ソフトウェア工学 42－1
（１９８５・６・２６）

インタラクティブ・プログラムの属性文法による合成

大石　東作　　長尾　順太郎＊　　大蒔　和仁　　高山　文雄
（電子技術総合研究所　　＊㈱日本ビジネスコンサルタント）

1、はじめに

我々は、現在整構造高度並列計算アーキテクチャ用のプログラム作成支援システム［１］を開発中である。このシステムのみならず、今日の多くの応用システムは、ＣＲＴ端末を介するインタラクティブ・プログラムとして実現されるのが常である。我々は、使いやすいインタラクティブ・プログラムの作成にあたって、属性文法による記述という統一的な実現方法で上記のシステムを効率的に開発している。

2、これまでのインタラクティブ・プログラムの作成法

インタラクティブ・プログラムにおいては、その使いやすさが性能の評価の重要な尺度となる。しかしながら、使いやすいシステムの設計法・実現法は、いまだ確立されおらず、各システムごとにａｄ　ｈｏｃに実現しているのが現状である。一般の応用システムの開発においても、プログラムを実際に作成し、これを試用してその問題点を洗い出し、改良を重ることにより、最終的な使いやすいシステムへと収束させていくのが普通であるが、インタラクティブ・プログラムの場合は、特に試用結果の設計仕様へのフィードバックが不可欠である。

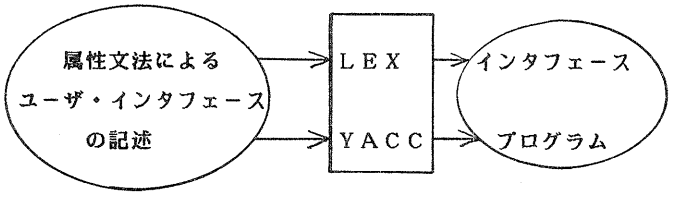
このようなインタラクティブ・プログラムを従来のプログラム言語で記述していたのでは、プログラム作成の生産性は低く、よってシステム改良のターン・アラウンド時間が長くなり、ひいては改良が実施されない場合もある。また低レベルのバグの発生も防止しがたい。

3、属性文法によるユーザ・インタフェースの記述

インタラクティブ・プログラムが使いやすいためには、プログラム自体の充実もさることながら、ユーザ・インタフェースやコマンド体系が良く出来ていることが重要である。良くできたユーザ・インタフェースを作成するには、良い設計が不可欠であるが、さらにシステムのプロトタイプを早期に作成し、従来法と同様にして最終的なプログラムへと仕上げていくことが必要となる。問題は、いかにしてこのプロセスを素早くかつ誤りなく実施するかである。そのためには、まずユーザ・インタフェースの形式的仕様を記述することが必要である。

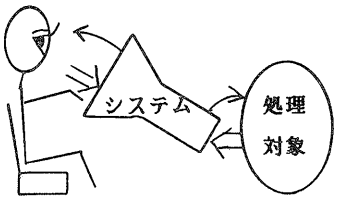
ユーザ・インタフェースの形式的仕様を記述する方法としては、状態遷移図によるものと構文規則（例：BNF記法）による２法が知られている［２］。両者は形式的にほぼ等価であり、また状態遷移図による方が読解性が良いとされている。しかし、同方法を利用するためのツールを得られないのが難点である。そこで、我々は構文規則にもとづく方法を採用することにした。

すなわち上記のプロセスを統一的な観点から実現するために、ユーザ・インタフェースの入力部の形式的仕様を属性文法で記述し、これを第１図のようにＵＮＩＸのＬＥＸ，ＹＡＣＣ［３］で処理することによりインタラクティブ・プログラムを合成している。



第１図、ＬＥＸ，ＹＡＣＣによる処理

従来、属性文法はプログラム・モジュールの仕様記述［４］、言語指向エディタの作成［５］等に応用されてきた。これらは第２図のシステムにおいて、処理対象であるデータが１種の言語表現（例えばプログラム）であるとし、属性文法による処理をほどこしている。我々はさらに同システムにおけるユーザ・インタフェース部にも別種の言語表現（コマンド言語体系）



