ソフトウェア工学 76－1
（１９９０．１２．４）

有向グラフの階層に基づく
対話型応用ソフトウェア仕様記述の一方法

西尾　高典
（株）日立製作所システム開発研究所

各種業務向け応用ソフトウェアにおける対話処理について，特に，業務手順があらかじめ定まっている定型処理の仕様記述法を提案する．定型処理を構成する画面やフィールドなどの対話要素を中心とした仕様記述を可能とするために，これらの対話要素における処理をクラスとして抽象化し，対話要素間の包含関係をクラス間の階層関係，対話要素から対話要素への遷移を遷移元クラスと遷移先クラスに対する上位クラスからの半コルーチン呼び出しと解釈する．これをモデル化するために，有向グラフの階層を導入し，対話要素間の遷移をこのグラフ階層上での走査として抽象化する方法を考案した．さらにこれに基づく対話処理記述法を考案した．

A Specification Method for Interactive Application Software based on Multi-layered Directed Acyclic Graphs

Takanori Nishio
Systems Development Laboratory, Hitachi Ltd.
1099 Ozenji, Asao-ku, Kawasaki-si, Kanagawa 215, Japan

We propose a specification method for interactive application software. The purpose of our method is to establish "component-oriented" specification, which means that interactive processes should be specified based on components visible to users, such as screens, blocks and fields. In the method, include-relationship between components constructs the hierarchy of classes, which behave as semi-coroutines of their parent classes. A transition is considered as the following sequence: the origin suspends and is detached from its parent, and this calls the terminus. This mechanism can be modelled as multi-layered directed graphs and the traverse on the graph-hierarchy. We show that interaction can be specified in a formal language based on these concepts.

1．はじめに

各種業務向け応用ソフトウェアに対して高度のユーザ操作性が要求されるにつれ，対話処理を中心とする応用ソフトウェア生産技術の向上が重要な課題となってきている[1]．このような対話型ソフトウェアの機能には定型処理と非定型処理とがある．いずれも業務上の問題を解決するための処理であるが，いつ・だれがこの問題解決を行うかという点で異なる．非定型処理では，ユーザ自身の問題解決と計算が実行時に並行して進められるのに対し，定型処理は，あらかじめ設計時に問題を解決して得られた解としての業務手順であり，ユーザはこれに従って計算を進める．非定型処理の設計・開発では，ユーザの問題解決プロセスに柔軟に対応しうる操作体系の提供が課題であるのに対し，定型処理の設計・開発では，問題解決そのものが課題であるといえる．本稿では，これを，ソフトウェア生産技術が一般に直面する問題と同質の課題として着目し[2]，その仕様記述法およびプログラム生成法について論ずる．

定型的な対話処理を，正規言語や文脈自由言語を識別するプロセス[3]としてモデル化するアプローチが盛んである[4][5]．コマンドやメッセージを終端記号，業務手順を生成規則，手順にともなうデータ処理を意味記述に対応づけるという考え方からも明らかなように，これらの研究のおもな関心は，形式言語処理技術の応用としての対話型ソフトウェアの自動生成にある．この手法は，例えば対話機能をもつ計算機言語による直接プログラミングに比べて，記述性をさほど損なわずに簡潔な仕様記述が可能である点ですぐれている．しかし依然として手続き主体の処理記述であり，対話処理の中心である画面やフィールドなどの対話要素を明示的にあつかうことができない．このために，処理記述のなかで実行イメージをとらえることが困難である，画面の構成と処理の構造が対応づけられないなどの問題が生ずる[6]．これは，対話処理を言語処理としてのみとらえることに限界のあることを示している．

本来，対話処理は，画面やフィールドなどユーザの目に見える対話要素を中心とし，これと処理を明示的に関係づけて仕様化すべきである．これによりはじめて見とおしのよい仕様記述が可能となる[7]．本稿では，対話モデルとよぶ対話要素によって構成されるグラフを導入し，これを基礎とすることによって，対話要素を中心とする仕様記述が可能であることを示す．

