ソフトウエア工学 34－8
（1984. 2. 9）

言語適応型プログラミング環境用マンマシンインタフェースとしての構造エディタPARSE*)

田中 厚　中所 武司
（日立製作所システム開発研究所）
高橋 勇喜　森 清三
（同左　大みか工場）

1．はじめに

マイクロコンピュータをはじめとする計算機の応用分野拡大に伴い、ソフトウェアの生産性と信頼性の向上は、ますます重要な課題となってきている。1970年代には、ソフトウェア工学という新しい研究分野が確立され、ソフトウェア開発工程全般にわたる方法論、技法、ツールの開発が行われてきた。中でも、構造化プログラミングに代表されるモジュール設計やプログラミング技法の進歩は著しく、その支援言語[1,2]も数多く開発され実用に付されてきた。

しかしながら、ソフトウェア開発費用全体に比べてこれらの言語の寄与部分には限度があることから、最近では、言語周辺のツールを含めた統合プログラミング環境[3,4]が重要視されている。

われわれは、統合プログラミング環境開発の基本方針として、各ツールの有機的結合による統合化と対象言語適応による高機能化を掲げ、これを「言語適応型プログラミング環境」[5,6]と名付けた。そして、まず、そのユーザインタフェースとなる構造エディタPARSE[7]を開発することにした。これは、従来のテキストエディタがプログラムを単なる文字列とみなした編集操作を行うのに対し、プログラムの文法的構造を基本とする編集操作を行うもので、信頼性の高いプログラムを効率よく開発することができる。

本文では、言語適応型プログラミング環境の設計思想と、構造エディタPARSEの機能、実現方式について述べる。

2．言語適応型プログラミング環境の設計思想

2．1　基本方針

従来のプログラミング環境においては、エディタ、コンパイラ、テストデバッガなどの基本ツールの他に、ライブラリ管理や文書化ツール、各種解析ツールなどがあるが、ユーザからみて次のような問題点があった。

(問題1)　多くのツールは、プログラミング言語に依存しないようにして汎用性を持たせているため、機能的に低水準である。

(問題2)　各ツールは個別に開発され、相互独立性が強いため、入出力仕様の違い、機能の重複、欠除などの不都合が生じ、複数ツールを組み合わせて使用することが難しい。

そこで、この問題に対処するため、次の基本方針が有効であると考えた。

(方針1)　各ツールの言語適応化による高機能化

(方針2)　各ツールの有機的結合による統合化

方針1は、ツールの機能を、それが対象とするプログラムを記述したプログラミング言語に適したものにすることを意味する。たとえば、先に述べた基本ツールとして、構造エディタ、構造化プログラミング言語コンパイラ、構造テストツールを備える。また、方針2は、プログラミング方法論、プログラム解析技法、データベース、ユーザインタフェースの共有化によって実現する。

2．2　統合化の条件

方針2の有機的結合は、次のように実現する。

(1) プログラミング方法論の共有

プログラミングの主要な作業である設計、作成、検証の間に一貫性を持たせるため、各作業の支援ツールに同一の方法論を反映させる。

(2) プログラム解析技法の共有

特定言語向きツールは、プログラム解析処理を伴うが、ツール間で共通な部分も多い。例えば、構文解析はコンパイラの他にも、構造エディタやテスト網羅率計測用コード埋め込みツールで必要となる。そこで、構文解析やフロー解析などの処理を共有化することにより、各ツールの高機能化が可能になるほか、対象とする言語の仕様がツール間で異なることも防止できる。

(3) データの共有

プログラムや各ツールの入出力データなどをデータベース化し、それらの間の関係を保持することにより、ツール間インタフェースの不一致を防止できるほか、プログラム変更に伴う再コンパイルや再テストの自動化、あるいは関連データの問い合わせなどの機能が実現できる。

(4) ユーザインタフェースの共有

ツール毎にコマンド言語や端末操作法の体系が異なっていてはシステムとして使いにくい。ユーザインタフェースの一様化は統合化の必須条件である。

2．3　システム構成

以上の方針に基づく言語適応型プログラミング環境のシステム構成を図1に示す。図の(1) はソフト

*PARSE Production And Reduction Screen Editor

ウェア構成で、前節(2) の条件に従い、プログラム解析系をバックエンドプロセサとする。一方、フロントエンドプロセサが与えるユーザインタフェースは、事実上、会話処理の多い構造エディタによって決まる。図の(2) は、ソフトウェア構造を階層的データ抽象構造で表現したものである。