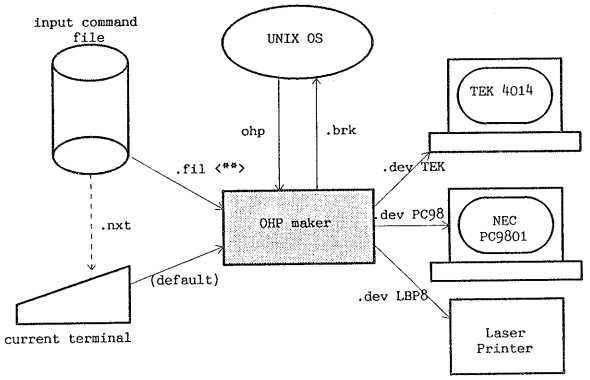
（コマンド言語）　（プログラム言語）

第２図、インタラクティブ・システムにおける２種類の言語

があるとして、これをも属性文法で処理することにした。すなわち２種類の言語体系が処理される。ユーザ・インタフェースにおけるコマンド処理をこの方法で実施すると、次のような利点が得られる。（属性）文法は、言語に対するメタ記述であるから簡潔に記述でき、合成されるプログラムの仕様とみなすこともできる。記法自体も、広く用いられているＢＮＦ記法に近いものであり、一般になじまれており理解しやすい。また、属性部により構文に対応する意味処理も記述できる。このような記述から、現実のプログラムが合成されるからラピッド・プロトタイピングの有力な方法の一つとなっている。記述の簡潔性は、変更（修正、追加、削除）の容易さをもたらし、不注意によるバグの発生を防ぎ、システムのカスタマイズやチューンアップも容易となる。合成されたプログラムの処理速度やメモリー所要量が実際的なものであれば、プロトタイプのみならず最終版システムとしても利用できる。処理速度についても、コマンド処理の場合は人間の思考時間や操作時間により遅さが相殺されることもあるから、バッチ処理的なプログラム言語のコンパイラの場合ほど制約を受けない。

４、属性文法による記述例（ＯＨＰ原稿作成プログラム:OHP maker）

ＯＨＰ原稿作成プログラムは、第３図にしめす概要をもち、ファイルまたは端末からＲＵＮＯＦＦ風コマンド列を入力して、レーザ・プリンタに、ＯＨＰ原稿を出力する。原稿の出来映えを、前もってグラフィック端末で確認することもできる。第４図にコマンド列の一例をしめす。文頭が”．”で始まるものは制御・描画コマンドであり、その他が地の文に相当する。制御コマンドで、出力装置(.DEV)・PEN番号(.PEN)・文字の大きさ(.WID)等を指定する。描画コマンドで線分等を描く。地の文は、制御コマンドの指定に従って文字列として描かれる。



第３図　ＯＨＰ原稿作成システムの概要

```
.ce
.thk   3
.term  300 1500 200 100
start
.arw2  100 0
/*.box 650 1500 300 200 */
.box4  0 0 300 150
fn.1
.arw0
.box2  0 0
fn.2
.arw2  200 0
.box2  0 0
fn.3
.arw2  50 0
.term4 0 0 200 100
end
.mova  450 1500
.arw2  50 -250
.box4  0 0 300 150
fn.4
.lseg2 450 0
.arw2  0 250
.mova  450 1250
.arw2  450 -250
.box2  0 0
fn.5
.lseg2 100 0
.arw2  0 500
.mova  750 800
An Example of The Higpl Program
.ff
.nxt
```

