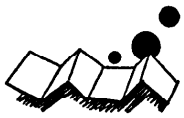
解　説



# 標準パターン活用の構造エディタ†

大筆　豊††　会田一夫†††　小尾俊之††

## 1. はじめに

エディタは，従来コーディング段階のみを支援するツールであった．最近では，人間と計算機のインタフェース機能を重視して，ソフトウェア開発環境としての機能を追加するものが多い．

本稿ではエディタを次のように分類して考えることにする．

（1） 一般的な文字列を編集するエディタ

（2） 特定の言語によるプログラム開発を支援するエディタ

（3） 特定のプログラム開発手法を支援するエディタ

特定言語によるプログラム開発の支援の例として，ほとんどのパソコンに備えられている BASIC のエディタがある．文法エラーを会話的に指摘するほか，内部的には解釈実行が容易に行えるような中間形式に変換する機能がある．UCSD-PASCAL[1]，Interlisp[2]，PL/I のサブセットである PL/CS を支援するコーネル大学の Cornell Program Synthesizer[3]，カーネギーメロン大学の Gandalf システムの ALOE エディタ[4] 等例が多い．これらのシステムではいずれも，エディタがプログラミング言語の文法知識を持ち，文法チェックを編集時に行うとか，構文通りに入力を強制するような方式 (Syntax Directed Editor) により，結果的には文法的に正しいプログラムが得られるようになっている．

特定のプログラム開発手法を支援する例として，段階的詳細化 (step-wise refinement) の手法を支援する DUAL[5] や本稿で解説する標準パターンを用いて設計文書とプログラムの対応関係を明確にしようとする大阪大学のπエディタ[6),7)] および筆者らが開発している SPISE[8),9)] もこの分類に入る．

一般的な文字列を編集するエディタを別にすると，編集対象を内部的に保持するのに，単なる文字列としてではなく，編集目的に応じた構造を持たせているものが多い．また通常の編集（文字列の挿入や変更）を行うものに加えて，編集対象のもつ構造に対するコマンドを持つ．このようなエディタを構造エディタと呼ぶ．

本稿では，構造エディタのうち，あらかじめ作成された標準パターンを用いて編集を行うエディタ（πおよび SPISE）の，設計思想およびその特徴について解説する．

## 2. πエディタ

プログラムの保守費用の比率は増加する傾向にある．保守時に満足して使える品質を持ったドキュメントはなかなか得にくいのが現実である．ドキュメントの質が悪いとは

（1） 必要な情報が欠除している．

（2） ソースプログラムとドキュメントの間に矛盾がある．

（3） 必要な情報を莫大なドキュメントの山から探すのが困難である．

のいずれかの欠点を持つ．その原因として

（1） コーディングやデバッグが終ってからドキュメントの作成を行うなどドキュメント作成の時期が良くない．

（2） ドキュメントとして何を記述すべきか明確に示されていない．

（3） ドキュメントの記述方法や管理方法が適切でない．

ことが挙げられる．これらの問題点を解決しようとするのがπエディタの目的である．

πエディタではフレームというあらかじめ作成してある標準的なパターンをもとに，プログラムの設計，

† Structure-oriented Display Editor Using Standard Pattern by Yutaka OHFUDE, Toshiyuki OBI (Systems and Software Engineering Division, Toshiba Corporation) and Kazuo AIDA (Information Systems Laboratory, Research and Development Center, Toshiba Corporation).