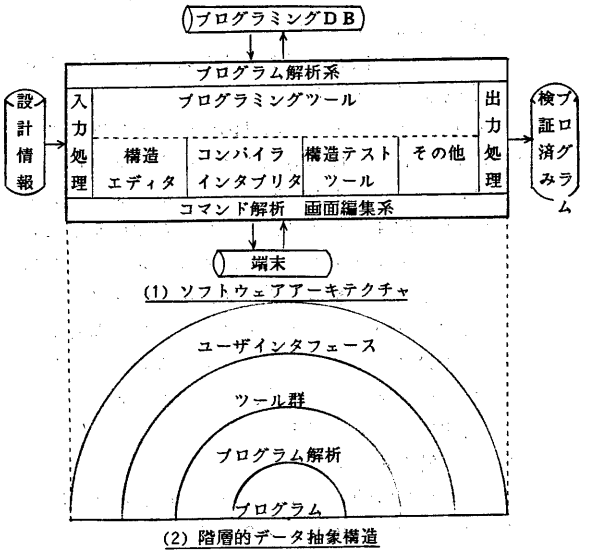

図1　言語適応型プログラミング環境の構成

2．4　基本機能

言語適応型プログラミング環境の基本機能は、表1の9モードに分けられ、ソフトウェア開発工程に従って概ねこの順序で使用される。

表1　言語適応型プログラミング環境の機能

NO.	モード名	モード名称	機能
1	Opening	開始モード	システム使用時の開始処理
2	Production	詳細化モード	文法誘導方式による プログラムの段階的詳細化
3	Quary	問い合せモード	コードレビュー用 プログラム解析情報の表示
4	Reduction	修正モード	プログラムの変更
5	Semantic analysis	意味解析モード	プログラムの意味解析
6	Testing	検証モード	プログラムの検証
7	Utility	ユーティリティモード	既存ツールの利用など
8	Validation	品質評価モード	品質の総合的評価
9	Writing	文書出力モード	各種ドキュメントの出力

3．ユーザインタフェースの設定基準

2章の統合化の条件で述べたユーザインタフェースは、プログラミング環境の使い勝手の善し悪しを大きく左右する。そこで、ここでは、特にマンマシン性について述べる。

「使い易さ」は、システムがユーザの行動を予測し、その行動を支配する要因を制御することにより実現できると考えられる。

ここで、ユーザの行動を支配する要因として、次のものが挙げられる。

(1) 仕事の目標：何をしたいか、結果的に何を得たいか

(2) 仕事の構造：その仕事がどのような要素から成り立っているか

(3) 仕事の手順：その仕事を遂行する手順をどのように組むか

(4) 仕事の処理能力：ある手順で仕事を遂行する場合、その手順をこなす能力があるか

このうち、(1)と(2)は遂行すべき仕事の性質、(3)と(4) はその仕事を遂行するユーザの性質に依存する。すなわち、仕事が決まれば(1)と(2)は一意に定まる。これに対して、(3)と(4)は言語適応型プログラミング環境が初心者から熟練者までを対象とするため、個々のレベルに応じた制御方法が必要になる。

これらの要因を制御するためには、システムは次の条件を満たす必要があると考えられる。

(1) 機能性：一般的に、ユーザが遂行しようとする仕事の目標および構造と、システムの持つ個々の機能との間にはギャップのあることが多い。たとえば、仕事の目標が、「条件文（ＩＦ～ＥＮＤ）を削除する」のに対して、テキストエディタの機能では、「”ＩＦ”のある行から対応する”ＥＮＤ”のある行までを削除する」と操作しなければならない。このため、ユーザは、このギャップを埋めながら仕事を進めなければならず、大きな負担となる。

(2) 習得容易性：初心者であればあるほどシステムの使用法を習得するには手間が掛かる。現状のように分厚いマニュアルを通読しなければ使えないシステムでは、作業の立ち上げに必要以上の労力を要するばかりでなく利用者離れの原因となる。