第４図　コマンド列の例

ＬＥＸ，ＹＡＣＣによる処理を前提として、前記の仕様を記述すると２つの形式が必要となる。一つは、ＯＨＰ原稿作成用の各コマンド・文字列・数字等を正規表現で表わし、これらをトークンとし定義するもので字句解析プログラム生成系ＬＥＸへの入力となる。同定義例を第５図（次頁）にしめす。もう一つはコマンド列の構文

```
%{
#include       "y.tab.h"
extern int     yylval ;
extern int     linnum ;
%}
%%
"/*".*"*/"          { ; } /* Comment */
^"."(dev¦ DEV)      {   return( DEV ) ; }
^"."(wid¦ WID)      {   return( WID ) ; }
^"."(pen¦ PEN)      {   return( PEN ) ; }
^"."(spc¦ SPC)      {   return( SPC ) ; }
^"."(lfd¦ LFD)      {   return( LFD ) ; }
^"."(ang¦ ANG)      {   return( ANG ) ; }
^"."(cr¦ CR)        {   return( CR  ) ; }
^"."(ncr¦ NCR)      {   return( NCR ) ; }
^"."(ce¦ CE)        {   return( CE  ) ; }
^"."(nce¦ NCE)      {   return( NCE ) ; }
^"."(thk¦ THK)      {   return( THK ) ; }
^"."(drwa¦ DRWA)    {   return( DRWA ); }
^"."(drwr¦ DRWR)    {   return( DRWR ); }
^"."(mova¦ MOVA)    {   return( MOVA ); }
^"."(movr¦ MOVR)    {   return( MOVR ); }
^"."(lseg¦ LSEG)    {   return( LSEG ); }
^"."(lseg2¦ LSEG2)  { return( LSEG2 );}
^"."(lseg0¦ LSEG0)  { return( LSEG0 );}
^"."(arw¦ ARW)      {   return( ARW  ) ;}
^"."(arw2¦ ARW2)    {   return( ARW2 ) ;}
^"."(arw0¦ ARW0)    {   return( ARW0 ) ;}
^"."(rect¦ RECT)    {   return( RECT ) ;}
^"."(box¦ BOX)      {   return( BOX  ) ;}
^"."(box2¦ BOX2)    {   return( BOX2 ) ;}
^"."(box4¦ BOX4)    {   return( BOX4 ) ;}
^"."(circ¦ CIRC)    {   return( CIRC ) ;}
^"."(circ2¦ CIRC2)  { return( CIRC2 );}
^"."(circ3¦ CIRC3)  { return( CIRC3 );}
^"."(term¦ TERM)    {   return( TERM ) ;}
^"."(term2¦ TERM2)  { return( TERM2 );}
^"."(term4¦ TERM4)  { return( TERM4 );}
^"."(fil¦ FIL)      {   return( FIL  ) ;}
^"."(wrt¦ WRT)      {   return( WRT  ) ;}
^"."(ff¦ FF)        {   return( FF   ) ;}
^"."(nxt¦ NXT)      {   return( NXT  ) ;}
^"."(brk¦ BRK)      {   return( BRK  ) ;}
^[^"."].*           {   return( STRING);}
"-"                 {   return( MINUS );}
[0-9]+
 { yylval = atoi(yytext); return(NUM);}
[/"."A-Za-z][/"."0-9A-Za-z]*
 { return( FN ) ;   }
[ \t,]              {  ; }
\n { linnum  =linnum+1; return( NL ); }
%%
```

第５図　各トークンの定義

形式と対応する意味処理を定義する属性文法であり、これは構文解析プログラム生成系ＹＡＣＣへの入力となる。同定義例を第６図にしめす（紙面の都合でかなり省略した）。

```
%{
#include  <stdio.h>
  char *dev_name,*fil_name ;
  char sttext[100] ; int stleng ;
  extern char yytext[] ;
  extern int  yyleng, yyval
%}
%start list
%token DEV WID PEN CR NCR THK DRWA DRWR MOVA MOVR LSEG
%token RECT BOX BOX2 BOX4 CIRC TERM
%token NXT FIL BRK FF FAC ARW PNT CE NCE NUM STRING NL FN
%left  MINUS
%%
list : /* empty */   /*..... rule .....*/
     ¦  list NL
     ¦  list  expr NL
     ¦  list STRING { store(); } NL
                    { strout( sttext,stleng ) ; }
     ¦  list  error NL {yyerrok;} ;
expr :  DEV  FN { dev_name=yytext; dev( dev_name ) ; }
     ¦  FIL  FN { fil_name=yytext; fil( fil_name ) ; }
     ¦  WID  width hight    { wid( $2, $3 ) ; }
     ¦  PEN  number         { pen( $2 ) ; }
     ¦  ANG  angle          { ang( $2 ) ; }
     ¦  CR                  { cr()      ; }
     ¦  NCR                 { ncr()     ; }
     ¦  CE                  { ce()      ; }
     ¦  NCE                 { nce()     ; }
     ¦  THK  number         { thk( $2 ) ; }
     ¦  DRWA xpos ypos      { drwa( $2, $3 ) ; }
     ¦  DRWR xrel yrel      { drwr( $2, $3 ) ; }
     ¦  MOVA xpos ypos      { mova( $2, $3 ) ; }
     ¦  MOVR xrel yrel      { movr( $2, $3 ) ; }
     ¦  LSEG xpos ypos xrel yrel { lseg( $2, $3, $4, $5 ) ; }
     ¦  ARW  xpos ypos width hight { arw ( $2, $3, $4, $5 );}
     ¦  RECT xpos ypos width hight { rect( $2, $3, $4, $5 );}
     ¦  BOX  xpos ypos width hight { box( $2, $3, $4, $5 ) ;}
     ¦  BOX2 width hight           { box2 ( $2, $3 ) ; }
     ¦  BOX4 xrel yrel width hight { box4 ( $2, $3, $4, $5);}
     ¦  CIRC xpos ypos diameter    { circ( $2, $3, $4 ) ; }
     ¦  TERM xpos ypos width hight { term( $2, $3, $4, $5 );}
     ¦  NXT                 { nxt() ; }
     ¦  BRK                 { ohpbrk() ; }
     ¦  FF                  { formfeed() ; } ;
xpos :   number { $$ = $1 ;} ;
ypos :   number { $$ = $1 ;} ;
xrel :   number { $$ = $1 ;} ;
yrel :   number { $$ = $1 ;} ;
width  : number { $$ = $1 ;} ;
hight  : number { $$ = $1 ;} ;
diameter : number { $$ = $1 ;} ;
angle  : number { $$ = $1 ;} ;
number : NUM    { $$ =   $1 ; }
     ¦ MINUS NUM { $$ = - $2 ; };
%%
store()
{ int i;
  for(i=0;i<=yyleng;i++) sttext[i]=yytext[i];
  stleng = yyleng; }
%
```