†† (株)東芝システム・ソフトウェア技術推進部

††† (株)東芝総合研究所情報システム研究所

```
[program design]
  ├program name  preprocessor
  ├main module
  │  ├name        main
  │  └details
  ├used module
  │  ├module name/name in instantiation  gtable/idtable
  │  └parameter in instantiation
  │        tsize=100
  │        contype=string
  ├used module
  │  ├module name/name in instantiation  string/string
  ⋮  ⋮
```

図-1 [program design] フレームのインスタンス
[ ] はフレーム名

```
[program design]
  program name    preprocessor
  main module
    name          main
    details
      [module]
        name
        description part
        execution part                    [process] フレームのインスタンス
          [process]
            purpose  initialize and read the first token
            statements
          [while]
            loop condition  token≠EOF
            purpose  until reading EOF, do preprocess
            statements
              [process]
                purpose  insert a pair of replaced token
                         and replacing token in idtable if
                         macro definition
                statements
              [process]
                purpose  replace token of macro expansion
                statements
                                  [while] フレームのインスタンス
```

図-2 main module の details 部分の展開

コーディングと同時にドキュメントの作成も行う.

### 2.1 フレーム

図-1 はフレームの例を示す. ゴチック体の部分は, [program design] フレームを示し, イタリック体の部分はユーザが入力する部分を示す. 図の左側の縦横の線はフレームの要素（フィールドと呼ぶ）間の関係を示す. フレームに必要な情報が入力されたものをインスタンスと呼ぶ. 図-1 の main module の detailes 部分を展開したものが図-2 である. ここでは [module] というフレームが使用されているが, このようにフレームのインスタンスの詳細を表わすのに他のフレームを用いることができる. このように出来る構造を P-tree (Program-tree) と呼ぶ.

この例でも分る通り, π エディタではプログラム作成方法として, 段階的詳細化を前提としており, 各段階の詳細化を行う支援として, あらかじめ作成されたフレームを用いることにしている. 最上位の抽象レベルの記述から, 段々と具体レベルに詳細化され, 図-2 の statements レベルでは具体的なプログラムが記述されることになる.

フレームを用いることにより, (1)関連する情報は同時に保持できる, (2)必要な情報はユーザに入力を強制することになり欠除することはなくなる, という利点が生じる.

### 2.2 π エディタの機能

π エディタの機能として次のものがある.

(1) フレームからインスタンスの実現
(2) P-tree の特定位置へのインスタンスの関係づけ
(3) P-tree 構造の変更
(4) P-tree 構造の移動
(5) フィールドへの文字列の挿入
(6) 入力のチェック

エディタ設計の目標として, (1)ユーザに使い易いこと, (2)フレームの追加や修正ができること, および (3)エディタそのものに拡張性をもたせること, を挙げているが, ここでは P-tree 構造の処理についての例を見てみることにする.

図-3(a)は [parameterized module] インスタンスの通常の表示モードを示す. コマンド ZOOM-OUT により, そのとき編集ポインタの示しているインスタンス全体を図-3(b)のように非表示の状態にする. 通常のモードに戻すには ZOOM-IN コマンドを用いる. インスタンス全体でなく, 特定のフィールドを非表示にするのに HIDE コマンドを用いる. 図-3(c)は HIDE コマンドにより module parameter フィールドが非表示の状態になっていることを表わす. 逆にこのフィールドを通常の表示モードに戻すのには SHOW コマンドを用いる.

P-tree 構造の表示/非表示だけでなく, 編集ポインタの親子兄弟ノードへの移動コマンド, ノード要素単位の削除や転写コマンドなどがあり, 構造全体を編集単位とすることが可能になっている.

```
[parameterized module]
  name/full name          gtable/generic table
  author                  Y. Nakamoto
  date (month/day/year)   1/10/80
⇒module parameter
    numeral parameter
      name/type/meaning   tsize/integer/size of gtable tsize>0
    type parameter
      name                conttype
      associated operation
      note                This designates type of content in gtable
    terms or notation for explanation
```

(a) [parameterized module] インスタンスの通常の表示モード

```
[parameterized module]   gtable
```

(b) ZOOM-OUT の結果
通常の表示へは ZOOM-IN で戻れる.

```
[parameterized module]
  name/fullname           gtable/generic table
  author                  Y. Nakamoto
  date (month/day/year)   1/10/80
⇒/\term or notation for explanation
    term/meaning          entry/item name entered in gtable
    term/meaning          gcontent/content corresponding to an enty
    summary
      This module manages a table: gtable. The operation "enter"
      inserts a pair of entry and gcontent into the table. The operation
      "search" searches the gcontent for the entry in the table.
    exceptions
```

(c) HIDE の結果
SHOW コマンドで "module parameter" フィールドは再表示できる.
⇒ 編集ポインタを示す.
/\ 非表示のフィールドがあることを示す.

