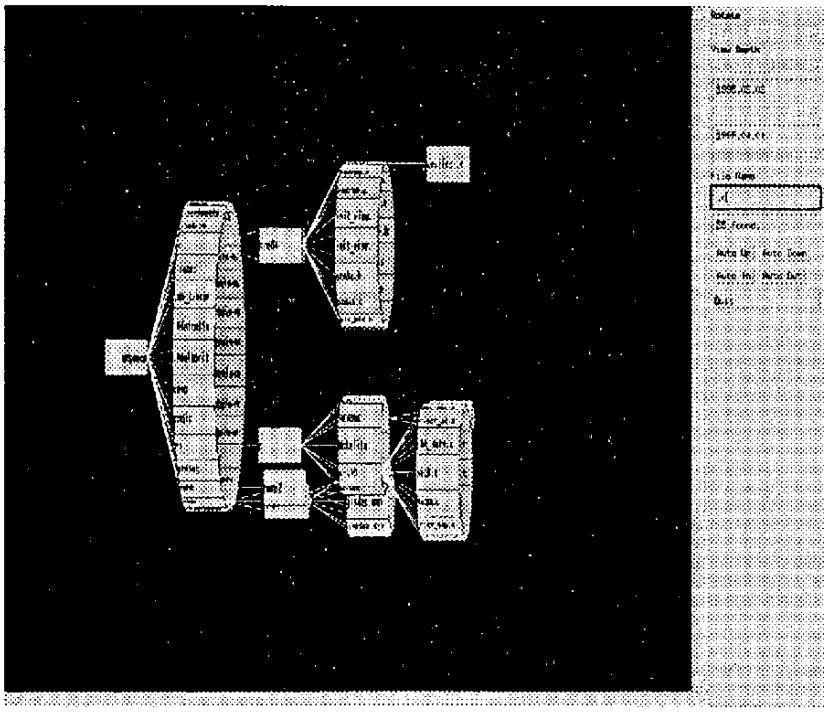


図4.2-10　名前によるハイライティングを行ったところ
（実際は該当部分の色が変わっているが、印刷の関係上分からない）

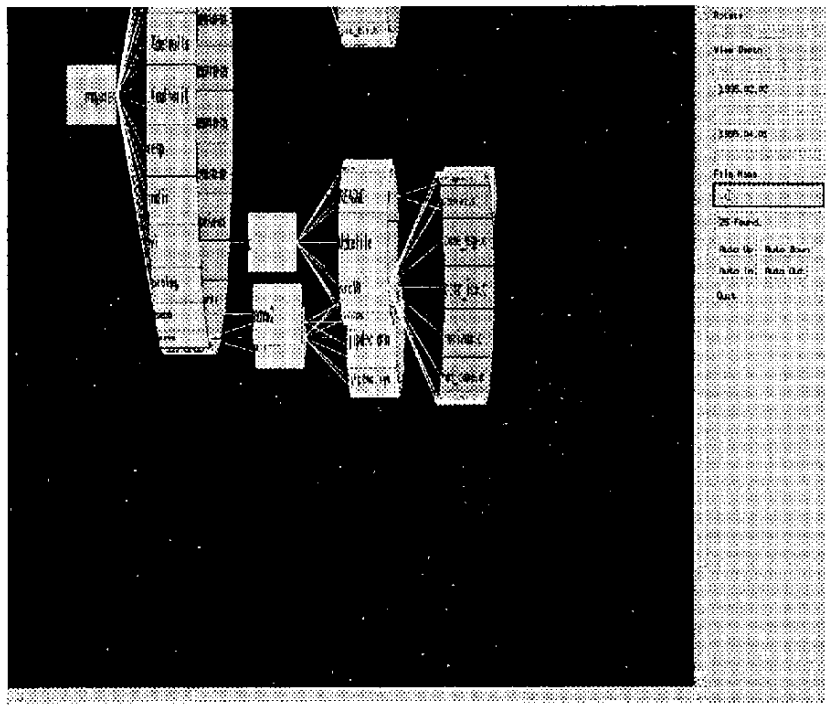


図4.2-11　ZoomInしたところ

過程で目の前にあるオブジェクト（の群れ）を拡大するというものであるが、オブジェクトのブラウジング、目的のオブジェクトの探索に効果がみられた。

(ii)　フィルタリングの効果

モデルのブラウジングは、モデルの回りでの視点の周回移動とZoomIn/ZoomOutで一応可能であるが、オブジェクトの個数が多くなればなるほど、いかに無関係なオブジェクトを目に触れないようにするかが重要となる。時間というパラメータが関係するオブジェクトでは、当然のことであるが時間によるフィルタリングは大変有効となる。また、名前によるハイライティングは、名前が内容等を反映して付けられている等、なんらかの規則に従って付けられていれば、探索の目安になる。

(g)　今後の課題

(i)　最適モデルの検討

今回のモデルには、coneどうしが3次元上で重なっているという大きな欠点がある。また、たとえ重ならないようなモデルであっても、画面の奥に表示されたオブジェクトは、手前のオブジェクトに邪魔されて見えず、ユーザは隠されたオブジェクトを見るために余分な操作をせざるをえない。

coneどうしが重ならないようなモデルを構築するアルゴリズムを導入することと、初期表示において、手前のオブジェクトによって隠されるオブジェクトが少なくなるような視点から見たモデルを

表示することにより、ユーザに余計な操作をさせないことが必要であろう。

(ii) 半透明表示

3次元のモデルである以上、画面手前のオブジェクトがその奥のオブジェクトを隠すのはやむを得ないが、半透明表示を使うことにより、隠されたオブジェクト見るためのユーザの余分な動作を軽減することができるのではないかと思われる。

(iii) 滑らかなアニメーション

インタラクティブなシステムである以上、ユーザを苛立たせないだけの応答速度を実現するのが当然であるが、残念ながら十分には実現できなかった。また、視点あるいはモデルが動く場合、アニメーション効果を実現する機能を組み込んだが、これも十分に効果をあげていない。以下のような理由によると考えられる。

① ハードウェアの面では、使用したワークステーションがごく一般的なものであり、特に3次元CG向けというものではないということ。

② ソフトウェアの面では、利用したX11R5のPEXが、陰面消去、陰線消去機能を有していないということ。このため、coneの表示順序をアプリケーションで制御した。線、面、文字等のプリミティブは画面に上書きされるため、各coneレベルごとに、Z座標値の小さいconeから順に、各coneでは、Z座標値の小さなノードからPEXに送っている。このため、Z座標の値が変わるような操作がなされた場合は、全体を書き直しており、モデルが大きくなると極めて時間がかかることになった。

③ ソフトウェアの面からは、もう一つ、スキルの不足によりXないしPEXの性能を十分に生かしきれなかったのではないかということ。

ハードウェアやPEXはさておき、アプリケーション側で対応可能な対策をとる必要がある。

(iv) 影の効果

今回は、試験的にはともかく、光源は導入しなかった。3次元の効果を高めるため、あるいは見栄えをよくするために陰影や影を付けることが必要かもしれない。

(v) ライブラリ化

PEXlibが提供するプリミティブは、Xlib並みで低レベルのものである。PEXを利用してある程度大きなシステムを効率よく作るためには、3次元の部品のライブラリが必要と感じた。

【参考文献】

[Boyle93] Boyle, John et al. : Design of 3D User Interface to a Database, Lecture Notes in Computer Science 871, pp.173-185 (1993).

[Chalmers92] Chalmers, Matthew et al. : BEAD:Explorations in Information Visualization, Proceedings of the Fifteenth Annual International ACM SIGIR Conference, pp.330-337 (1992).

[Chalmers93] Chalmers, Matthew: Visualization of Complex Information, Lecture Notes in Computer Science 753, pp.152-162 (1993).

[Hemmje94] Hemmje, Matthias at al. : LyberWorld - A Visualization User Interface Supporting Fulltext Retrieval, Proceedings of the Seventeenth Annual International ACM-SIGIR Conference, pp.249-259 (1994).