第６図　属性文法の定義

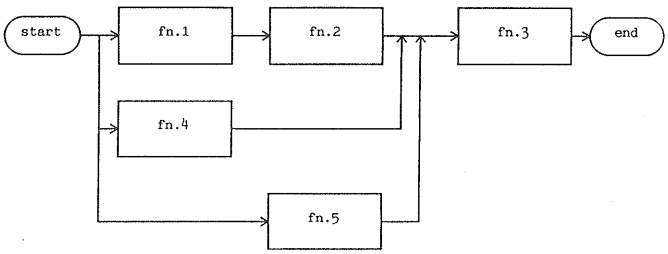
第６図の本プログラムの属性文法による記述例において、ルールのlist部で、空文、制御・描画コマンド(expr)、地の文(STRING)、エラー(error)の切り出しを指定している。地の文部は、第２図における処理対象に相当する部分であるが応用の性質上から何も処理しないでそのまま文字列として出力している。expr部では、各

コマンドごとに詳細な解析を実施し属性記述により意味処理部の起動とパラメータ渡し({・・・;}部)を指定している。パラメータxpos yrel,hight等はnumberに相当するものであるが、読解性を高めるためにあえて別の非終端記号として定義している。このような冗長な表現は構文解析の効率を悪くするが、仕様の意味定義には不可欠である。

```
drwa ( x,y )
int x,y  ;
{
 int dx,dy;
 switch( fdev ) {
  case TEK :
  case PC98 :
   fprintf(yyout,"%c",GS); putpos(cx,cy);
   adjust( &x,&y ); putpos(x,y); fprintf(yyout,"%c",US); break ;
  case LBP8 :
   fprintf(yyout, "%c",0x31); putpos(cx,cy);
   dx=x-cx; dy=y-cy; putpos(dx,dy);
   fprintf ( yyout, "%c", IS2 ) ; /* gend */ break ;
  }
 cx=x; cy=y;
 lastatr ( &tatype, &tax, &tay, &taxr, &tayr, &tawidth, &tahight ) ;
 setlatr ( getltype( x,y ), x, y, tax, tay,    tawidth, tahight ) ;
}
```

第７図　意味処理部の本体（ごく一部）

コマンドの解析は、終端記号（token）に到達するまで実施される。意味処理部の本体は、通常のCプログラムであり出力装置ごとの処理が記述されている。第７図に.drwa コマンドに対応する部分をしめした。



An Example of The Higpl Program

第８図　処理結果の一例

これらの仕様とプログラムをVAX-750 UNIX上で処理し実行させたところ期待どうり結果が得られた。第４図のコマンド列を処理した結果を、第８図にしめす。合成されたOHP maker プログラムはメモリ所要量、処理速度の点でも特に問題はなかった。応用の規模からして当然のことと考えられる。この例の他に実用規模の画面エディタのコマンド解析部（コマンド数：約１２０）もこの方法で記述している。