図-3 表示モードの例

π エディタはデータ・ゼネラル社の汎用シニコン NOVA-3 (主記憶 192K バイト, ディスクメモリ 20 M バイト) 上に開発されている. 現在も応答性, 操作性の改善, 機能の拡張を目的として開発が続けられている.

## 3. SPISE

ソフトウェアの生産性を向上させる命題に対して多くの努力がなされているが, ソフトウェアの再利用は, この命題に対する解決策の1つである. 再利用の対象は単にプログラムコードだけでなく付随した文書 (仕様書, 設計書) を含める. 再利用の単位を部品と呼ぶことにすると, 次のように分類できる.

(1) ブラックボックス部品：機能とインタフェース条件のみ知って利用するもの

プログラム： ID

```
プログラム                    表題
--------------------------
概要説明 ---@
見出し部 ---@
環境部   ---@                 内容
データ部 ---@
手続き部 ---@
```

(a)

手続き部：

```
手続き部
--------------------------
procedure division.
実行制御 ---@
機能パターン ---@
内部パターン ---@
例外処理 ---@
```

(b)

図-4 スケルトン例

(2) ホワイトボックス部品：機能だけでなく, その内容を知り, 一部変更あるいは穴うめ方式で利用するもの

前者の分類にあたるものとして, ライブラリ・ルーチン, パッケージ・プログラム等があり, 通常関数呼び出しや CALL 文で利用される. 後者の例として, 設計方法を含んだフォームシート, プログラム全体のパターン, プログラムレベルの制御構造等がある. SPISE はエディタ環境のもとで後者の支援を目的として開発された.

### 3.1 プログラム構造のフレーム表現

SPISE では再利用部品のことをスケルトンと呼ぶ. 図-4 はスケルトンの簡単な例を示す. 図-4(a)ではプログラムの全体構成を表わすパターンをスケルトンとして作成している. 図-4(b)はその一部「手続き部」を表わすスケルトンである. 図で, @の付く行は下位の詳細を表わすスケルトンがすでに存在することを示す. 図-5 はこれらのスケルトンを用いて, 作られたプログラムを示している. 図の中で, @, &, および % の記号があるが, それぞれ