2．定型処理

2．1　定型処理の概要

定型的な対話処理では，業務手順や業務で用いる画面のレイアウトがあらかじめ定められている（図1）．ユーザから見た対話処理の中心は画面である．手順に従って次々と画面を遷移しながら業務を進める．個々の画面はフィールドから構成される．同時にアクセスできるフィールドの集まりをブロックと呼ぶ．ある画面に遷移したとき，その中の高々ひとつのブロックにアクセス可能であり，これを開いたブロックという．開いたブロックを閉じて別のブロックを開くことをブロック遷移とよぶ．手順に従って次々とブロックを遷移しながら業務を進める．画面の最後のブロックの処理が終了するとその画面自体の処理も終了する．

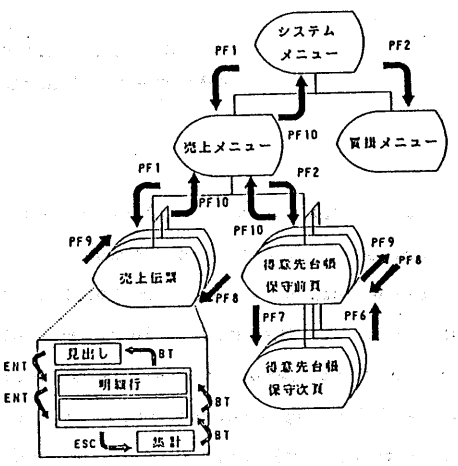

図1．定型処理の例

これを計算機処理の側から見てみよう．対話処理は，計算機のメッセージ出力とユーザのコマンド入力が交互に繰り返されることで進められる．ある画面に遷移した際，その画面を構成するひとつのブロックを開いて，メッセージを出力する．メッセージ出力では，開いたブロックの出力フィールドに記載するためのデータが算出・表示され，コマンド入力待ちの状態となる．この間，開いたブロックの入力フィールドへのユーザ書き込みが可能となる．コマンド入力が発生すると，書き込まれたフィールドデータがとりこまれる．入力コマンドやデータの評価結果に従って次の処理画面またはブロックが決定され，その画面・ブロックに遷移してメッセージ出力に戻る．

2．2　定型処理の「非定型性」

(1)「行きつ戻りつ」

定型処理には，画面やブロックなどの対話要素について，後続する対話要素があらかじめ決定された形で提供されているという特徴がある．複数の後続をもつ場合は，コマンドやデータの評価結果に応じてこれらのなかのひとつが選択される．

後続関係により業務手順があらかじめ設定されているところが「定型的」たるゆえんなのであるが，いわゆるバッチ処理に比べれば依然として「非定型的」である．つまり，設計・開発時にあらかじめ想定した業務手順どおりに業務が進められるとは限らない ── 後続関係で規定される対話要素間の順序が，動的な実行順序と一致するとは限らない ── のである．

代表的な例は，業務手順上の「行きつ戻りつ」の発生である．例えば，ある画面がいくつかのブロックから構成されており，お互いの間に後続関係が定義されているとする．ユーザは，先頭ブロックの処理からはじめて，次々と後続するブロックへと処理を進める．ところが，既入力データの誤りを発見するなど，処理済のブロックを再び処理しなおす場合に，先行するブロックを逆にさかのぼって順次たぐりながら目的のブロックまで戻る必要がある．また，前に処理した画面の内容を参照したいとき，先行する画面を逆にさかのぼって順次たぐりながら目的の画面まで戻る必要がある．このとき，動的な実行順序は後続関係と一致しなくなるのである(図２)．