(3) 柔軟性：熟練者であればあるほど、それまでの経験や知識の蓄積から使い勝手に関する本質的な不満を訴えるようになる。例えば、ユーザがシステムの持つ機能を最大限に活用し、個々の操作を最小の時間や手順で行おうとすれば、操作がマクロ化できるような機能が必要になる。

(4) 対話性：人間同士の会話では、聞き手が「わからない」と言わない限り、話し手は相手に正しく伝わっているものと考えるが、コンピュータ相手の場合には誤解が生じやすい。そのため、ユーザの意図が、簡単正確にシステムに伝わり、システムからの応答がユーザに簡単に理解できることが必要である。

尚、(1) が機能性に関する設定基準であるのに対して、(2)、(3)、(4) は操作性に関するものである。

4．構造エディタＰＡＲＳＥの基本機能

4．1　開発目的

まず、機能性の観点から述べる。従来のテキスト

エディタはプログラムを単なる文字列とみなし、文字単位あるいはまとまった文字列からなる行単位のテキスト編集を行うものであった。このように、プログラマはプログラムを操作対象とする時はそれを文字と見る一方、思考対象とする時はそれを意味ある文章と見る。即ち、書くレベルと読むレベルの間に大きなギャップがあり、これが、単にプログラミング効率の低下ばかりでなく、誤り発生の原因になっている。

そこで、構造エディタＰＡＲＳＥは、このような従来のエディタに関する意味的ギャップを除き、操作対象のレベルを思考対象のレベルにまで引き上げることを目的とする。また、ＰＡＲＳＥの対象言語ＳＰＬ[6),8),9)]は構造化プログラミングを支援しているが、方法論を反映したプログラミング環境の構築という基本方針に沿って、ＰＡＲＳＥも構造化プログラミングを支援するものとする。

次に、操作性の観点から述べる。最近のようなソフトウェア適用範囲の拡大に伴い、経験の浅いプログラマの比率が増加している。そのため、プログラミング環境中のツールは、誰でも簡単に使えることが必要である。そこで、初心者にはツールの操作法を誘導し、熟練者にはその仕事に応じた柔軟なユーザインタフェースを与えることにする。

このような目的から、ＰＡＲＳＥには次のような機能を持たせた。

(1) 段階的詳細化機能
(2) 構造化コーディング機能
(3) 構文要素単位の編集機能
(4) プログラム構造の解析機能
(5) 構文要素単位の画面編集機能
(6) コマンド入力の誘導機能
(7) 階層的ヘルプ機能
(8) システム状態表示機能

このように、構造エディタＰＡＲＳＥは、２章で述べた言語適応型プログラミング環境の９モードのうち、開始モード（Ｏモード）から修正モード（Ｒモード）までを支援する。

４．２　段階的詳細化機能

Ｅ．Ｗ．Ｄijkstra 教授の提唱する「一度に一つの決定」の原則に沿った段階的詳細化法[10)]によるプログラミングを支援する。そのため、ＰＡＲＳＥが扱うプログラムに次のような詳細化未定義部分(Refinable) を導入する。

- ＳＰＬ文法を表すＢＮＦ規則に用いた非終端記号
  （例：<STATEMENT>）
- ＳＰＬの文の代わりになる擬似文
  （例：”データ　ヲ　スタック　ニ　ツム”）
- ＳＰＬの手続き参照
  （例：”スタック(Ｓ)ニ　データ(Ａ)ヲツム”）
- ＳＰＬの新データ型参照
  （例：ＶＡＲ　Ｓ：ＳＴＡＣＫ）

これら４項目のうち、後の２項目はＳＰＬ自身が有する段階的詳細化支援機能である。第２項は第３項の手続き参照に相当するが、構文規則に制約されないでコメント風に記述できるようにするため導入した。また、第１項は、文法規則に沿った詳細化を促すために導入した。

(1) プログラム機能の段階的詳細化機能

まず、処理概要を擬似文で記述した場合、この擬似文の詳細化を要求すると、それがコメントに変更され、<STATEMENT> の入力要求状態になる。一方、処理概要を手続き参照形式で記述した場合、詳細化要求すれば擬似文と同様にインライン展開形式になり、定義要求すればその手続き本体の枠組み（テンプレート）が表示され、手続き定義の要求状態になる。