4.3 テキストビジュアリゼーション

ビジュアリゼーションというと、3次元のカラーを多用した図形を先ず思い浮かべてしまう。しかし、計算機から人間に情報を伝える際の基本となるテキストデータも、ただ文字の列として提示されるだけではなく、ビジュアルな変形を施されて、視覚的に提示されることも多い。この節では、このようなテキストの視覚的提示と、その大規模なドキュメントへの応用について述べる。

4.3.1 ページの視覚化

テキストをディスプレイに表示する時には、ページという単位を使うことが最も多い。ページの上に文字を横に並べて行を作り、行を縦に並べてパラグラフを作る。いわゆるタイプセッティングであるが、これだけではあまり「視覚化」という感じはしない。たぶん一番原始的な視覚化はプログラムのタイプセットに現われる。いい例がLispのpretty printingである。これは、インデント（字下げ、段下げ）を利用してプログラムのネスティングの情報を視覚的に表示する。さらに進化したLispWEBというシステムでは、図4.3-1のような表示方法も考案されている[神林91]。ここでは、対応する開き括弧と閉じ括弧の現われる行を縦の線で示している。

26. プログラム部分で defun 式により定義されている関数が、どのモジュールで使用されているかを表示する関数。
定義されている関数名のリストは、global 変数 *define-function-names* に (symbol-name . print-string) の形で保持されている。 その各関数名について、それが使用されているモジュール番号を関数 get-function-used-module-numbers により (結果は逆順になっているので、 reverse する必要あり) 調べる。

```
(defun print-function-used-module-numbers ()
  (when *define-function-names*27
    ;; defun で定義される関数がある場合は、その関数が使われているモ
    ;; ジュール番号を表示する。
    (dolist (func *define-function-names*27)
      ;; func は、 (symbol-name . print-string) の形になっている。
      (let ((modules (reverse
                      (get-function-used-module-numbers17 (car func)))))
        (when modules
          (write-to-TeX-file60 #\Newline "{\\footnotesize␣関数{\\bf␣" (cdr func))
          (write-to-TeX-file60 "} は、セクション" (car modules))
          (dolist (num (cdr modules))
            (write-to-TeX-file60 ",␣" num))
          (write-to-TeX-file60 "で使われている。}\\par" #\Newline)))
        (setq *define-function-names*27 '()))))
```

関数 print-function-used-module-numbers は、セクション 25 で使われている。

図4.3-1 LispWEBをかけて読みやすくした出力例

物理屋さんなどの間で広く使われているダム端末でも動くTeXのプレビューアに、dvi2vduがある。このプレビューアにはbox modeというモードがあって、このモードにすると、テクトロ・エミュレーションができる端末ならば、すべての文字を単純な箱（四角形）として表示してくれる（図4.3-2）。モデム経由で使っているときなど表示速度が速くなり、文書の大体のレイアウトは一目了然となるので、大変便利なモードである。これもテキストビジュアリゼーションの好例であろう。

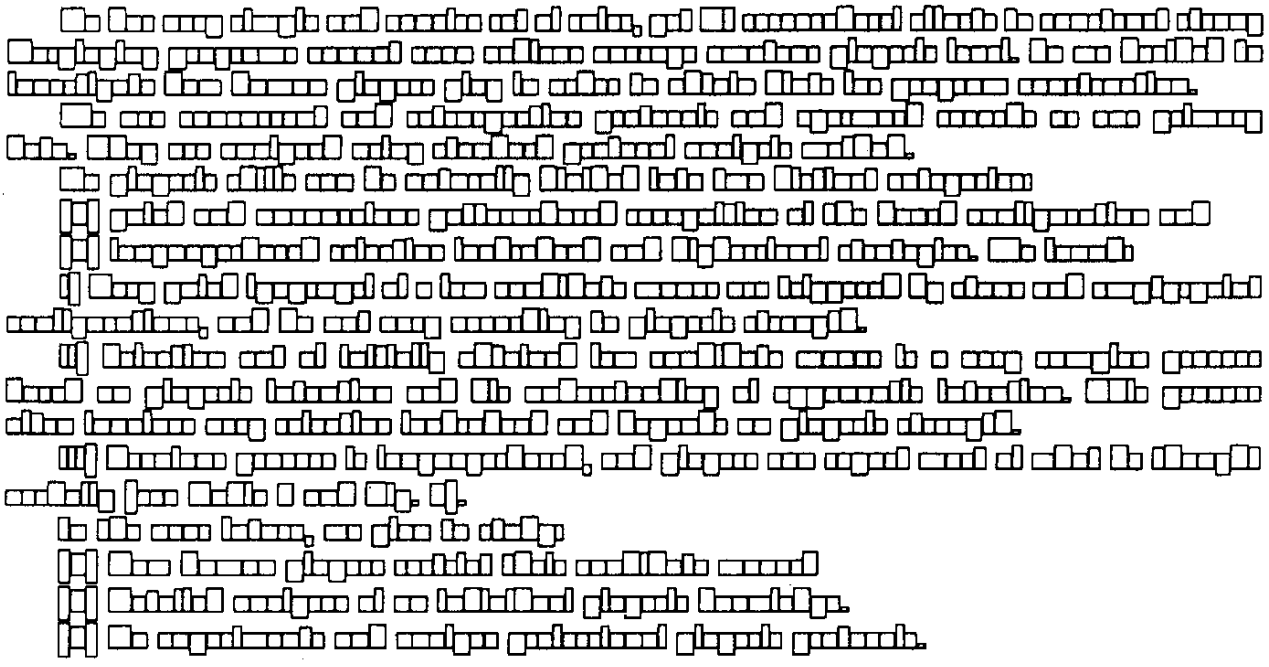


図4.3-2　dvi2vduのbox modeの表示例

さらに手のこんだものとして、ベッカーとマルカスがずいぶん昔に考案したプログラムマップがある[Baecker & Marcus90]。これは、プログラム全体の概要を知るために、プログラムテキストをタイプセットしたページを、先ず一定の縮尺率（2.5, 5, 12.5, 50などが使われるが一番手ごろなのは12.5）で縮小し、要所要所にそこで定義されている関数の名前など（annotationと呼ぶ）を書き込んだものである。こうすると、30ページほどが1ページに収まってしまう。さらに不思議なことに、こんなに縮小してannotation以外の文字は全く読めないにもかかわらず、だいたい何がどこら辺にあるかぐらいは見て取れるのである（図4.3-3参照）。彼らは、このマップを目次の代わりに使うばかりでなく、さらにこの中に実行頻度の情報を埋め込むというような使い方も提案している。何ページもの情報を1ページに詰め込んでいるという点では、次に述べる本の視覚化に一歩踏み込んでいることになる。

以上、いくつかの例で見てきたように、テキストを視覚化するというのは単なるタイプセットではなく、利用者が必要とするであろう情報をテキストから抽出し、コンパクトにうまく表現してやることであることが分かる。