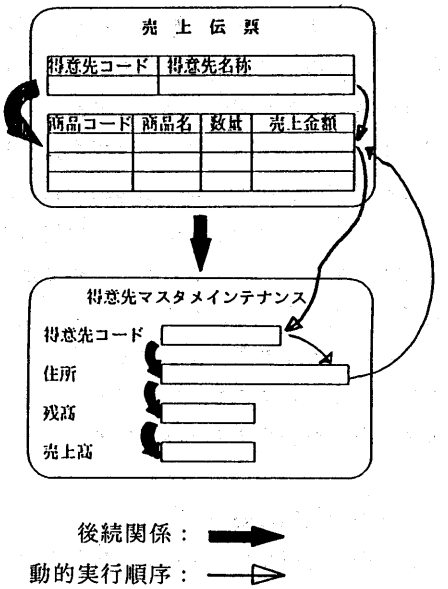

図２．動的な実行順序

(2) 処理の中断と再開

「行きつ戻りつ」が発生しない場合は常に，先行ブロックの処理を終了してから後続ブロックへ遷移するのに対し，「行きつ戻りつ」の場合には必ずしもこのようには実行されない．あるブロックの処理を終了する前に一時的に別ブロックの処理に移り，終了後に再度本ブロックの処理に戻る場合や，あるブロックから処理を移す目的のブロックまでの間に通過するブロックの場合などである．

これを解釈するために，中断・再開という概念を導入する．ある対話要素から別の対話要素への遷移は，遷移元の処理の中断と遷移先の処理の再開である．例えば，通過するだけの対話要素では，他からの遷移時に処理を再開して待ち状態となり，遷移コマンド投入時に処理を中断して他へ遷移する動作として解釈される．

（３）クラスとインスタンス

伝票発行において，同じ伝票形式の画面を繰り返し使用して伝票処理を行う場合や，ひとつの伝票画面が同じ形式の明細行をブロックとして複数個もっておりこれを繰り返し使用して明細処理を行う場合など，同種別画面・ブロックの反復処理を行う場合が多い．共通の形式の画面やブロックではあるが，扱うデータは毎回異なるのであるから，意味的にはそれぞれ別の実体である．

「行きつ戻りつ」が発生しない場合は，例えばこの伝票処理において，前の伝票画面に戻ることがないのに対し，「行きつ戻りつ」の場合には戻る可能性がある．これを実現するためには，意味的に別の実体を別々に扱えなければならない．このためには，対話要素の処理をそれぞれクラスとみなし，新しい処理においてそのインスタンスが生成されたと解釈する方法が有効である．

２．３　定型処理の階層性

ひとつの定型処理は，一般に，伝票入力・伝票発行・台帳更新など，いくつかの業務から構成される．個々の業務は複数の画面によって実行され，さらに個々の画面は複数のブロックから構成される．このように，定型処理を構成する種々の対話要素には包含関係があり，これらが入れ子構造になって処理を構成する．

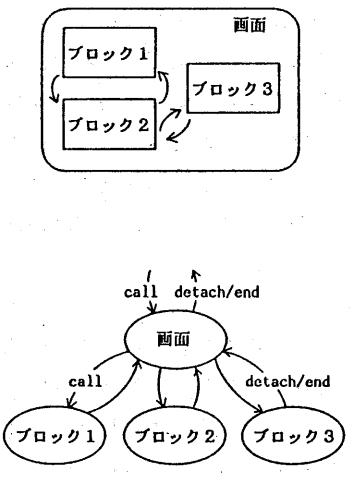

図３．対話処理の定式化

対話要素間の包含関係は，業務間の遷移や画面間の遷移，ブロック間の遷移のように，遷移の階層を形成している．例えば，ある業務の画面のブロックを現在処理中のとき，その画面上の他ブロックへの遷移，その業務の他画面への遷移，他業務への遷移が可能である．階層をもつ遷移は，（２）で述べた中断・再開の概念によって解釈できる．他業務への遷移の場合は，現在の画面およびそのブロックで中断し，他業務の中断している画面およびそのブロックから再開する．同じ業務内の他画面への遷移の場合は，現在のブロックで中断し，他画面の中断しているブロックから再開する．