次に、データ構造の詳細化機能として、ユーザ定義データ型参照部分に対して詳細化要求すれば、型名がコメントに変更され、データ型の入力要求状態になる。

(2) 構文要素の段階的詳細化機能

構文要素の段階的詳細化では、Refinable(詳細化未定義部分）を逐次表示しその詳細化を促す。現在詳細化要求中のRefinable を、Current Refinable (以下CRと略す)と呼ぶ。このとき、可能な詳細化の形式の一覧を表示するが、その中から一つを選択する方式と、対応するテキストを直接入力する方式を設けた。そして、Refinableを、

- プログラムの基本構造を表すもの
  (例:<MODULE>,<PROCEDURE>)
- 上記項目の一部分で、複数のシラブルから構成されるもの
  (例:<STATEMENT>,<DCL>,<EXPRESSION>)
- 個々のシラブル（例:<IDENTIFIER>,<CONSTANT>)

に分け、第１項は選択のみ、第３項はテキスト入力のみ、第２項は両方式可能とした。

尚、作成済みのソースプログラムをＰＡＲＳＥで編集できるようにするため、上述のような端末からの入力のかわりにファイルから入力する一括入力機能を導入した。

４．３　構造化コーディング機能

ＳＰＬでは、無条件分岐を排し、基本制御構造を選択文、反復文、複合文(これらを基本制御文と呼ぶ）に限定している。そこでＰＡＲＳＥでもこのような構造化コーディングを支援するため、入力したい基本制御文のメニュー選択あるいは専用コマンド（テンプレートコマンド）の入力により、そのテンプレートを自動生成して表示するようにした。一般の文は<STATEMENT>がCRの時にはテキスト入力可能

*SPL Software Production Language

としているが、基本制御文はブロック構造を有するため、テキスト入力を許さず、常にテンプレート経由で詳細化させることにした。

4．4　構文要素単位の編集機能

プログラムの修正は、プログラム入力の場合と同じく、構文要素単位に行えるようにして、修正時のエラー誘発を防止する。修正としては、挿入、削除、置換、移動、探索が基本であり、次のような機能を設けた。

(1) 挿入は、まず、該当部分にカーソルを移動後、挿入コマンドによって、カーソル位置に挿入可能な非終端記号を挿入する。その後、プログラム入力時と同じ方法で挿入したいテキストに置換する。尚、挿入コマンドのオペランドに、挿入したいテキストを直接記述すれば、挿入操作は1ストロークで行える。

(2) 削除は、該当部分にカーソル移動後、削除コマンドによって行う。尚、還元コマンドを用いれば、カーソル位置のテキストに対応する非終端記号に置換した後、その省略コマンドを投入することにより削除できる。

(3) 置換は、削除の場合と同じ方法で該当部分を非終端記号に置換した後、プログラム入力時の方法で新しいテキストに置換する。

(4) 移動は次のように行う。ＰＡＲＳＥは、プログラムを格納する領域を二つ持つ。一つは、言語仕様の定義の開始記号<PROGRAM>から始まり、文法的に常に正しいプログラムを格納する領域（プログラム領域）で、一つは、プログラムの一部分を格納する領域（ワーク領域）である。そして、まず、移動すべき部分をワーク領域へ退避し、つぎに、それをプログラム領域の挿入すべき部分に挿入コマンドで移す。