Section 2.1

Program Map

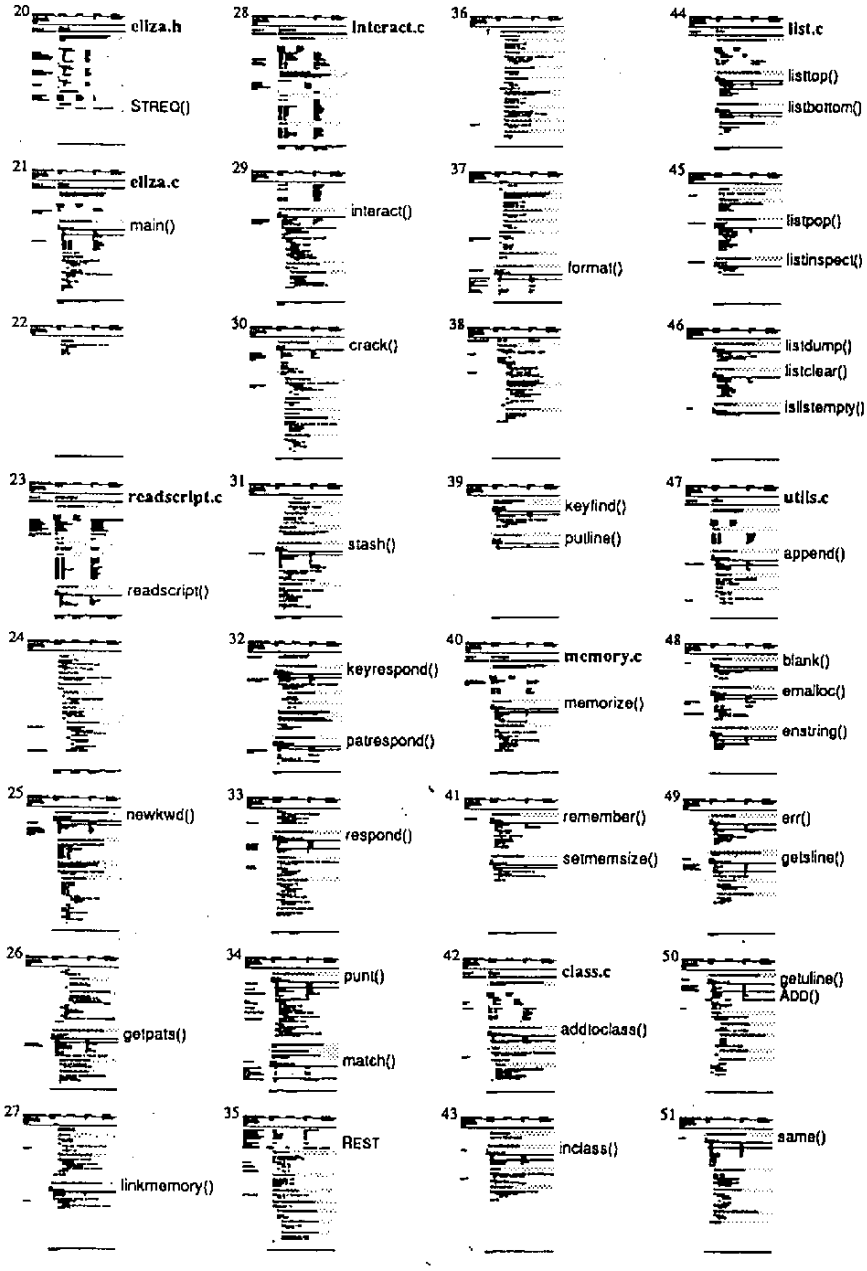


図4.3-3　プログラムマップの表示例（縮尺率は12.5）

4.3.2　本の視覚化

視覚的な本は、各種の図鑑、美術全集を始め料理の本など[斉藤93]世の中にはたくさんある。これらは元々内容が視覚的なのだから、できあがったものに図が多いのは当り前と言えば当り前である。これらは「本の視覚化」ではない。最近はやりだしたExpand Book（いわゆる電子本）は本の視覚化の例かもしれない。それらの中にも、そもそも内容が視覚的なものと、元は普通の本であったものを

電子化したものの二通りがある。後者の中には、本報告書の2.4.2の(c)「エキスパンドブック」でも議論されているように、安易に作ったためにほとんどが元のテキストのままで、あまり「視覚的」でないものも多い（例えば、Minskyの"The Society of Mind"のCD-ROM版[Minsky94]）。電子化されて検索が容易になるなどのメリットはあるものの、ディスプレイのクオリティがまだまだ紙に追いついていない現状では、あまり利益がない。目次から特定の章に飛んだりする機能などが付加されているが、これぐらいのことならばページ数さえ書いてあれば昔の本でもできる。

計算機上のテキストデータを検索したり読んだりしたりするときに、物理的な本のメタファを忠実に利用することによって、マンマシンインタフェースを向上しようという試みもある[工藤 et al. 93]。これなどは「本の視覚化」と呼べなくもない。本を開いた状態がディスプレイに表示され、本の厚みの情報なども表示される。ユーザは本をぱらぱらめくったり、しおりを入れたり、ポストイットでメモ書きをしたりという感覚で、この「仮想本」とやり取りしながら表示された情報を眺めることができる。しかし、普段見えている物理的な「本」を計算機のディスプレイに表示しただけでは何となく物足りない。視覚化あるいは可視化というからには、普段は見えないものが見えてこなければならないと思うからである。

もう少し進んだ話に、PADと呼ばれるシステムがある[Perlin et al. 93]。これは、特に本だけをターゲットとしたシステムではないが、カレンダや書類や本などの情報の全体を、とにかく画面に全部表示する。本のような大量のデータの場合は当然縮小が行われて、最初の画面には、例えばその本の表紙しか表示されていない。それを「虫眼鏡」で順次拡大していくと、目次が見え、章が見え、最後にはテキストそのものが見えてくるというシステムである。とにかくすべての情報へのアクセスパスを点でもいいから画面の上に置いておくというのがこの設計の要点で、ハイパーテキスト風のいくつも分岐のある小説もトリー状に表示できるし、いくらでも詳細な部分へ入って行ける地図などという応用も考えているらしい。

色々の試みがなされているが、一冊の本程度の情報であれば、従来のテキストエディタで用が足りることが多いということだろうか？本ぐらいの単位では、ことさらに視覚化する必要はないのかもしれない。もちろん、その本に書いてある内容を分かりやすく視覚化するという全く別の話も可能性としては存在するが、そんなすごい話が実現するのは遠い将来のことだろう。

4.3.3 図書館の視覚化

普段は見えないものを見えるようにするのが視覚化／可視化であるとすると、その真価が発揮されるのは、やはりテキストデータベース全体あるいは図書館に蓄えられている文書全体の視覚化であろう。

図書館にある本の量を見てみると、表4.3-1のように規模によってかなり大きな差があるが、テラバ

イトのオーダであることに違いはない。こんな量のテキストを一目瞭然に俯瞰することは不可能である。しかし、上で述べたプログラムマップやPADシステムのように、なんらかの縮小あるいは省略を行って、それらのテキストのエッセンスだけを取り出すことはある程度できるのではないかというのが、テキストデータベース全体の視覚化の基本的アイデアである。

表4.3-1　図書館の保持するデータ量

各種図書館の蔵書数	データ量（概数）
武蔵野通研の蔵書（16万冊）	0.1 Tbytes
国立国会図書館（69年から現在まで；約120万冊）	1 Tbytes
ハーバード大学図書館（1045万冊；1990年当時）	10 Tbytes
米国国会図書館（Textのみ）	40 Tbytes

例えば、本の表紙を上に向けた状態で、広い床の上に2次元的にそれらの本を全部並べた状態を、はるか上空から眺めている場面を想像してみよう。これで一応は図書館の全蔵書を「俯瞰」することができる。もちろん並べ方には工夫が必要で、「上中下」というシリーズや雑誌の連続する号などは当然物理的に隣り合う位置に並べておかなければならないし、分野ごとにも固めて一つの領域に並べることになるだろう。突拍子もないアイデアのように思うかもしれないが、こうして2次元的な空間配置を固定しておけば、取りあえず、既知の本（つまり一度アクセスしたことのある本）へのアクセスはかなり容易になるだろう。さらに、ある分野の本を捜し探しまわるのも物理的な近さを頼りにすることで容易になる。

これを実現するに当たって、以下のようなことが問題点として挙げられる。

① どのような並べ方がユーザにとって分かりやすいものであるかを明らかにする必要がある。

② テラバイト・オーダの情報が存在するとすると、例えば1000 X 1000ドットのディスプレイでは、1ドットにメガバイト・オーダの情報を圧縮して表示しなければならない。このとき何をどのように圧縮するのか。

まだ簡単な解決法は見つかっていないようだが、これらの問題が解決すれば、このような大規模テキストデータベースのビジュアリゼーションは重要な技術になると思われる。

4.3.4　別の見方

上で述べたビジュアリゼーションは単なる情報表示ツールではなく、当然ドキュメントの世界（テッド・ネルソンの言うところのdocuverse）を動き回るためのナビゲーションツールでもなければな

らない。先ずはマウスと、PADで使われているような「虫眼鏡」で十分であるが、もっと気の利いた動き方をサポートする必要が生ずるかもしれない。画面表示を3次元にすることは、かえってナビゲーションを困難にすると思われるのであまり有効ではないだろう。

ナビゲーションとは若干異なる見方として、ここで紹介したビジュアリゼーションをフルテキストデータベースのgrep（正規表現によるパターンマッチ）の結果を表示するインタフェースと捉えることもできる。grepをかけて探した単語や言い回し、用語などを含むドキュメントが、例えば画面の中でハイライト表示（といってもごく僅かな光点だろうが）されるというわけである。大規模なドキュメントベースの検索用のプログラムにMG（Managing Gigabytes）というのがあるが[Witten et al. 94]、それのためのインタフェースと考えることもできる。

4.3.5 おわりに

ビジュアリゼーションというと図や絵を思い浮かべることが多いが、最も基本的な情報であるテキストもビジュアリゼーションの対象となり得ることを示した。さらに、大規模なテキストデータベースの全貌をディスプレイの一画面に表示し、検索やナビゲーションの助けとするというアイデアについて述べた。

【参考文献】

[Baecker & Marcus90] Ronald M. Baecker and Aaron Marcus: ``Human Factors and Typography for More Readable Programs'', ACM Press (1990).