５、属性文法による方法の可能性と問題点

上記のOHP maker プログラムの第１バージョンが出来たところで試用してみると、基本的な描画コマンドは完備しているにもかかわらず、非常に使い勝手が悪かった。これは描画プログラム一般に言えることであるが、ほとんどの描画において図形要素の配置のための絶対座標の指定が必要であるという原因による。そのために。グラフ用紙の上にまず下絵を手書きで描いて絶対座標値を決定する等大変手間がかかる作業を強いられる。

そこで.arw2,box2,box4 等のコマンドを追加して、相対値の多用、前回使用値の再利用、図形要素の端点座標の自動計算等をはかり絶対座標値の使用を最小限にとどめることを可能とし、結果としてOHP maker プログラムの使い勝手は大幅に改善され、十分使用に耐えるものとなった。コマンドの追加にあったては、1)コマンド・トークンの追加、2)属性文法によるコマンドの意味処理の追加、3)意味処理本体部の機能追加の３箇所に局所化された。意味処理本体部も、もともとの基本機能に付加的な情報（相対値、前回値）を追加して利用することになり、システムの基本構成を変えることもなっかつた。このためコマンドの追加はごく容易であったにもかかわらずプログラムの高機能化がはかられた。また従来のプログラムにおける機能強化にありがちな、プログラムの構造の良さが崩れ読解性が悪くなることも全くなかった。さらに省略形式のコマンドの追加も、トークンの正規表

現部への同型式の追加のみで簡単に実施できた。

上のような経験から、属性文法による方法はプロトタイプ作成や試用結果の設計仕様へのフィード・バックによるシステムの調製、使いやすいユーザ・インタフェースの作成等に十分強力であるという認識を得た。

一方出来上がったシステムの性能の点から見ると前記のように特に問題はなかったが、メモリ所要量についていま少し見てみよう。第１表がOHP maker システム各部のメモリ所要量である。LEX,YACCによって生成された文字解析・構文解析プログラムはいずれも表駆動型であり、プログラムのなかで表の占める領域が大きい。これは文字解析プログラムで著しく、全体のメモリ量でもかなり大きいので実行速度の点でも問題があろう。コンパイラ等に使用するするのであれば、文字解析部のみを手書きするのも有効であろう。また、YACCが生成するのはLALR(1) 型の上昇型パーザであるが、実用上の効率（メモリ所要量・実行速度）からすると、PASCALのコンパイラ等において実績のある再帰下降型のパーザを最終的には手書きするのも実用システムを作成する上での一方法であると考えられる。

	仕様定義	Cプログラム	オブジェクト	
字句解析部	3625	22815	19057	(LEX)
構文解析部	4485	15196	6626	(YACC)
意味処理本体部	−	25122	25732	
プログラム全体	−	−	45297	

第１表　システム各部のメモリ所要量　（単位：バイト）

６、おわりに　　属性文法特にＬＥＸ，ＹＡＣＣを用いる方法の現状における最大の問題点は、それらの処理が字句および構文解析部の自動生成に限定され、応用プログラムの意味的処理部本体の生成には全く有効でない点にある。この方法をさらに有効なものとするには、意味処理部に蓄積された「知識」［６］としてのプログラム・モジュール等を活用することが必要となろう。

最後に研究の機会をあたえられた柏木電子計算機部長、日頃御指導いただく棟上ソフトウェア部長、佐藤情報システム研究室長、二木言語処理研究室長、植村プログラム研究室長に感謝します。

参考文献

［１］大石　他”整構造高度並列計算アーキテクチャ用プログラム作成支援システムの構成”情処学会２９回全国大会7P-8

［２］R.J.K.Jacob,"Using formal specifications in the design of a human-computer interface",Com. of ACM,Vol.26,Num.4,1983

［３］UNIX Support Tools Guide,Western Electric,1982

［４］T.Katayama:"HFP:A hierachical and functional programming methodology based on attribute grammer",TR. of TIT,CS-K8002,1980

［５］大蒔　他”属性文法に基づく言語指向型エディタ作成システム”信学論,Vol.J67-D,No.1,pp.25-32,1984

［６］大石”汎用問題解決システム(Meta-88)における「知識」ベース、情処２６全大会３Ｃ－２