@：下位の詳細を表わすスケルトンが存在すること

&：フレームの表題行といい, その行の表わす抽象表現の詳細化がすでになされていること

%：下位のスケルトンは存在しないが詳細化が必要

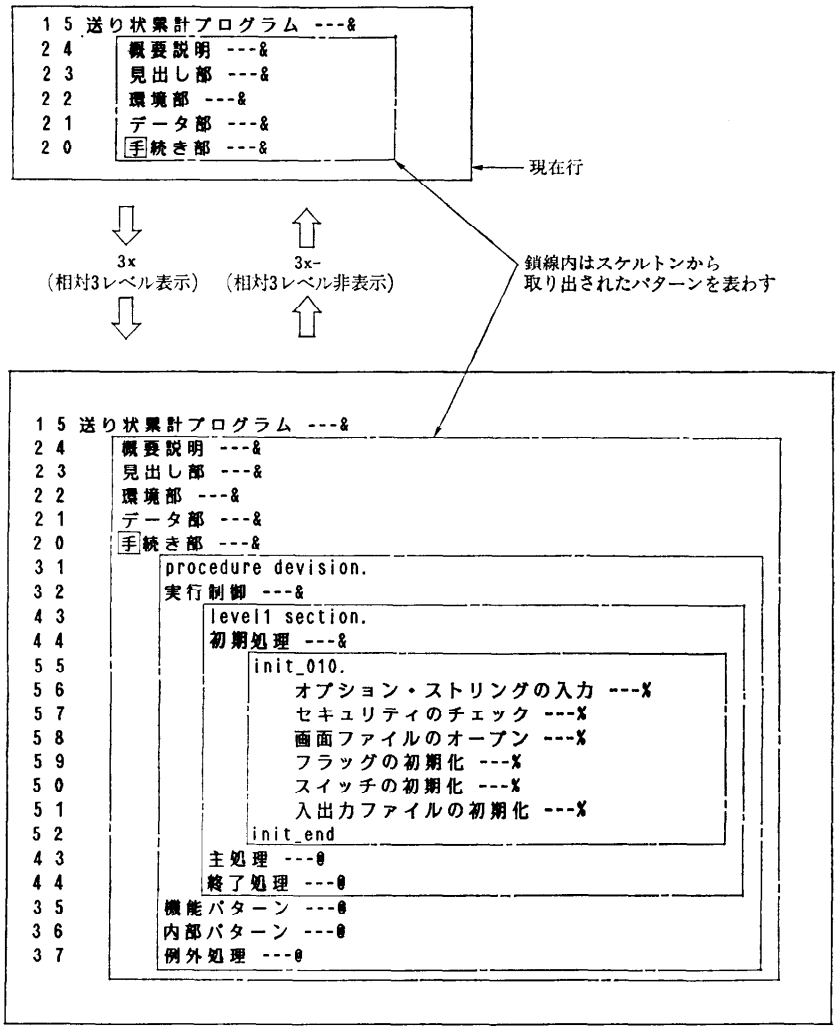



図-5　フレームの表示/非表示の例

であること
を示す．% 行に対する詳細化は通常の編集機能を用いて行うことになる．

SPISE ではスケルトンを用いて実現されたプログラムの構成要素のことをフレームと呼ぶ．図-5 で分るようにプログラムはフレームを要素とする木構造として作成する．(π エディタのフレームおよびインスタンスが，SPISE ではそれぞれスケルトンおよびフレームに対応する．)

スケルトンは，ID，表題および内容部よりなる．表題はスケルトンが使われたときフレームの表題行になる部分であるが，適切な名前に変更することができる．図-4(a)の「プログラム」という表題が「送り状累計プログラム」という表題に変えられている．

例でも分るように，SPISE ではプログラムの設計コーディング手法として段階的詳細化の手法を前提としている．各段階の詳細化を支援するのに，あらかじめ与えられたスケルトンを用意しておき，それらをエディタ環境のもとで検索させ，組み立てていく．

### 3.2 SPISE の機能

SPISE の主な機能として次のものがある．

(1)　スケルトンの管理
(2)　CRT 画面の管理
(3)　フレームの編集
(4)　レポートの出力
(5)　文字列の編集 (ED レベル)