[神林91]神林隆:プログラムを含む高品質ドキュメント作成システムLispWEB, 情報処理学会研究報告(91-SYM-62), Vol.91, No.105, 62-1, (1991.11).

[工藤 et al. 93]工藤正人、岡田謙一、松下温: 人間の空間情報処理能力を活用したユーザインタフェース: BookWindow,情報処理学会ヒューマンインタフェース研究会、情処研究報告Vol.93, No.35, 93-HI-48-2, (1993.5).

[Minsky94] Marvin Minsky: first person: Marvin Minsky, The Society of Mind, The Voyager Company, (1994).

[Perlin et al. 93] Ken Perlin and David Fox: Pad – An Alternative Approach to the Computer Interface, SIGGRAPH 93, pp.57–64, (1993).

[斉藤93]斉藤康己: ビジュアルインタフェース研究の落し穴, ビジュアルインタフェースの研究開発報告書の1.4節, 日本情報処理開発協会, (1994.3).

[Witten et al. 94] Ian H. Witten, Alistair Moffat and Timothy C. Bell: ``Managing Gigabytes – Compressing and Indexing Documents and Images'', Van Nostrand Reinhold, (1994).

5.　実用化動向

5. 実用化動向

5.1 PDAに関する動向

1992年1月、冬季コンシューマエレクトロニクスショーにおいて、そのキーノートスピーチでApple社の前会長John Sculley氏がPersonal Digital Assistant（以後、PDA）という新しいコンセプトを発表してから、多くのメーカーがこの新しいデバイスの開発に参加し、様々な種類のPDAが製品化されてきている。

一方で、従来からの、電卓から発展してきた電子手帳タイプの携帯型デバイスも歴然と存在し、何がPDAで何がそうでないのか、PDAというデバイスで我々は何を期待していいのか、また、PDAが今までのものと異なる道具として、人間の能力を十二分に発揮させる道具として活用されるためには、何が必要なのかなどの問に対して明確な答えを見つけにくいことも事実である。

本節では、このような問題意識に基づいて、この新しいデバイスが今後健全な発展を遂げるために、何を課題として捉えるべきなのかについて、現在の動向を基に考察する。

5.1.1 携帯端末の分類

携帯端末を広義に定義して、「携帯可能な情報処理デバイス」と位置付けると、様々な用途を目的とした様々なタイプのものが含まれる。これらをすべて含めてPDAと見る見方もあるが、ここでは、PDAを携帯端末の1つのクラスと見ることにより、議論の発散を避けることにする。

携帯端末は、その用途や形態から次の4つに分類できる。

(1) PDA

強力なCPUを備えることにより、機動性を保持しながら通信系の機能の拡充や文字認識を伴うペン入力などを備え、Personal Organizer（いわゆる電子手帳）を進化させた形になっている。

電子手帳と同様、バンドルアプリケーションは電源投入とともに使用可能な状態となり、一方、汎用なOSを備えることにより新たなアプリケーションをロードし実行することも可能である。

バンドルアプリケーションとしては、以下のようなものが挙げられる。

- 住所録
- カレンダ
- To-Doリスト
- ノートブック、スケッチパッド
- 財務管理
- ゲーム

・辞書

・計算器

・商用ネットワークへのアクセス用フロントエンド

・ファックス

・電子メール

(2) Keyboard-based Personal Organizer

いわゆる電子手帳で、価格が安いこと、小さいことがポイントである。CPUパワーの不足から文字認識を伴うペン入力は備えず、入力にはキーボードを利用する。

そのまま使えることを前提とするため、バンドルアプリケーションは豊富である。

バンドルアプリケーションとしては、以下のようなものが挙げられる。

・表計算

・アポイントメント管理

・電話帳

・ワープロ

・計算器

・汎用目的データベース

・世界時計、アラームクロック、ストップウォッチ

・ノートパッド

・名刺管理

・スケジュール管理

・定型文書作成支援

・ラベル作成支援

・カレンダ

・To-Doリスト

・アウトラインプロセッサ

・簡単なドローソフト

・データ転送

・電子メール

・通信ソフト

(3) Pen-based Portable

ペン入力を備えた点を除けば、ハードウェア、OSともに汎用のパソコンそのものである。汎用のパソコンであるため、PDAやPersonal Organizerのようなバンドルアプリケーションはない。通常のパソコンのようにアプリケーションソフトをインストールして利用する。

(4) Vertical-market Pen Tablet

業務目的の携帯機である。価格、性能はPen-Based Portableに近い。

アプリケーションはメーカーが注文に応じてカスタムアプリケーションを作成するものが多い。中には、定型フォーム入力を目的とし、バンドルされているフォームの作成支援ソフト等でユーザが独自に作成できるものもある。

バンドルアプリケーションとしては、以下のようなものが挙げられる。

- カスタムアプリケーション
- カスタムフォーム
- フォーム作成支援
- スクリーン作成支援
- データベースツール
- サイン認識
- 遠隔データ収集
- 通信ソフト

5.1.2 PDAの動向

(1) PDAの条件

5.1.1で示した狭義のPDAの特徴を一言でいうならば、次のように書けるだろう[Halfhill93]。

- 複数の異なる機能が実行可能な計算パワー（CPU、アプリケーション機能）
- 携帯可能でどこでも使える（サイズ、重さ）
- 長時間使用可能なバッテリーパワー（バッテリー、消費電力）
- 無線などを利用したコミュニケーション機能（通信機能）
- 多くの人が購入可能な価格で、誰もが使える（価格、ペンベースユーザインタフェース）

しかし、PDAが広くユーザに受け入れられるためには、これらの条件を単に満足させるだけではなく、ユーザから要求のあった機能を加え、ユーザインタフェースは簡素化し、主要な機能に必要な通信機能を統合することが求められる。誰もが使える、何でもできるを目指すより、PDA製品は、ある機能において、現在、市場にあるものよりも優っていることが求められる。やがてユーザはこの新しい製品について学び、これを受け入れるようになるであろう。新製品の常として、消費者にその製

品を使うことによって得られる利益を分からせることがまず必要である。このような観点から、PDAの市場におけるニーズは次のように捉えられる。

(a) 現在、市場にない（特にラップトップと比較して）機能を提供すること

現在、WindowsやMacintoshのノートブック、ラップトップ、パームトップを使っているユーザはそのプラットフォーム上で動くアプリケーションを頼りにしている。よってPDAがこれらのユーザに使われるためには、ノートブックが提供していない機能、より高いレベルの機能、そして携帯可能という特性を与えられなくてはならない。しかしノートブックはPDAが提供できるほぼすべての機能を組み込むことが可能であるので、結局は、携帯性と大きさ、重さなどがPDAにとっての差別化の重要なキーとなるであろう。これは、PDAの初期のユーザが、既にポータブルパソコンのユーザであるということを考えると、ますます重要である。もしPDAがノートブックに対して明確な利点を示せなければ、これらのユーザはPDAを使おうとはしないであろう。PDAがノートブックに対して携帯性において優るということを示すためのキーは、データ入力のための複数のオプション、小さくて軽いということ、そしてワイヤレスコミュニケーションである。この最後のワイヤレスコミュニケーションにおいてこそ、PDAは（ノートブックに対する）差別化を強調できると考えられる。

(b) コンピュータ使用経験のないユーザにも使いやすい環境を提供すること

多くの潜在的なPDAユーザは、PDAをコンピュータとしてではなく、システム手帳やコミュニケーションの道具として使う人々である。これらの人々は、現在の携帯電話、ページャ、メッセージサービスなどのユーザであり、今の業務をもっと効率的に行うためのものを必要としているセールスマン、エクゼクティブたちである。これらのユーザは、それを使うのに新たに勉強したり、メンテナンスをしたり、余計な時間をとられたりすることを嫌う人種であって、PDAを使うことで時間を節約できることを期待している。