２．４　定式化

定型処理全体はクラス[8]の階層構造から構成され，個々のクラスが対話要素の処理を定義すると考える．クラスには，個々の画面やブロックに対応するものが含まれる(図３)．

共通の親をもつクラスは，後続関係についての半順序集合を形成する．自分の親より先行／後続する親の子孫のクラスはすべて自分より先行／後続すると考える．後続関係は半順序関係であるので，ある対話要素が後続する対話要素に後続することはないものとする．

クラスの階層において，子クラスは親クラスの半コルーチンと考える．あるクラスのインスタンスは，自分の初期処理を実行したのち，子のインスタンス生成，呼出(call)を次々と実行し，終了時に自分の終了処理を実行しその親に戻る(end)．ここで次に生成・呼出をする子の選択は，後続関係とそのときユーザが投入した遷移コマンドによって決定される．自分の祖先の遷移が指定された場合は，処理を停止(suspend)して親に制御を戻す(detach)．３章で提案する対話モデルは，以上述べたような定型処理の形式的記述の基礎となるモデルである．

3．対話モデル

対話処理システムの仕様は，その処理構造と，ユーザコマンドの意味定義によって決定される．対話モデルは，対象システムの処理構造をモデル化すると同時に，コマンドの意味づけの基礎を与える．

3．1 対話モデルの構造

対話処理システムを，非循環有向グラフ[9](Directed Acyclic Graph)の階層を走行する機械としてモデル化する．このとき，対話処理はこの階層構造の走査としてモデル化される．このグラフの階層構造のことを対話モデルと呼ぶ(図４)．

対話モデルを構成する個々のグラフGは，頂点の集合Vと弧の集合Eから構成される（〈V，E〉と表記）．頂点は対話要素を抽象化したもの，弧は後続関係を抽象化したものである．弧e∈Eは２つの頂点v_1，v_2∈Vの順序対（v_1,v_2）である．v_1，v_2をそれぞれ弧eの始点，終点と呼ぶ．(v_1,v_2)，(v_2,v_3)，･･･，(v_{n-1},v_n)がすべて存在するような頂点の集合P＝{v_1,v_2,･･･,v_n}を経路，v_1とv_nをそれぞれ経路Pの始点と終点と呼ぶ．このときv_1からv_nへ到達可能であると言う．対話モデルを構成するグラフはすべて非循環であるという特徴を有する．つまりグラフ上のすべて頂点について，そこから到達可能なすべての頂点からもとの頂点へは到達不可能である．従ってグラフGは有限集合上の半順序を表現している．各グラフは，これに属するすべての頂点vについて（v_0，v）となるような唯一の上限頂点v_0をもつとする．

グラフG上の頂点v∈Vが他のグラフG'を参照するときGをG'の親であるとみなすことにすると，親子関係により階層構造が構成される．この親子関係は，対話要素間の包含関係を抽象化したものである．個々のグラフは，複数の親をもつことができる一方，それ自身の子孫を親にもつことはできない．すなわち対話モデルを構成するグラフの集合において，この親子関係は半順序である．この集合は他のすべてのグラフより上位にある唯一のグラフG_0をもつ．

個々のグラフGは，これを参照する他グラフの頂点ごとに有限個のインスタンスをもつことができる．あるグラフGの，これを参照するある頂点vに関して生成されるインスタンスは，生成された順に順序づけられており，その序数によって例えばv(1),v(2),･･･,v(N)などと表記して識別する．

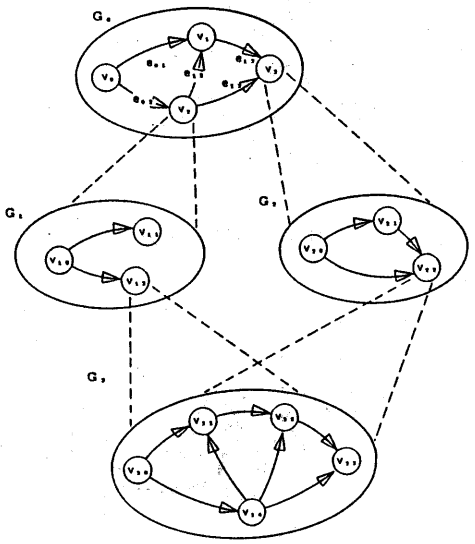

図４．対話モデル

3．2 システムの状態

システムの状態は，システムを構成する個々のグラフのインスタンスの状態によって一意に決まる．各インスタンスの状態空間は，対応するグラフを構成する頂点ごとに，これが参照する他グラ