(5) 探索は、探索コマンドを用いて構文要素単位に行う。探索方向としては前後両方向が可能である。

このように、プログラムの修正は、すべて非終端記号を経由して行うようにし、一般のテキストエディタのような、文字列操作を禁止した。

編集の例を図2に示す。

4．5　プログラム構造の解析機能

プログラムの机上テストやレビューを支援するため、次の機能を導入した。

(1) ＳＰＬのモジュール階層図の表示

(2) ＳＰＬの手続き呼び出し関係図の表示

(3) 手続き参照の位置に、手続き本体の表示

従来コンパイルリスト上でこれらの情報を参照するには、分厚いリストをたぐる必要があり手間が掛ったが、本機能により、会話型による効率的なコードレビューが可能になる。

4．6　構文要素単位の画面編集機能

プログラムの言語構造に適した画面表示／編集機能として、次の機能を設けた。

(1) 自動インデンテーションによるソースプログラム表示機能（プリティプリンティング）

(2) モジュールおよび手続き単位の画面スクロール機能

(3) 構文要素単位のCR移動機能

4．7　コマンド入力の誘導機能

エディタの操作を簡単にするため、次のようなコマンド構文誘導方式を導入した。

(a) 入力可能なコマンド名あるいはオペランドの一覧をメニューとして表示する。

(b) メニューの選択は、メニューと１：１に対応したキーボード上のファンクションキーの押下によるワンタッチ入力で行う。

(c) 選択されたコマンド名あるいはオペランドは、逐次画面上に表示する。

(d) 一コマンドの誘導が完了すると、自動的にコマンド実行処理に移る。

一般的にメニュー選択方式では、選択項目が多い場合にそれらを一度に表示しきれないという問題がある。そこで、選択項目はグループ分けし、そのグ

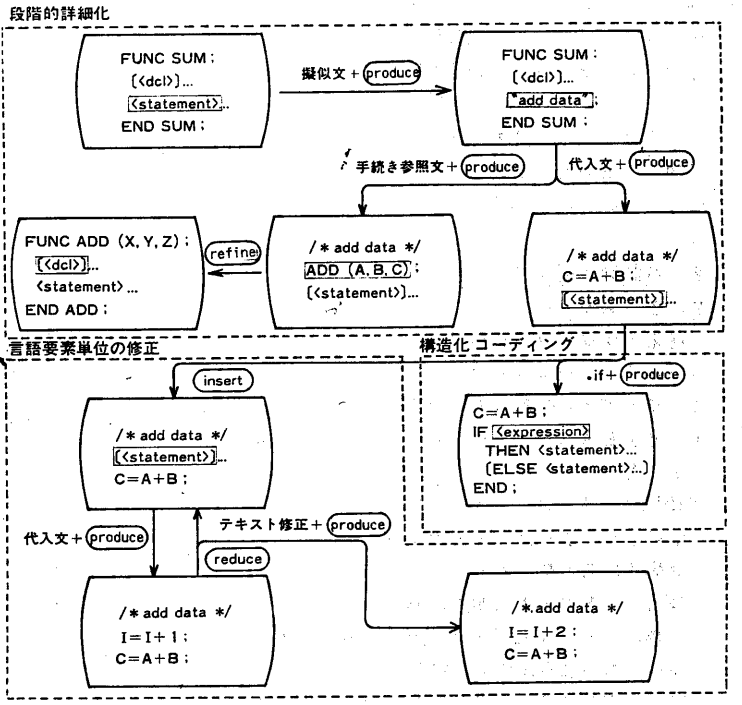

図2　ＰＡＲＳＥを用いたプログラム編集例

ループ単位に表示するようにした。そしてメニューをスクロールするコマンドを導入することにより、この問題点を解決した。また、コマンド構文の誘導途中でその入力を中止するコマンドを導入した。さらに、テキストで直接入力しなければならないオペランドについてはその旨を示す表示を行う。

尚、メニュー選択方式は慣れるに従って煩わしくなる傾向があるため、コマンドをテキストで直接入力する従来方式も併用する。

4．8　階層的ヘルプ機能

プログラムやコマンドは、その構文の誘導だけでは意味がわからないので、次に示す方式で意味的ガイドを行うヘルプ機能を導入した。

(a) ヘルプコマンドを設け、その入力でヘルプ状態にする。

(b) このとき表示されているメニューの中の一項目が選択されると、そのヘルプメッセージを出力し、ヘルプ状態を終了する。

(c) ヘルプメッセージは、意味の説明を中心とした自然語で表現する。

尚、ヘルプメッセージは、対象言語やコマンドの構文要素と１：１に対応付けているので、構文誘導によるプログラムやコマンドの入力では、ユーザに対してトップダウンなガイドができる。

4．9　システム状態表示機能

ユーザがシステムの状態を見失わないようにするため、コマンド実行のたびに、画面のスクロール単位、プログラム入力モード、現在実行中のコマンド、対象言語の構文要素の入力属性などのシステム状態を表示する。