その点で現在の手書き認識技術は、残念ながらまだ満足できるレベルに達していない。どうやってコマンドを入力するかという問題は、PDAのコンセプトの大きな課題である。

ある機能を使うのに要する時間というのも重要なファクタである。もしユーザが机の上に、PDAと電卓を両方持っていたら、計算する時にきっと電卓を使うであろう。それは、電卓の方が専用のボタンを備え、反応が速く、より使いやすいからである。計算を行う時には、単調な階層のない入力が使われるのに、PDAでは、電卓モードに入らなくてはならないのである。さらにサイズまたは押した感じがしないために、または手書き文字の認識のために、数字の入力が（PDAでは）非常に遅くなってしまう。消費者は、既に電卓というものに慣れ親しんでいる。PDAについては、使いたい機能にプルダウンメニューなどを使わずにより速くアクセスでき、データ入力を簡単に速く行えるようなユー

ザインタフェースが必要である。入力の最終ゴールは音声入力であるが、その技術は未だに未熟な領域をでない。

(2) PDAの重要要素技術

以上の考察を基に、PDAにとって重要となる要素技術とその動向を以下に示す。

(a) 小スクリーンユーザインタフェース

PDAは、携帯性が重要であるため、必然的にディスプレイが小さくなる。小さいディスプレイで、ユーザにとって分かりやすく、しかも、希望する機能に素早く到達できるユーザインタフェースを構築するためには、それなりの工夫が必要となる。

ここでは、小スクリーンユーザインタフェースにおける工夫を「知的インタフェース」、「実世界メタファ」の2つのキーワードで説明する。

(i) 知的インタフェース

PDAもパソコンも基本的にはGUIを利用するが、この両者の大きな違いは、マルチウィンドウが表示可能か不可能かという点である。小スクリーンユーザインタフェースでは、マルチウィンドウを表示することが困難であるため、ツールの変更が即画面の切り替えとなる。人間は、通常複数のツールを使うことによって作業を行う。パソコンのGUIでは、マルチウィンドウによってツールの切り替えをウィンドウの切り替えで行え、かつ複数のウィンドウを同時に表示可能であるため、ツール切り替えのためのメンタルロードが少なくてすむ。しかし、PDAでは画面が小さいため、1つのツールの表示のみで画面を占有せざるを得ない。

「知的インタフェース」はこの問題を解決する。「知的」という言葉から、推論や学習など人工知能の手法によるものを想像するかも知れないが、現状では「知的」とは、「知的に動作するようにデザインされている」という意味で理解するほうが正確である。例えば、AppleのNewtonで実現されているintelligent assistant機能は、このための最も特徴的な機能である。会議中のメモに"Call John"という覚書が含まれていれば、これをペンで選択してIntelligent Assistantアイコンに触れるだけで、Johnという名前のアドレス帳カードを表示し、正しいJohnを選択するだけで、スピーカからダイヤルしてくれる[Thompson93]。

このような知的インタフェースの技術は、今後、人工知能の技術を取り入れ、ユーザごとのツールの連携の仕方を学習したり、自動的にマクロ操作を学習していくような個人適応機能を有したユーザインタフェースへと発展していくだろう。

(ii) 実世界メタファ

一般に、新しい種類のツールは、その機能が豊富であればあるほど使いこなすために費やす時間が多く必要となる。多くのPDAユーザは、PDAを使うことで時間を節約できることを期待しているセールスマン、エクゼクティブたちであり、一般に、使うための勉強やメンテナンスに余計な時間をとられることを嫌う。

昨年度の報告書では、次世代のビジュアルインタフェースとして、実世界のメタファを利用したユーザインタフェースの試みがいくつか紹介された[JIPDEC94]。これらの研究は、パソコンやワークステーションの処理能力が向上し、ユーザインタフェースのためにふんだんに利用できる時代を想定して、仮想現実感や実写ビデオイメージを利用したリアリティの高いものであったが、PDAにおいても、実世界メタファを利用するというコンセプトを採用しているものが出てきている。

図5.1-1、 図5.1-2、 図5.1-3は、General Magic社のMagic Capによるユーザインタフェースの一例である。Magic Capは、Desk、 Hallway、 Downtownという日常生活の環境をメタファとして採用することにより、初心者にもすぐ使えるユーザインタフェースを持ったPDAのためのOSである。Deskには、Name card、 Notebook、 DatebookなどのPIM（Personal Information Manager）が置かれており、Hallwayに出るとそこには、Game room、 Library room、 Store roomなどの別の仕事をするためのソフトウェアが別の部屋として配置されている（Deskも1つの部屋である）。Downtownに出れば、そこにはオンラインサービスへアクセスするためのクライアントソフトウェアが建物として表示されており、ユーザはサービスを受けるために各建物に入るように操作を行う。

実世界メタファの実用化はPDAに限ったことではない。Apple社が1994年より始めたネットワークサービスであるeWorldもそのメイン画面において、街のイメージをメタファとして採用しているし、1995年に発表されたMicrosoftの新しいGUIであるBOBも、部屋のイメージをメタファとして用いたものとなっている。パソコンやPDAのユーザの裾野が広がっていくにつれて、このように直感的に分かりやすいユーザインタフェースが望まれてくる。

(b) 情報アクセス

PDAによってユーザの生産性を向上させるためには、通信機能を強化するとともに、様々なオンラインサービスへのアクセスを容易に素早くできることが重要となる。このためには、通信を行う場所を限定しないための無線通信機能のサポートも重要であるが、これに加え、オンラインサービスそのものが便利であることも重要である。

オンラインサービスは、次に示すような3つの世代の発展として捉えることができる[Reinhardt94]。

(i) 第1世代（Character-based、 Command line oriented）

ホストコンピュータから送られてくるキャラクタデータをターミナルエミュレータによって表示す

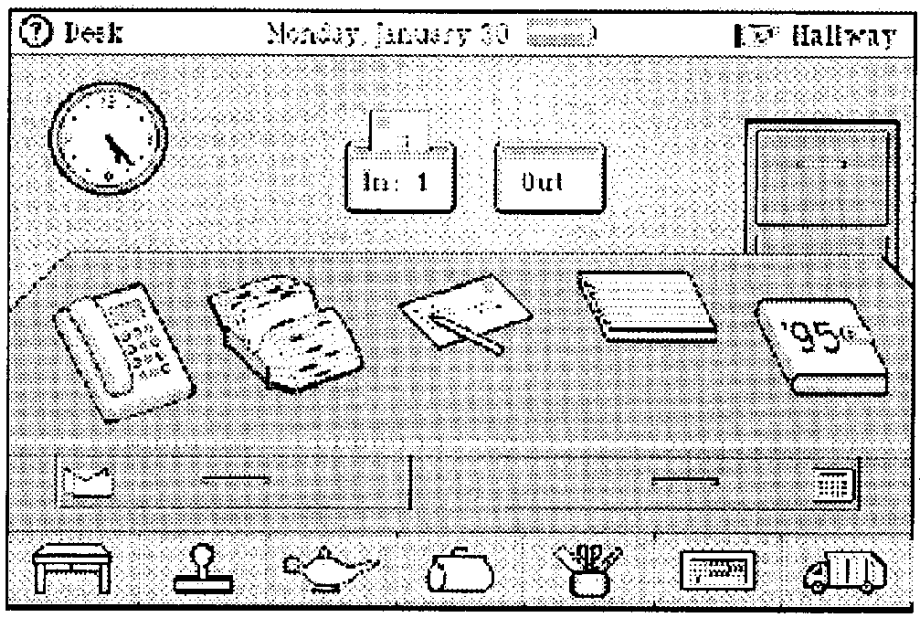


図5.1-1　Magic Capのユーザインタフェース例　(deskシーン)