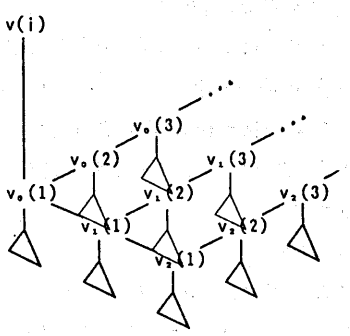

図５．インスタンス木

フのインスタンスの集合から構成される．例えば，あるグラフがv_0, v_1, v_2の３頂点から構成されるとすると，このグラフの各インスタンスの状態空間は$\{v_0(1), v_0(2), \cdots, v_1(1), v_1(2), \cdots, v_2(1), v_2(2), \cdots\}$となる．状態空間の要素を子とみなせば，これは，グラフの階層に対応するインスタンスの木を形成する(図５)．各インスタンスの状態は，これのもつ子インスタンスのいずれかである．初期状態は，対応するグラフの上限頂点の第１のインスタンスである．

インスタンス木の最上位頂点から現在の状態を次々とたどって最下位頂点にいたる経路が必ず唯一存在する．システムの状態は，例えば「もしこの画面を開いたらこのブロックが開くはず」というような潜在的な状態を示すものであるのに対し，この経路を構成する状態のベクトルは，ある時刻において実際に何が開かれているかを示すものであり，表示状態と呼ぶことにする．

３．３　対話モデル上の遷移

(１) 遷移の意味

遷移は，対話処理の内容に応じてユーザがコマンドによって指示するものである．これは，システムのある表示状態から別の表示状態への遷移として意味付けられる．先に述べたとおり，表示状態はインスタンス木の根から葉までの経路によって定義されるので，別の表示状態への遷移とは，その経路のなかのひとつのインスタンス（これを操作対象とよぶ）の状態がある子から別の子に移ることを意味する(図６)．

したがって，遷移は，操作対象となるインスタンスと操作種別opの組 $\langle G, op \rangle$ で意味づけられる．ここで操作種別opとは，操作対象インスタンス上の頂点$v \in V$から頂点$v' \in V$への訪問のしかたを意味する．

(２) 操作種別

グラフ$G = \langle V, E \rangle$のインスタンスを対象とする操作として，次，元，後，前，先頭，末尾の６つを設定する(図７)．

次操作は，頂点$v \in V$から，これを始点とする弧$e \in E$の終点の頂点v'への訪問である．元操作は，頂点$v \in V$から，これを終点とする弧$e \in E$の始点の頂点v'への訪問である．次操作により，インスタンス$v(i)$からインスタンス$v'(1)$への遷移が発生する．v'のインスタンスがまだ存在しないときは，$v'(1)$の生成とこれへの遷移が発生する．また，元操作により，インスタンス$v'(i)$からインスタンス$v(1)$への遷移が発生する．v'が上限頂点のときはこの操作は無効である．

後操作，前操作はともに，頂点$v \in V$から自分自身への訪問である．後操作により，インスタンス$v(i)$からインスタンス$v(i+1)$への遷移が発生する．$v(i)$が最後に生成されたインスタンスである場合には，$v(i+1)$の生成とこれへの遷移が発生する．また，前操作により，インスタンス$v(i)$からインスタンス$v(i-1)$への遷移が発生する．iが１のときはこの操作は無効である．

