ヒューマンインタフェース 42−11
(1992. 5. 12)

第四世代言語とグラフィクユーザインタフェース

上田 穣（日本ウェーブフロント）・藤田 喜彦（ＣＳＫ）・
白井 靖人（静岡大学・教育学部）・國井 利泰（東京大学・理学部）

Abstract: To achieve the best cost performance, superimpose a graphical user inteface over the fourth generation language which is originally constructed based on ASCII terminal use.

Keywords: 4GL (fourth generation language), DBMS, DBML, GUI (graphical user interface), visual programming

1 初めに

日本人の若者を徒弟制度で時間をかけて育てられる時代は終った。労働力不足でかつ離職率が高い今日においては、国籍年令経験を問わず新規採用した者を、いかに短い教育期間で、即戦力に出来るかが、企業存続の条件となりつつある。パッケージ・ソフトウエアも内部機能が有るのは当たり前で、使い易さが売れる決め手になりつつある。

2 第四世代言語（４ＧＬ）の現状

存在するソフトウエアの７０％は、事務計算用である。１９６０年代、汎用大型機しか無い時代に、ＣＯＢＯＬとアセンブラでがむしゃらにシステム開発が行なわれた。その内、個々に開発されたシステム間に共通点が多いことが分かってきて、生産性向上の手段として、まず、データベースシステムの提唱があり、第四世代言語、そしてＣＡＳＥが順次登場した。

第四世代言語は、高級言語であれば、特定ハードウエア、ＯＳにより異なるのが通常である
(1) データベース記述に伴うファイル、フィールド・データなどの定義
(2) ワーク領域の確保
(3) 入出力に関する諸指定
について考慮する必要が無いよう構築されている。その結果としてユーザは
(4) 処理のロジック記述
だけを考えればよい事を建前とする。故に、第三世代言語に比べ生産性が高いとされる。
これを可能にしたのは、アルゴリスムが集大成整理されてきて、「表形式データで事務処理をする」と、適用業務の内容が明確になったからである。

１９７０年代半ば、ＤＢＭＳ，４ＧＬ，ＣＡＳＥそれぞれの得意分野は別々であった。しかし１５年後の今日、大型汎用機上でのＤＢＭＳのＤＢＭＬ機能が大いに発展したため、結果として、大型汎用機上での４ＧＬはその機能を漸次浸食されてきている。

Graphical User Interface for the Fourth Generation Language
UEDA Minoru (Wavefront Japan Ltd.), **FUJITA** Yoshihiko (CSK Corp.), **SHIRAI** Yasuto (Shizuoka University), **KUNII** Tosiyasu (University of Tokyo)

近年ダウンサイジングが進行して、ＤＢＭＳも４ＧＬも、ワークステイションやパソコンに移植されつつある。この状況で両者を比較すると、同一プラットフォームでの４ＧＬの機能に比べ、ＤＢＭＳのＤＢＭＬは、

(1) 複数ファイルで、キイによるデータ検索
(2) 統計計算を含む計算処理

を主要な機能とし、同一ファイル内での

(3) フィールド同士の単項式計算

を実行できる程度である。
これに対し、４ＧＬは、条件判定文と繰り返し文のより肌理の細かい使用が可能で、その特徴を用い、個別命令を数十連結したマクロ処理ができることで優位性を保っていると言える。
現行の４ＧＬは、汎用大型に接続されたＡＳＣＩＩ端末の使用を前提として開発されたものが大半なので、ダウンサイジングにより、全くの一般人の使用が急増するにつれ、ユーザーインターフェイスの改良が求められている。

3　ユーザーインターフェイス研究の現状

アイコンとマウス、マルチウインドウを主要素とする各種グラフィック・ユーザー・インターフェイス（ＧＵＩ）ツールの急速な発展につれ、これを、ＣＯＢＯＬ，Ｃなどで行なわれている現在のプログラミングを初心者でも使えるようにビジュアル化する研究が進展してきた。
第三世代、第四世代言語を問わず、ＧＵＩツールを用いて、文字データの入力以外は、キイボード入力を不要にするということで、研究方向が一致している。
個別処理を示す関数アイコンをマウスで選択し、画面上の適切な場所まで移動して配置する。１つのアイコンに、それぞれ入力、出力が付属している。一種の絵合わせパズル的感覚で、ユーザーが複数のアイコンを組み合わせる。システムが、それを自動的に解釈して、文字表現として評価することで、１つの適用業務が処理される。制御機能として、繰り返しループが必要な場合は、図同士を入れ子構造にして対処する場合が多い。
ウインドウは、多目的ウインドウと編集用ウインドウの最低二種を使い分けている。
１９８０年代に多くの研究が行なわれ、プロトタイプがいくつか発表された。Ｃ言語のような汎用性を、アイコンの積み木を組み上げるようにして実現しようとするのであるが、大半が教育用で、実務に広く使用されているものは皆無というのが現状である。