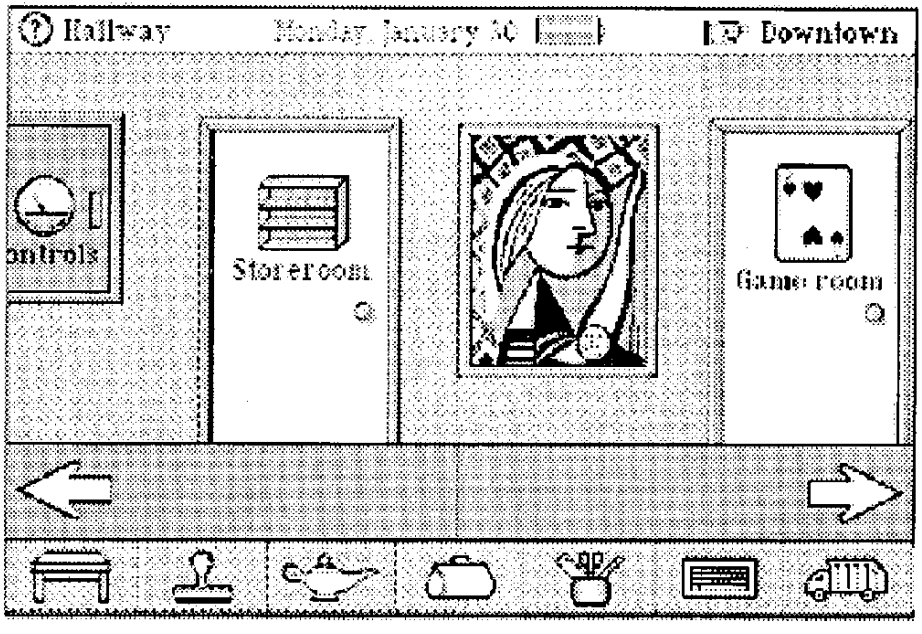


図5.1-2　Magic Capのユーザインタフェース例　(hallwayシーン)

る形態である。ホストコンピュータとのセッションは、コマンドベースで行われる。キャラクタによって簡単なメニューがホストから表示される場合もあるが、コマンド解釈やプロンプトなどのユーザインタフェースはすべてホスト側から表示される。

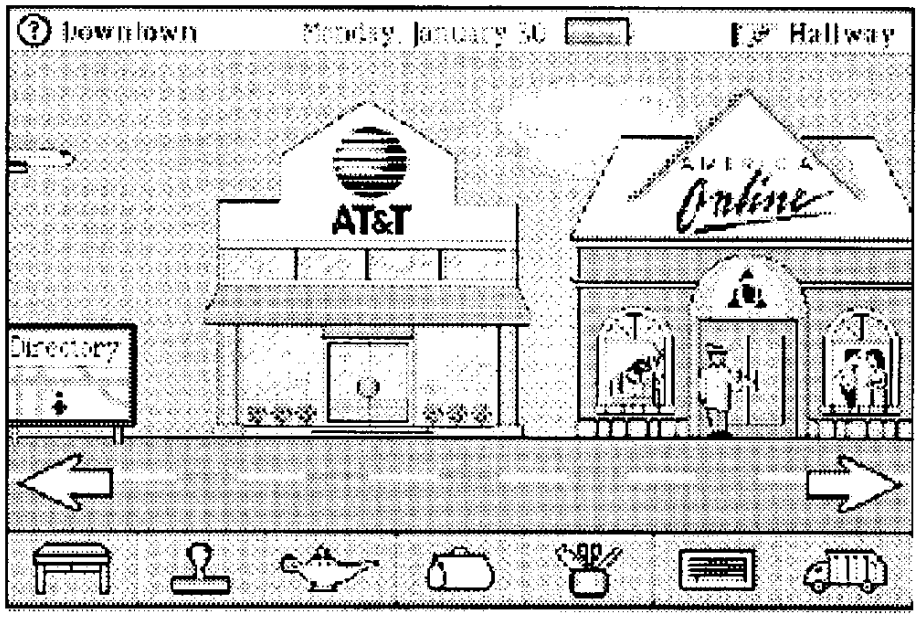


図5.1-3　Magic Capのユーザインタフェース例 （downtownシーン）

例えば、初期のInternet、 NiftyServe、 CIMを用いないCompuServeなどである（図5.1-4)。

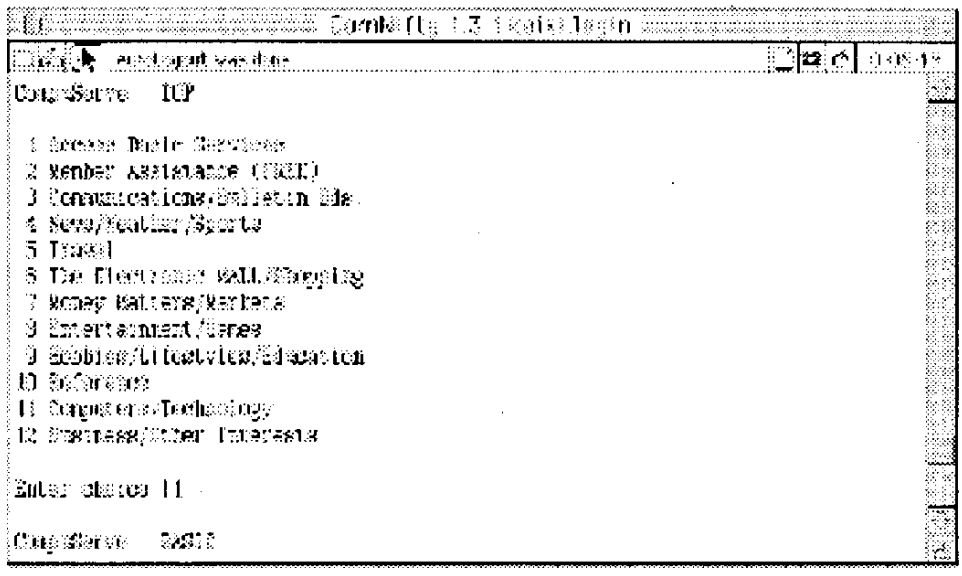


図5.1-4　第1世代の画面例 （CompuServe）

(ii)　第2世代（Graphics-based)

ターミナルとしてパソコンを利用し、所定のクライアントソフトウェアをパソコン側にインストールすることにより、パソコン側で分かりやすいGUIを表示するようにしたものである。ホストとパソ

コン間は、ネットワークごとの独自プロトコルで通信するが、GUIはすべてパソコン側が有し、ホストからパソコンへはGUIの表示要求が送信されるため、通信のオーバーヘッドは少ない。

例えば、Prodigy、 America Online、 eWorld、 Ziff-Davis Interchange、 Mosaic/WWWなどである(図5.1-5)。

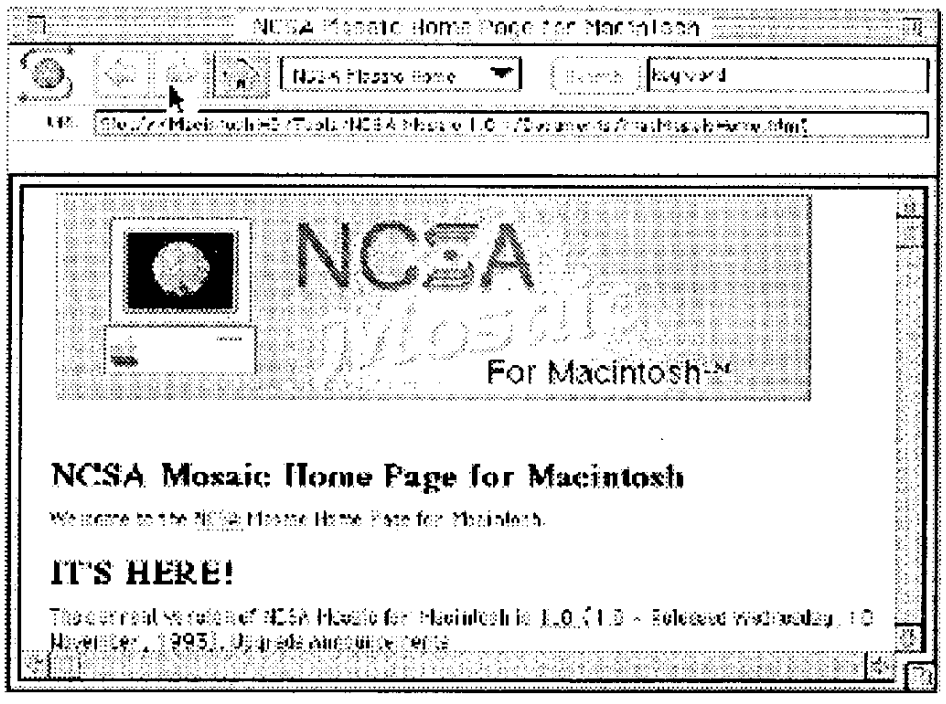


図5.1-5　第2世代の画面例 （NCSA Mosaic）

(iii)　第3世代（Agent-based）

ターミナルとしてパソコンやPDAが利用され、GUIがターミナル側に持たれる点は第2世代と同じであるが、情報の検索などの処理がエージェントとしてホスト（ネットワーク）側に送り込まれ、接続を切り離した後でもネットワーク内で処理が行われる。すなわち、第2世代のようにセッションベースで処理が行われるのではなく、すべての処理はストア&フォワードベースで行われる。このため、無線通信のような時間当たりの通信料が高い通信手段を用いている場合には、通信時間を短くできるため適している。