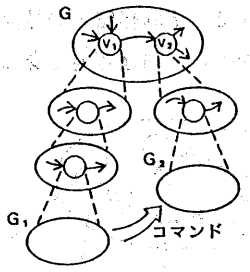

図６．遷移の意味

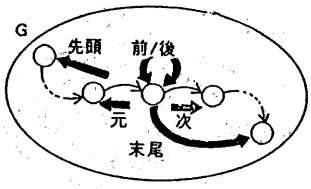

図７．グラフ操作の種別

先頭操作，末尾操作は，それぞれGの上限と(もしあれば)下限への訪問である．それぞれ元操作，次操作の閉包と考えられる．

(3) コマンドの定義方法

コマンド定義 ── 制御キーCとグラフ操作〈G,op〉との組〈C,〈G,op〉〉 ── は，操作対象G以下のすべてのグラフにおいて定義できる．定義場所のグラフより下位のすべてのグラフにおいてこの定義が有効となる．また，あるコマンドの定義が重複する場合は，そのうちの最下位のグラフでの定義が優先する．

例えば，グラフG。上において，システムメニューから伝票メニュー，伝票メニューから伝票への遷移を，コマンドpf1によって，逆に伝票メニューからシステムメニュー，伝票から伝票メニューへの戻りを，コマンドpf2によって実行する場合，〈pf1,〈G。,元〉〉，〈pf2,〈G。,次〉〉をG。で定義しておけば，これらのブロックで有効となる．

4．対話処理定義言語

前章で述べた対話モデルに基づいて対話処理仕様を定義するためのひとつの言語の文法の概略を図8に示す．

<対話要素定義>は，対話処理の含む対話要素ごとにその処理内容を定義するものであり，対話モデルにおけるグラフの定義に対応する．この定義のなかの初期処理・終了処理に続く<処理記述>は，それぞれその対話要素のインスタンス生成時，処理終了時に実行される処理内容を定義する．構造定義に続く<構造記述>は，その対話要素が下位にもつ他の対話要素の参照とそれらの間の半順序関係を定義する．これは，グラフを構成する頂点の集合およびそれらの半順序を定義するものである．コマンド定義に続く<コマンド記述>は，その対話要素を含む下位対話要素において有効なコマンド群を定義する．<コマンド記述>は，前章でのべたコマンドの定義＜C,＜G,op＞＞を言語的に記述するものである．ユーザに見える機能キーなどの物理的なコマンドと対話モデル上の遷移操作とを結び付ける役割を果たす．

5．対話モデルの特徴

対話処理記述に一般に広く用いられている状態遷移モデル[2]と比較しながら，対話モデルの特徴を述べる．

対話モデルの最も大きな特徴は，グラフの頂点が処理を表しているという点である．一方，状態遷移モデルでは弧(「遷移」)が処理を，頂点が「待ち」状態を表す．

状態遷移モデルにおける一つの遷移(入出力変換)は，対話処理サイクルの「メッセージ受信」から「メッセージ送信」までの処理に対応する(図9)．「遷移先決定」の直後に画面が切り替わるので，一つの遷移が，遷移元，遷移先の画面処理を合わせ持つことになる．これは，仕様記述を著しく複雑にする．この問題を解消するためには，「出力メッセージ作成」から「遷移先決定」までを一つの処理とするのが望ましい．対話モデルでは，「メッ

```
<対話処理定義>  =  <対話要素定義> 。。。
<対話要素定義>  =  <対話要素名>
                     初期処理     <処理記述>
                     構造定義     <半順序指定>{;}。。。
                     コマンド定義 <コマンド記述>{,}。。。
                     終了処理     <処理記述>
                   処理定義終了
<半順序指定>    =  <対話要素名> -> <対話要素名>{,}。。。
<コマンド記述>  =  <コマンド種別>:<処理記述>
<コマンド種別>  =  PF1 | PF2 | ・・・ | PF24 |
                   PA1 | PA2 | PA3 | ENT
<処理記述>      =  <文>{;}。。。
<文>            =  <繰返し文> | <選択文> | <代入文> | <操作文>
<操作文>        =  <操作名>(<対話要素名>匚,<次指定>コ)
<操作名>        =  先頭 | 末尾 | 次 | 元 | 後 | 前 | 終了 | 取消
<次指定>        =  <正整数>
```