4　新しい方針

「よいユーザインターフェイスとは何か」の原点に戻り、成功しているＧＵＩの実例を検討する事により、望ましい方針を求めていくことにする。よいグラフィックス・ユーザー・インターフェイスは、次の相矛盾する条件を同時に満足しなければならない。

(1) 初心者の学習時間が短い
　用語が馴染み易く、命令体系が簡潔かつ論的一貫性を持ち、入力エラーが少ない。
　画面上に表示された視覚的なオブジェクトを操作することで実現する。
(2) 熟練したユーザーが、より早く実行できるように、バッチ処理的な近道が利用可能。

上記(1)(2)を両立させるために、何が何でも全てをマウスとアイコンで処理するのでは無く、洗練されたデザインにより、視線と手の移動距離を最小にするというコスト・パフォーマンスの高さを第一目的にするべきである。

成功しているＧＵＩの例は、ＣＡＤとお絵描きソフトであるが、両者とも適用業務の内容が明確であり、まずグラフィックス・サブルーチンとして順次開発蓄積されたものが、サブルーチン・パッケージとなった。次にマクロ言語が開発され、その後初めてアイコンを用いるＧＵＩが開発されたのである。

ＣＡＤやＣＧを専門としてきた者にとっては、ＧＵＩというのは、１９７０年代初めに開発された２次元グラフィックス機能の一部でしか無い。しかしアスキー端末でＣＯＢＯＬで事務計算ソフトウエアの開発をしてきた人々の一部の目には、グラフィックス機能が珍奇なものに映り、その面白さに溺れる傾向が見受けられる。手段が目的になっているのである。

5　研究用プロトタイプの仕様

本研究では、第四世代言語としてＳＴＹＬＥ*を用いた。その体系は、３０余のコマンドと２０余のスイッチの組合せの約８０の命令群からなる。（使用頻度の高いのは、その内約３０命令である）　基本的には、ＢＡＳＩＣ言語と同じインタープリター方式であるが、バッチ様式で使用できる。独自のデータベースを持つが、他のＤＢＭＳとのネットワークが可能である。またマルチ・トランザクション可能である。

* ㈱コンピュータソフトの登録商標

同時に呼び出せるデータファイルは２つで、ループは２階層である。この機能で全ての適用業務に対処出来ることが、経験的に分かっている。このＡＳＣＩＩ端末使用を前提として開発された第四世代言語に、ＩＢＭ　ＡＩＸプラットフォーム上のＸ－ｗｉｎｄｏｗ／Ｍｏｔｉｆを用たＧＵＩを構築した。

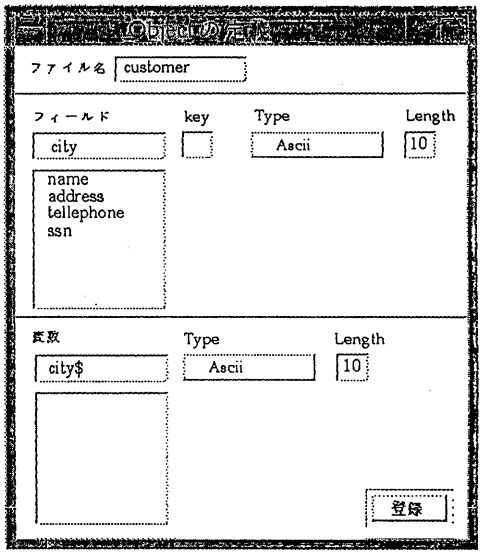
現在ＳＴＹＬＥは、数千人の従業員を持つ組織で管理業務に使用されているが、ＢＡＳＩＣでプログラミング開発するようにして使用されている。その作業過程を以下に述べる３段階のＧＵＩを、いわば被せる形で変更する。