例えば、AT&T PersonaLink、 IBM Intelligent Communicationsなどである（図5.1-6)。

パソコンの性能が向上し価格が低下するとともに、パソコン側でGUIを表示する第2世代の形態が取られるようになってきた。上記の3つの世代でいえば、米国では第2世代が花盛りであり、日本ではようやく第2世代の入り口に来ているといった状況である。今後、PDAが広く普及し、無線通信が多くのユーザに利用されるようになれば、第3世代の形態が重要となってくる。

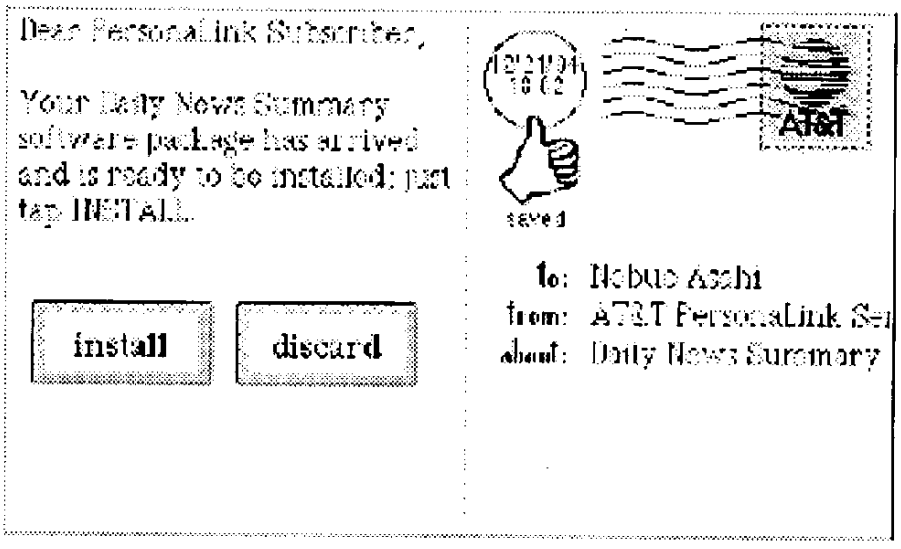


図5.1-6　第3世代の画面例　(AT&T PersonaLink)

5.1.3　おわりに

PDAにとって重要となる要素技術とその動向について述べた。

PDAを利用して、幅広いユーザが生産性を向上させるとともに、豊かな生活を送ることができるためには、情報提供者、ネットワークオペレータ、メーカーが協力し、コンピュータ技術、通信技術、ユーザインタフェース技術などを総合的に考慮しつつ環境を整えていく必要がある。

【参考文献】

[Halfhill93] Tom R. Halfhill: PDAs Arrive But Aren't Quite Here Yet: Byte October (1993).

[JIPDEC94] 日本情報処理開発協会: ビジュアルインタフェースの研究開発報告書 (1994).

[Reinhardt94] Andy Reinhardt: The Network with Smarts: Byte October (1994).

[Thompson93] Tom Thompson, et.al.: Ease of Use Is Relative: Byte October (1993).

5.2 今後の動向

5.2.1 はじめに

ここ数年、コンピュータの世界でインタフェースといえばディスプレイやキーボード、マウスといった入出力機器の世界を中心に語られてきた。つまりインタフェースとは、コンピュータの中の仮想世界（機能）と実世界（ユーザ）の境界面であるとして、デザインされてきた。しかし、今やプロセッサの小型高性能化、記憶媒体の高密度化、通信技術の進歩等により、コンピュータが現実世界の中に不可視の状態に溶け込んだり、コンピュータの内部の仮想世界の向こうに別の現実世界がつながっていたりする状況が日常的になりつつある。すなわち、現在は21世紀にかけて仮想世界と実世界を明確に分離する従来のパラダイムが大きく変化しようとしている時期であり、Ubiquitous Computing[Weiser91]、Augmented Reality[ACM93]、Virtual Reality[Fisher86]、といった技術に注目が集まるのもこのパラダイムシフトを象徴しているといえよう。

ヒューマンインタフェース技術の目的自体にも変化が生じている。従来のインタフェース技術は、人と機械のインタラクションを円滑に行わせることを目的とするものであった。しかしながら、現代では人と人の間のインタラクションを強化する、あるいは人と社会の関わり方を支援するといったことに研究の重点が移りつつあるように感じられる。CSCWの研究[Ishii91]やVirtual Communityの研究[White94]などがその好例である。ここでは、これらヒューマンインタフェース技術における新しい動向について概観する。

5.2.2 Ubiquitous Computing

コンピュータがコンピュータとして存在を意識される間は未成熟段階であり、その存在が実世界に織り込まれて不可視になってこそ深い技術として社会に受け入れられたといえる、とするUbiquitous Computingの考え方は、マイクロプロセッサの急速な進歩と相まって大きなインパクトをインタフェース研究者に与えた。基本的な考え方自体はTRONにおける電能都市等で既に提唱されていたものであるが、仮想世界の構築に走りすぎたコンピュータ技術に対する警鐘としての視点に新鮮味があった。

現実にUbiuquitous Computingは次々と応用を開けつつある。興味深いのはUbiquitous Computingなどというキーワードはもちろん、コンピュータサイエンスやヒューマンインタフェース技術などという意識は一切なしにそれが進められている点である。このことはUbiquitous Computingの経済効果の大きさの証明にほかならない。実際、大型トラックのタイヤにマイクロプロセッサを埋め込んで摩耗状態から交換時期を警告したり、マラソンランナーの靴紐にマイクロプロセッサをつけて正しいコースを走ってきたかをチェックしたり、といった意外ともいえる応用は枚挙に暇がない。デバイス面からは非接触読み取り技術の進歩もUbiquitous Computingを支えている。スーパマーケットの商品に非接触読み取り可能な識別タグ（LSI）を刷り込んでおいて、レジで篭に入った全商品の種類と価格

を一瞬で読み取るといった試みもなされている。

こういった技術は、これまで築かれてきたインタフェース技術を無意味にしてしまうインパクトを有する。駅の切符の自動販売機を例にとる。切符の自動販売機は、従来からインタフェースが種々検討されてきたものの一つである。実際、新型の券売機採用の前には、その機械を試験的に置いてビデオカメラを付けユーザの使用状況を記録してプロトコル解析を試みている。音声で切符を買える券売機の試用が行われたのも記憶に新しいところである。しかし旅客が非接触読み取り可能なIDの付いたカードを持ち、改札の出入りでそれを読んでその客の口座から引き落とすようにすれば、切符を買うというインタラクション自体が消滅する。したがって、そのインタラクションを円滑化するためのインタフェース技術も不要となる。

この意味でUbiquitous Computingは、「非本質的なマンマシンインタラクションの極小化」を実現する技術とみなせる。上の例でいえば、本来は「電車に乗って移動する」のが目的であり、そのために切符を買ったり、駅員にそれを見せたり、回収してもらったりというのは二次的あるいは本来的には不要なインタラクションであり、Ubiquitous Computingにより消滅してしまう。インタフェース研究者が発案した、ヒューマンインタフェース技術者の失業を招く技術との皮肉な見方もできないことはない。無論ここで注意すべきは、「不要なインタラクション」なるものの判断には十分慎重な社会的、文化的検討が必要である、という事実である。切符を買うという、本来の目的から見れば冗長な行動には、旅客に自分の経済的能力や目的地への距離感覚、利用しようとしている路線が正しく自分の目的地へ行くのか等、多くのフィルタリングの機会を与えている。近い将来実現されるであろう非接触読み取り型の電子パスの普及時もこれらに対する配慮が必須であり、場合によっては新たなインタラクションを設定する必要が生ずることも十分に考えられる。このようにUbiquitous Computingは、以下のことをヒューマンインタフェースを研究開発する人々に要求しているのではなかろうか。