4．10　その他

(1) 直前のプログラム操作の取り消し

(2) プログラムのファイルへの退避

(3) システムの終了

尚、表２にＰＡＲＳＥの機能一覧を示す。

5．ＰＡＲＳＥにおける対象言語の定義機能

構造エディタＰＡＲＳＥにおけるユーザインタフェースの柔軟性を向上させるため、次に示す対象言語の仕様は、定義系で定義し、ＰＡＲＳＥが起動されるたびにそれらを読み込んで処理する方式を採用した。

(1) ＢＮＦ記法による対象言語の構文規則の定義

左辺の非終端記号の詳細を、非終端記号および終端記号を用いて右辺に記す。

(2) 構文要素の段階的詳細化における"段階"の定義

一つの構文要素の詳細化は、ＢＮＦの左辺をその右辺で置き換える操作一回に対応させる。

(3) 構文要素の選択メニューの定義

構文要素の詳細化可能形式は、現在詳細化中の構文要素に対応するＢＮＦの右辺とする。

表２　ＰＡＲＳＥの機能一覧

分類	コマンド	機能
プログラム生成	PRODUCE	構文要素の詳細化
	REFINE	関数、新データ型の定義
	OMIT	省略可能構文要素の省略
	INPUT	プログラム一括入力機能の起動
	COMMENT	コメントの入力
	SETCM	コンパイルモード指定文の入力
プログラム修正	INSERT	構文要素単位の挿入
	REDUCE	構文要素単位の還元
	DELETE	構文要素単位の削除
	FIND	指定文字列の探索
	TRIM	構文要素のワーク領域への移動
	LISTTR	TRIMされたサブトリーの表示
プログラム解析	LISTMT	モジュール階層図の出力
	LISTFT	関数呼び出し関係図の出力
	INLINE	関数参照文の位置に定義部の表示
コマンド入力誘導	HELP	ヘルプメッセージの出力
	CANCEL	コマンド入力の途中取り消し
	SCROLL-MENU	コマンドメニューのスクロール
画面制御	SCROLL-SCREEN	画面のスクロール
	MOVE-CR	CURRENT REFINABLE の移動
システム制御	SAVE	プログラムのファイルへの退避
	UNDO	直前操作の取り消し
	END	システムの終了

(4) 基本制御文のテンプレートの定義

選択文、反復文、複合文のテンプレートは、それらの構文要素に対応するＢＮＦの右辺とする。

(5) 構文要素の入力属性の定義

ＢＮＦ左辺の構文要素の可能入力属性、すなわち、直接テキスト入力、メニュー選択入力、両者併用入力のいずれかを、構文要素ごとに指定する。

(6) 構文要素のヘルプメッセージの定義

ヘルプコマンド投入時に表示すべきメッセージを、構文要素ごとに指定する。

(7) ソースプログラムのインデンテーション形式の定義

ＢＮＦ右辺の構文要素に対し、改行と欄下げ指定を用いて、プリティプリントの形式を定義する。

尚、対象言語ＳＰＬの仕様の定義では、ＰＡＲＳＥの操作性を向上させるため、次のような工夫をしている。

(1) 構文要素の段階的詳細化のための手順数を減らすため、ＢＮＦ左辺の個数を少なくする。すなわち、右辺にできるだけ多くの構文要素を集める。

(2) 詳細化可能な構文要素をユーザに直感的にわからせるため、ＢＮＦでは再帰的表現を避ける。すなわち、できるだけ繰り返しや省略可能表現を用いる。

図３に、ＳＰＬの定義の例を示す。

```
N  <PROGRAM> ::= ?0 <MODULE>...

N  <MODULE> ::= <ENV_MODULE> ;
              | <PROC_MODULE> ;

N  <ENV_MODULE> ::= ENVIRONMENT <MODULE_NAME>
                      [(<MODULE_NAME>)];
                      [?2 <SPEC_UNIT>;]
                       ?2 <DCL_UNIT>...;
                   ?0 END [<MODULE_NAME>]

N  <PROC_MODULE>::= PROCESS <MODULE_NAME>
                      [(<MODULE_NAME>)];
                      [?2 <SPEC_UNIT>;]
                      ?2 <FUNC_UNIT>...
                   ?0 END <MODULE_NAME>

N  <SPEC_UNIT> ::= SPECIFICATION:
                     ?2 <FUNC/NEW_TYPE>...
                  ?0 END

N  <FUNC/NEW_TYPE> ::= FUNCTION <FUNC_HEADING>
                         [:<TYPE>]
                         [OPTIONS(<FUNC_OPTION>)];
                     | TYPE <NEW_TYPE_HEADING>;

N  <FUNC_HEADING> ::= <FUNC_NAME>
                      [<FUNC_RESTRICTION>]

N  <NEW_TYPE_HEADING> ::= <NEW_TYPE_NAME>
                            [<NEW_TYPE_RESTRICTION>]

N  <DCL_UNIT> ::= DECLARATION
                    [OPTIONS(<MEMORY_OPTION>)];
                     ?2 <DCL>...
                  ?0 END;
```