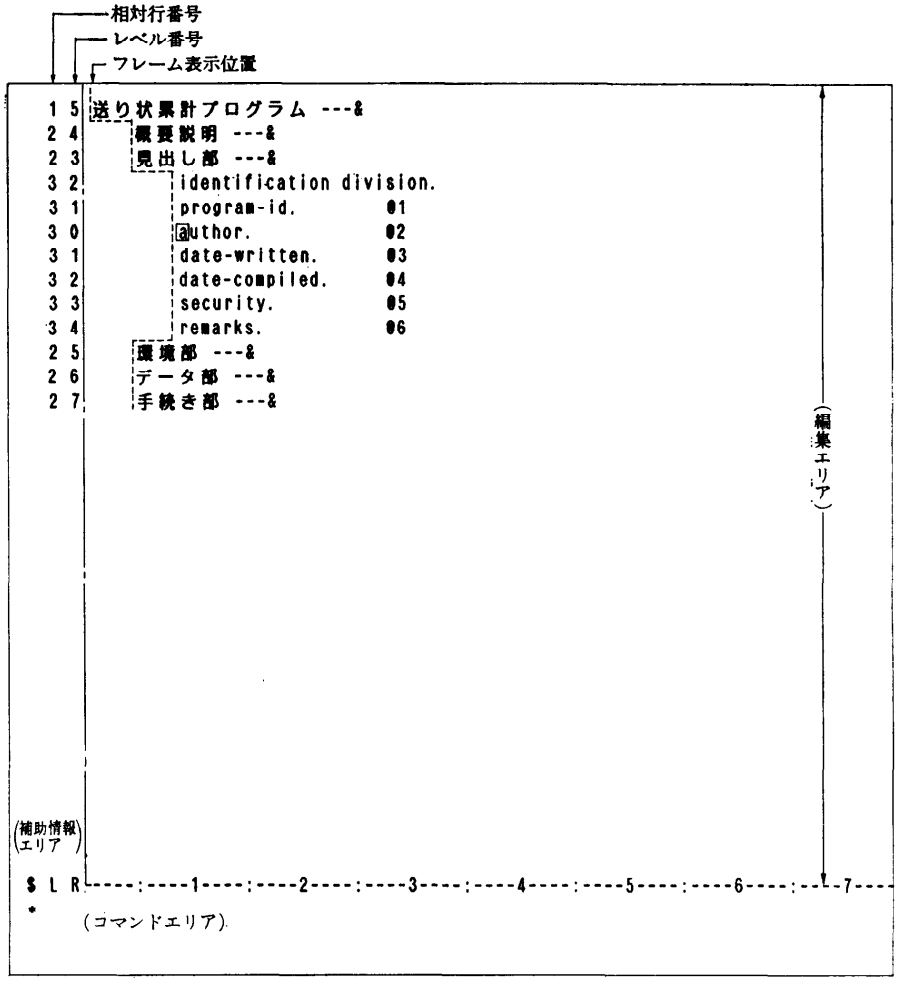



図-6 SPISE の CRT 画面

ここでは特徴的なことについていくつか説明する.

図-6 に CRT 画面の一例を示す. 画面を上下2部分に分割し, 上方を編集対象文字列の表示エリア, 下方をコマンドの入力エリアとしている. 画面左端には, フレーム構造のレベル番号を表示することができる.

フレーム構造はπエディタの P-tree と同様木構造になっており(図-5), 木構造を扱う機能がある. CRT 画面の表示行数には限りがあり, プログラム全体を表示対象とすると常に部分的な表示しかできなく, プログラム全体の構造や, 全体の中での位置づけが把握できない. これを解決するため, 図-5 に示すように, コマンドによりプログラムの概要のみを表示したり, 逆に具体的な詳細部分を表示することが可能である.

図-7 は SPISE で扱う主なデータの流れを表わしている. スケルトンの作成は編集プログラムで行うが, その管理は別のプログラムで行うことになっている. SPISE で扱うデータは内部的には木構造をしており, 直接コンパイラにかけることはできないのでコンパイル用に通常の文字列に変換できるようになっている. また普通のエディタで作成されたプログラムをSPISEで編集することも可能である. この場合まず構造を持たない全体を構造を持たない1レベルであるような1つのフレームとみなし, 意味のある単位ごとに抽象的な表題をつけフレームの構造として作ることが可能である. この機能を用いるとトップダウンに詳細化を行うだけでなく, ボトムアップに構造化することも可能となる.

SPISE は，UNIX マシン UX-300 上に実装されている．分散処理プロセッサの応用ソフト開発用として移植を行う予定である．

## 4. おわりに

本稿では標準的なパターンを利用したプログラム開発を支援する構造エディタ π と SPISE の紹介を行った．両者に共通する主な特徴は

（1） 標準的なパターンを利用してプログラムを構築すること．

（2） プログラムドキュメントとコードを同一ファイルに保存蓄積できること．

（3） 内部的には木構造を持ち，構造要素に対する操作が可能であること．

（4） CRT 画面の制約を除くため，ツリー構造を利用して全体構造の表示や，全体での位置づけを明らかにしながら詳細部の表示を可能にすることがある．

エディタはマンマシンインタフェースとして操作性が重要な要素となる．本稿で解説した2つのエディタは通常のキャラクタ・ディスプレイを用いることを前提として設計されているが，たとえば，マルチスクリーン機能を持つ高機能ワークステーションに同じ目的のエディタを開発するとより操作性に優れたものとなろう．