システムを起動すると、画面上部にメニューバーが表示される。最左のメニューを選択するとユーザーが何をしたいのかプルダウンメニューとして選択できる。例えば、データファイルの定義、入力、キイによる検索などの１つを選択すると、右側のメニューバーの下にプルダウンメニューが表示される。８０余の命令中、この処理に必要とされる十数ケの命令群だけが示される。

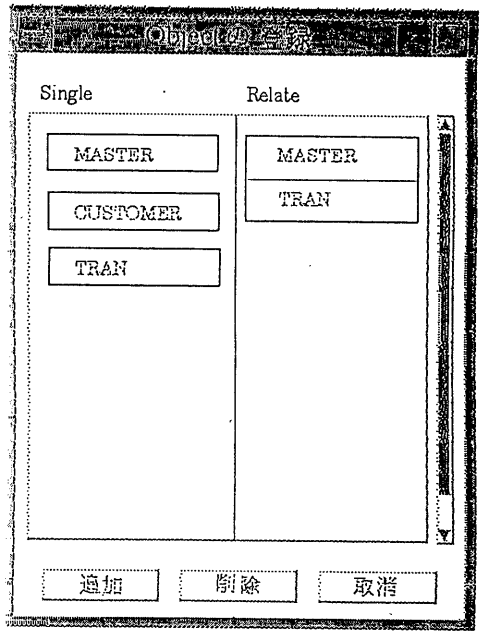
第一段階　アルゴリズムのインスタンス化
基本方針は、データのオブジェクト指向化ではなく、４ＧＬの命令が表すアルゴリズムのオブジェクト化である。すなわち、個々の命令をアルゴリズムをクラスと考え、それに具体的なファイル名、フィールド名、変数名、データ値、行と列の指定を与えて、アルゴリズムをインスタンス化する。８０余の命令は機能として、下記の９種類になる。

(1)	システム	password, exit, HEL等
(2)	制御	IF, LOOP, DO SUBROUTINE等
(3)	ファイル	新規、書込、呼出、削除等 CREATE, APPEND, LOAD, PURGE
(4)	フィールド	新規、書込、呼出、削除等 CREATE, DEFINE,　　ERASE
	キイ指定	INDEX
(5)	変数	新規、書込、呼出、削除等 SET, DEF　　RERATE, CLEAR
(6)	データ	新規、書込、呼出、削除等 INPUT, APPEND, SORT, RETRIEVE
(7)	レイアウト	コメント文、変数とデータの 行列指定、左中右詰め指定 TAB(列、行)
(8)	入力（画面）	INPUT
(9)	出力（画面、印刷）	PRINT

このインスタンス化されたアルゴリズムは、単独で単一の処理をする。
図－１で(4)のフィールドの定義をし、図－２で作成されたファイルの一覧を表示。



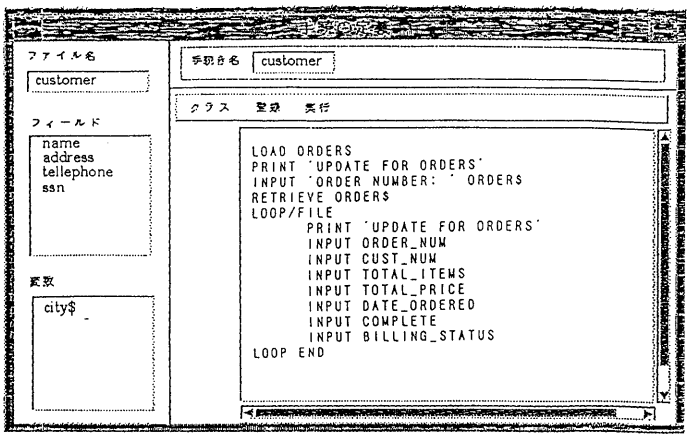
図－１　インスタンス生成



図－２　作成されたデータファイル

第二段階　プロシージャー
上記のインスタンスを数十並べて、バッチ様式で処理を行なえるようにしたものをプロジージャーと呼ぶ。（図－３はデータ入力の例。既に定義されたフィールド群が左にリスト・ウイジェットとして表示され、その中から必要な項目を選択する。）