注) N、NT、T:入力属性、?n:段付け指示

図3　ＳＰＬの定義例

6．構造エディタＰＡＲＳＥの実現方式

6．1　システム構成

構造エディタＰＡＲＳＥのシステム構成を図4に示す。

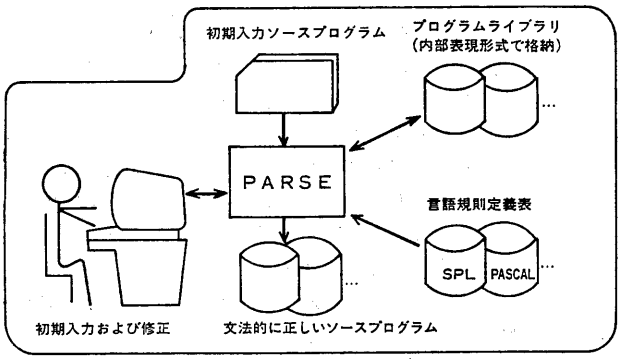

図4　ＰＡＲＳＥのシステム構成

6．2　処理方式

(1) 導出木によるプログラムの表現

プログラム編集機能を効率的に実現するため、プログラムは、ＢＮＦからの導出過程を示す導出木で表現している。

(2) 文パーサ

直接入力可能な構文要素に対して入力されたテキストを、ＢＮＦから得られた表を駆動して構文解析し、導出木を作成する。

(3) アンパーサ

インデンテーション情報に従い、導出木をソースプログラムに変換して、表示する。

7．むすび

以上、言語適応型プログラミング環境の設計思想と、そのマンマシンインタフェースとなる構造エディタＰＡＲＳＥについて述べた。特に、本文で述べた構造エディタＰＡＲＳＥの使い勝手の向上条件は、エディタだけに限らず、会話型システム一般に適用できるものと考える。

最後に、本研究の機会を与えて頂いた日立製作所大みか工場 久保 岳明氏、ならびに、有益な御討論を頂いた同工場　林 利弘氏に感謝いたします。

参考文献

1) B.H.Liskov,et al.:Abstraction Mechanisms in CLU,Comm.,ACM,Vol.20,No.8,pp.564-576 (Aug.1977)

2) Proceedings of ACM-SIGPLAN Symposium on the Ada Programming Language,SIGPLAN.Notices,Vol.15,No.11 (1980)

3) T.Teitelbaum et al.:The Cornell Program Synthesizer:a Syntax-directed Programming Environment,CACM,Vol.24,No.9,pp.563-573 (Sep.1981)

4) P.Medina-Mora et al.:An Incremental Programming Environment,IEEE Trans.Software Eng.,Vol.SE-7,No.5,pp.472-482 (Sep.1981)

5) T.Chusho et al.:A Language-adaptive Programming Environment Based on a Program Analyzer and a Structure Editor,9th World Computer Congress,IFIP'83

6) 中所　他：言語適応型プログラミング環境の設計思想、情報処理学会第27回全国大会、PP.525-526 (昭58-10)

7) 田中　他：言語適応型プログラミング環境における構造エディタPARSEの機能と実現方式、同上　PP.527-528

8) 林　他：制御用トップダウンストラクチャードプログラミング言語-SPL-、日立評論、Vol.60,No.3,pp.59-64 (1978)

9) 中所　他：段階的詳細化、データ抽象化を支援する言語SPLのコンパイル技法、情報処理学会論文誌、Vol.21,No.3,pp.223-229 (昭55-5)

10) E.W.Dijkstra:Notes on Structured Programming:Structured Programming,Academic Press,London and New York,pp.1-82 (1972)