図8．対話処理記述言語文法

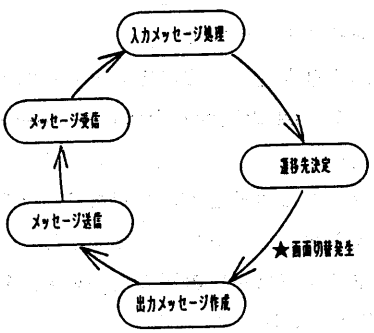

図９．対話処理サイクル

セージ送信」／「メッセージ受信」の対が，下位グラフ訪問／戻りの対と適合するので，頂点を画面処理と対応づけることが可能となり，上記の問題は解決される．

状態遷移モデルでは，遷移が受信／送信の単位なので，ブロック以外の対象の遷移をモデル化することは出来ない．これは，画面処理，ブロック処理等が一つの遷移に混在することを意味し，仕様の簡潔性を損なう．対話処理を対象別かつ階層的に仕様化できることが，了解性向上のために望ましい．対話モデルの場合，一つの頂点の詳細化は，処理のトップダウン分割と対応する．従って，画面処理，ブロック処理等をその階層毎に完全に分離して記述することができる．

３.で述べたとおり，対話モデルは，処理の停止／再開をモデル化しており，操作性の高い対話型ソフトウェアを実現できる．また，前／後操作に示されるとおり，インスタンス列に対する操作をもつので，例えば複数頁にわたる伝票画面に対して前頁／後頁の操作，表の各行に対して前行／後行の操作を提供できる．

６．おわりに

ＯＡシステムに搭載される対話型応用ソフトウェアの仕様記述法及びプログラムの自動生成法を提案した．有向グラフの階層を対話モデルとして導入し，この上で対話処理におけるオペレータの操作を定式化したことに特徴がある．複雑な遷移処理を簡潔に仕様可能であると同時に，従来別々に扱われてきた画面間の遷移処理，画面内の項目遷移処理などを，統一的に扱うことが可能となる．

参考文献

[1]Bos,J.V.D,Plasmeijer,M.J.,Hartel,P.H.: Input-Output Tools:A Language Facility for Interactive and Real-Time Systems,IEEE Trans. on Softw. Eng.,Vol. SE-9,3,pp.247-259 (1983).

[2]本多波雄：オートマトン・言語理論，コロナ社，東京 (1972).

[3]Reinsner,P.:Formal Grammer and Human Factors Design of an Interactive Graphics System,IEEE Trans. on Softw. Eng.,Vol.SE-7,pp.229-240 (1981).

[4]Pilote,M.:A Programming Language Framework for Designing User Interfaces, in Proc. SIGPLAN '83 Symp. Program. Lang. Issues in Softw. Syst.,pp.118-133 (1983).

[5]Olsen,D.R.,Dempsey,E.P.:Syntax Directed Graphical Interaction, in Proc. SIGPLAN '83 Symp. Program. Lang. Issues in Softw. Syst., pp.112-117 (1983).

[6]Draper,S.W.,Norman D.A.:Software Engineering for User Interfaces,IEEE Trans. Softw. Eng.,Vol. SE-11,pp.252-258 (1985).

[7]西尾，今城，千吉良：有向グラフの階層に基づく対話型応用ソフトウェアの仕様記述の一方法，情報処理学会第39回全国大会予稿集 (1989).

[8]Dahl,O.J. et al.:Structured Programming, Academic Press Inc.,London (1972).

[9]Aho,A.V., Hopcroft,J.E.,Ullman,J.D.:Data Structure and Algorithms,Addison-Wesley (1983). 邦訳：大野．データ構造とアルゴリズム，培風館，1987.