１つのプロシージャーの処理内容を決め、必要なアルゴリズムのインスタンスを順次作成する最後にmake file を実行すれば、インスタンスの作成順序に無関係に、システムが完全なプロシージャーを作成してくれる。ユーザーは、個々の命令をどう並べるか心配する必要が無いという意味で非手続き型だと言える。

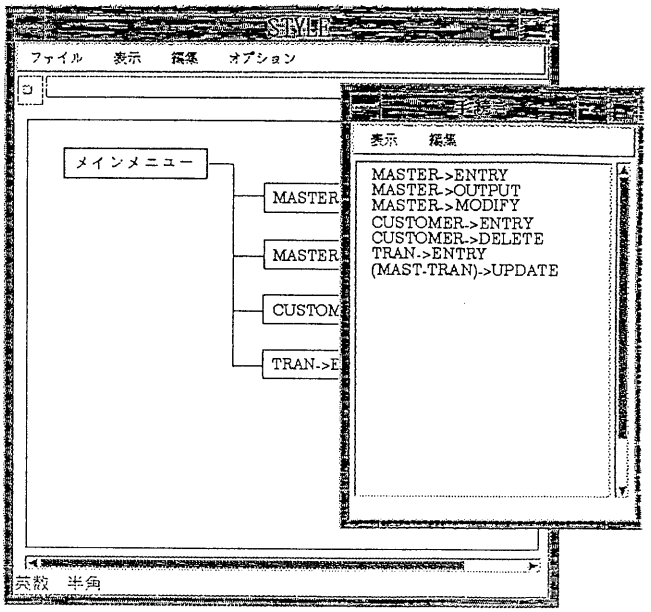


図－３　プロシージャの事例

ＳＴＹＬＥの場合、データファイルを、同時２つメモリ上に呼び出して、順次処理する。主たる処理の内容は下の３種である。

(1)　入出力に関するもの
　　データファイルの呼び出し
　　出力用レイアウトの指定
　　実際の入出力
(2)　同一ファイル内のデータ処理
　　欲しいデータの検索方法
(3)　複数ファイル間のデータ処理
　　キイ同士のマッチング

ここで繰り返し命令LOOPが使用されるが、ファイルを最初からEOF まで読むか、開始と終了レコードを指定して、LOOPするかである。LOOPが２階層使用される時は、内側のLOOPは開始フィールドと終了フィールドを指定する。
判定条件IFは、指定フィールドに指定データを持つレコードを検索するのみである。



図－４　配列されたプロシージャー群

第三段階　プロジェクト
この段階で初めて手続き型処理が必要になる。既に作成された１０数ケの基本データファイル群と、１０数ケのプロジージャーがディスク内に存在するとする。何を最終データとして欲しいのか決める。プロシージャー＃１を起動して、複数のデータファイルを順次呼び出し、その処理結果を１つのファイルとして出力する。
次に、出力ファイルを、入力ファイルの１つとして、プロシージャー＃２を作動する。前項と同じようにして、１つの出力ファイルを得る。すなわち、データ駆動型でボトムアップである。（図－４は使用可能なプロシージャの一覧と、処理のために順序づけられたプロシージャー群を示す。）
この操作を十数回（数十回では無い）繰り返すことにより、第三世代言語で数十万ステップから百万ステップを要した事務処理計算が、ビジュアルプログラミングとして実行可能になる。
即ち、各プロシージャーが必要とする入出力データは予め判明しており、最終結果として必要なデータが何か決まれば、プロシージャーアイコンを手続き的に配列することにより、解が得られる。

４　おわりに
現在使用可能なＧＵＩ構築ツールを駆使する事により、既発表のビジュアル言語との比較、従来の文字端末使用の４ＧＬに比べて、生産性の違いを実例を通じてより明確にしてゆきたい。

参考文献
Kunii TL ed (1989) Visual Database System, *Northholland, Tokyo*, p.546
Chang S and Ichikawa T (1987) Visual Language *Pleneum Press, NY*, p.460
Chang S ed (1990) Principles of Visual Programming, *Prentice Hall, p.*372
Shneiderman B (1987) Designing the User Interface, *Addison-Wesley Publishing, NY*, p.523.
ｺﾝﾋﾟｭｰﾀｿﾌﾄ株式会社 (1987)「STYLE ｲﾝﾄﾛﾀﾞｸﾄﾘﾏﾆｭｱﾙ」 p.178