① 「ユーザと機械のインタフェース」という表層的考えからの脱却。

② 部分的なインタラクションでなくシステム全体として考える。

③ ユーザの行動の真の目的と意味を社会的、文化的視点から考察する。

必要とされるスキルも、ウィンドウシステムや各種認識技術あるいはインタフェースガイドラインといったものに対する知識から、マイクロプロセッサや非接触読み取り可能なメモリといったデバイス技術から社会システムに至る垂直統合能力が重要となろう。

5.2.3 マルチモーダル

マルチモーダルはUbiquitous Computingに比べると、従来のインタラクションの考え方の自然な発展という意味で、インタフェース研究の枠組みの中ではより扱いやすく感じられる。しかしながら、Ubiquitous Computingのように経済的効果が明確でないため、実験室でのトイシステムに終わりがち

なのもまた真実である。比較的成功したと思われるのはペン入力であるが、これもポインティングデバイスとして機能するからであり、手書き文字入力が市民権を得たとの判断には時期尚早といえる。音声入力に関していえば状況はさらに深刻であり、長い研究にも関わらず日常的なインタフェースとしては実用になっていない。「人間並みの音声認識能力」という要請はほとんど「人間並みの知能」を要求しているのと大差ないことを考えれば、このような認識精度が近い将来に達成されるとは考えにくい。

マルチモーダルには未来がないと言い切れればむしろ幸いなのであるが、不幸にして実験室のトイシステムは実用にならないと感じつつもとても魅力的な例が多い。いま音声入力を含むマルチモーダルインタフェース技術にとって重要なことは、多くのユーザからフィードバックをもらいうる応用を探し出すことであるということについては異論は少ないと思われる。音声、ジェスチャなどの認識をベースとしたマルチモーダルインタフェースはその本質として、以下の問題点を有する。

① ユーザのトレーニングを要する。

② 認識誤りの可能性が常にある。

したがって応用としては、上記の問題点が不自然でなく、願わくば楽しみの一部であるようなものが望ましい。一例としては、最近電子手帳の応用ソフトの一つとして子供の間で人気の電子ペットが挙げられる。犬などの電子ペットを育てるシミュレーションソフトを考えると、飼い主が音声でいろいろ話しかけると反応するようなシステムは現状の技術でも十分に可能と考えられる。話す内容をペットに教えることが、システム内部から見ればユーザ自身の音声を学習することになるわけで、トレーニングに必要な忍耐を、育てる楽しみの形でユーザに自然に課することが可能かもしれない。また認識誤りもユーザが悪いのではなく、電子ペットの能力不足あるいは機嫌の悪さからの不服従に見せることができる。姑息な手法との批判もあろうが、一つにはこういった応用の積み重ねで「音声認識技術に対する認識」を高めていくことが重要なように思われる。

もう一つの可能性は、限られた音声認識（例えば日時のみ認識）と録音機能（音声メモ）を使ったPDAを突破口とすることであろう。米国では既にこの種の機器が200ドル以下で販売されており、実ユーザも存在するようである。

5.2.4 エージェント

エージェント指向は人によって定義が千差万別であるが、ここでは「人間の機能特性のうちの一つないしいくつかを模擬したソフトウェア（エージェント）を単位に、システムを構築しようとする考え方」という範囲の広い定義を用いることにする。従来は最も人間らしい部分である知能を模擬したものがエージェントと考えられる傾向があったが、最近流行のTelescriptでのエージェントはむしろマイグレーションの機能を強調しており、知性とは縁遠い存在である。すなわちTelescriptエージェ

ントは人間の頭ではなく、移動するための「足」を模擬したものと考えられ、この意味で立派に前述のエージェントの定義にフィットするものである。また一般にインタフェースエージェントと呼ばれるものは、キーボードマクロ、すなわち「手」の機能に人間の顔や表情あるいは姿形をかぶせた程度のものとなっている。

足エージェントや手エージェントは広く利用されるようになろうが、「学習」という機能を模擬した頭エージェントへの道は遠い。頭エージェントの問題点は前述の音声入力の問題と酷似しており、コンピュータの中では当面電子ペット的位置付けを脱することは困難と予想される。繰り返し操作を検出して学習するエージェントシステムであるEagerが、エージェントの容姿として猫を選んだのも理由のないことではない。固く考えずに、一方通行で無味乾燥になりがちなコンピュータのインタフェースに潤いを与え、ユーザ固有の「賢い猫」を育てる楽しみを与える技術として積極的に評価していくと思わぬ発展が期待できるかもしれない。

5.2.5 CSCW

人類の昔からの夢の一つに、どこにでも瞬時に移動できるテレポート能力がある。物理的に移動するのは説明が難しいので、精神が移動するというアイデアもSF小説ではよく使われている。このアイデアはさらに、精神がそれ単体として存在しうるとするものと、あまり気持ちはよくないが何か(最近の某人気アニメ番組ではこれを「器」と呼んでいた)に乗り移るタイプとに分類される。器に必要な基本条件は、それが周囲を観察する能力を持つことである。

これから数年の間に、ほとんどのノートブックタイプのコンピュータは、TV電話として機能させるためのカメラを備えるようになると予測される。言い替えると、ノートパソコンは手足こそないものの、ネットワークにつながることによって人が入るための器の条件を満たすことになる。したがって、いわゆるTV会議をする際も例えば3人が実際に会議室にいて、あとの4人はいろいろな場所にいるとした場合、会議室に4台のノートパソコンを置いて、それぞれを4人のための「器」にすれば高価なVirtual Reality技術を用いなくても臨場感に富んだミーティングが行えよう。もちろんノートパソコンのディスプレイにはそれぞれの人の姿が映っていたり、その参加者の発言がスピーカから流れ、時には説明したい文書も表示されていたりするであろう。またカメラは器に入っている人が自由にコントロールでき、会議室の人々の様子を観察するのを助けているに違いない。

遠くにいる人のための「器としてのコンピュータ」という視点は、Virtual Realityにおけるテレイグジステンスの概念に近いが、インタフェースはもとより社会的にもこれまで経験のない多くの問題を生ずると考えられる。既に米国ではあまりに多くの人々がビデオカメラを常時持ち歩いている、すなわちUbiquitous Videoの状態にあり、このことが様々な問題を引き起こしていると報告されている。Ubiquitousにネットワークに接続され、カメラを持ったコンピュータが存在する世界は明るく考える

とどこにでも好きなところに行ける世界であり、暗く考えると誰がどこにいて何を見聞きしているか分からない状態といえる。インタフェース的には、例えば「器に入っているときには必ずディスプレイには自分の顔と名前が表示されている」といった簡単なガイドラインでも不正な使用を妨げるのにかなり効果がある。Ubiquitous Computing時代の課題の一つはセキュリティやプライバシ保護であるというのはよく指摘される点であるが、器が満たすべき性質についても検討が必要であると思われる。

【参考文献】

[ACM93] Back to Real World, Communications of the ACM, Vol.36, No.7, July, 1993.

[Fisher86] Fisher, S.S., McGreey, M.., Humpshries, J. & W..Robinett, Virtual Environment Display System, Proceedings of the 1986 Workshop on Interactive 3-D Graphics, Chapel Hill, North Carolina, 1986.

[Ishii91] Hiroshi Ishii & Naomi Miyake, Toward an Open Shared Workspace: Computer and Vvideo Fusion Approach of Teamworkstation, Communication of the ACM, 34, 12, December 1991, pp.37-50.

[Weiser91] Mark Weiser, The Computer for the 21st Century, Scientific American, September, 1991, pp.66-75.

[White94] Sean White, Online Communities on the WWW, Stanford Computer Forum WWW Workshop - September 20-21, 1994.

平成 7 年 3 月 発 行

発行所　財団法人　日本情報処理開発協会
東京都港区芝公園 3 丁目 5 番 8 号
機 械 振 興 会 館 内
TEL　(03)3432-9384

印刷所　有限会社　盛光印刷所
東京都千代田区飯田橋 4 丁目 6 番 3 号
宝第 3 ビル
TEL　(03)3264-1851

06－R003