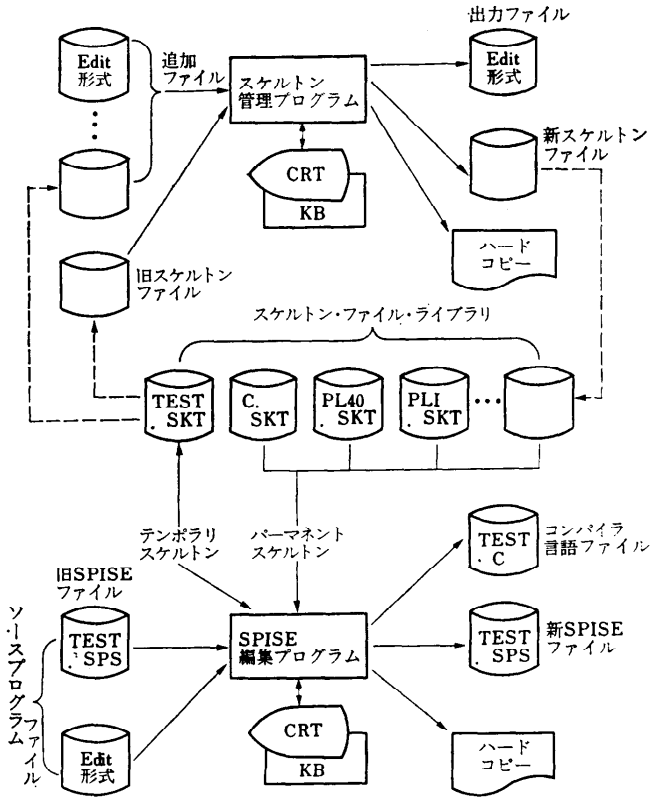



図-7 SPISE のデータフロー

## 参考文献


1) Koehler, Clark: The UCSD Pascal Handbook, PRENTICE-HALL (1982).
2) Waren, Teitelman: Interlisp Reference Manual, Xerox Palo Alto Research Center (1978).
3) Tim, Teitelbaum and Reps, Thomas: The Cornell Program Synthesizer: A Syntax-Directed Programming Environment, Comm. ACM, Vol. 24, No. 9, pp. 563-573 (1981).
4) Feiler, P. H. and Medina-Mora, R.: An Incremental Programming Environment, Proc. 5 th Int. Conf. on Software Eng., pp. 44-53 (1981).
5) Petrone, L., Leva, A. D. and Sirovich, F.: DUAL: An Interactive Tool for Developing Documented Programs by Step-wise Refinements, Proc. 6 th Int. Conf. on Software Eng., pp. 350-357 (1982).
6) 車谷博之，中本幸一，萩原俊郎，都倉信樹：プログラム作成支援システム π の構造エディタ，情報処理学会ソフトウェア工学研究会資料，17-8, pp. 29-32.
7) Nakamoto Yukikazu, Iwamoto Tadahiro, Hori Masato, Hagihara Kenichi and Tokura Nobuki: An Editor for Documentation in π-System to Support Software Development and Maintenance, Proc. 6th Int. Conf. on Software Eng., pp. 330-339 (1982).
8) 会田一夫，大筆 豊，小尾俊之：標準パターンによるプログラミング支援ツール：SPISE-II，情報処理学会第26回全国大会講演論文集，pp. 613-614 (1983).
9) 会田一夫，大筆 豊，小尾俊之：標準パターンによるプログラミング支援ツール：SPISE-II，情報処理学会ソフトウェア工学研究会資料，33-3 (1983).


（昭和59年4月12日受付）