ICOT Technical Memorandum: TM-0639

TM-0639

ESPを用いて開発した
汎用構造エディタSEMACS

吉武淳(三菱電機),
梶山拓哉(アーティフィシャル・
インテリジェンス), 小久保岩生(三菱総研),
藤田正幸(三菱総研), 坂井公, 横田一正

December, 1988

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

ソフトウェア工学 63 - 3
(1 9 8 8. 1 1. 2 5)

ESP を用いて開発した汎用構造エディタSEMACS

吉武 淳
三菱電機㈱コンピュータ製作所

小久保 岩生　　藤田 正幸
㈱三菱総合研究所

梶山 拓哉
㈱アーティフィッシャル・インテリジェンス

坂井 公　　横田 一正
財新世代コンピュータ技術開発機構

逐次型推論マシンPSI 上に構造エディタSEMACS(Syntax-directed Extension of pMACS)を開発した。その特長は、

(1) 編集対象となる構造の文法をBNF で記述できるという意味で汎用性を持つ
(2) ユーザの好み、習熟度に応じた使い方ができる
(3) 効率の良い構文解析手法を採用している
(4) 単独でも、他のシステム開発環境の１要素としても使うことができる

などである。これらの特長の実現には、オブジェクト指向の機能を持った論理型言語ESP(Extended Self-contained Prolog)の特性も大きく寄与している。

Grammar-independent Structure Editor SEMACS developed by using ESP

Jun YOSHITAKE
Computer Works, Mitsubishi Electric Corporation
325, Kamimachiya, Kamakura, Kanagawa, 247 Japan

Iwao KOKUBO　　Masayuki FUJITA
Mitsubishi Research Institute,Inc.
2-3-6, Otemachi, Chiyoda, Tokyo, 100 Japan

Takuya KAJIYAMA
Artificial Intelligence Corporation
Bunzanshinsen Blg, 10-10, shinsen, Shibuya, Tokyo, 150 Japan

Ko SAKAI　　Kazumasa YOKOTA
Institute for New Generation Computer Technology
Mitakokusai Blg, 1-4-28, Mita, Minato, Tokyo, 108 Japan

The structure editor SEMACS(Syntax-directed Extension of pMACS) has been developed on PSI(Personal Sequential Inference Machine). It has the following features:

(1) Structures can be edited according to its grammar represented in BNF.
(2) The way to use it is changeable according to the user's taste or skill.
(3) It has adopted an efficient parsing method.
(4) It can be used solely, or as a component of a system development environment.

To realize these features, the characteristics of the programming language ESP(Extended Self-contained Prolog) are effective. The ESP language is an extension of Prolog with object oriented functions.

1　はじめに

財新世代コンピュータ技術開発機構(ICOT)では第５世代コンピュータシステム研究の一環として定理証明支援システム(CAP)[9] と分散知識ベース管理システム(Kappa)[8][15]を開発している。それぞれのプロジェクトにおいて、構造に依存した操作によりデータの編集を効率良く行うことが強く求められている。CAP では数式や論理式の編集、Kappa ではテーブルのスキーマの編集などである。そこでそのためのツールとして、構造エディタSEMACS(Syntax-directed Extension of pMACS)を両者共同で開発した。

一般に、ソフトウエアシステムのうち、ユーザによるテキストの入力・修正作業が多いものは、その作業を支援する高度なユーザインタフェースが必要となる。文字列を編集の対象とするテキストエディタに対して、（ある文法に従った言語で表現される）構造を持ったデータを編集の対象とするエディタは構造エディタ[1][2]と呼ばれ、そのようなユーザインタフェースを実現する有効な手段となる。テキストエディタと比較した場合の構造エディタの長所としては、

① 編集コマンドの高度化
② 対象システムとの有機的結合

などがある。

しかし、そのような長所があるにもかかわらず、テキストエディタに比べて広く使用されている構造エディタは少ない。その理由としては次のことが考えられる。

① 言語、システムへの依存性・・・対象とする言語またはシステムに対する依存性が高い。そのため、他の言語やシステムで直接利用したりその技術を移転したりすることが難しい。また、対象言語自身の変更にも弱くなる。
② 操作性の悪さ・・・文法に無関係に自由な記述が行えるテキストエディタに比べ操作上の制約が多く、行いたい操作が円滑に行えない。

構造エディタSEMACSはこれらのことを考慮し、複数のシステムで利用できる使いやすいインタフェースの核となることを目的として開発された。以下、本論文では、２節でまずSEMACSの機能概要について説明する。そして、３節で上述の問題点がSEMACSではどのように解決され、ソフトウエアシステム開発上のどのような特長につながっているかを述べる。４節ではSEMACSのもう１つの特徴として、開発に使用した言語ESP(Extended Self-contained Prolog) が、SEMACSそのものとその開発にどのように役立ったかについて論じる。そして、５節でまとめを述べる。

2　機能概要

2．1　文法定義と表示法

(1) 文法定義、構文解析

SEMACSで使用する文法は、利用者がBNF 風の形式で自由に定義することができる。図2.1-1 に例を示す。

```
program ::= list(statement,";").
statement ::= "if",test,"then",statement,["else",statement] |
              "while",test,"do",statement                   |
              t(name),":=",expr                             |
              "(",program,")"                               |
              "write",t(name)                               |
              "read",t(name).
test ::= expr,c-op,expr.
expr ::= expr,op2,expr1 | expr1.
expr1 ::= expr1,op1,expr0 | expr0.
expr0 ::= "(",expr,")" | t(name) | t(integer).
c-op ::= "=" | ")" | "(".
op2 ::= "+" | "-".
op1 ::= "*" | "/".
```

図2.1-1 文法定義の例

この定義のうち、""で囲まれたものとt() で示されたものは終端記号で、それ以外は非終端記号である。t(name) は任意の識別子、t(integer)は任意の整数をそれぞれ表す。list(...) はSEMACSの文法で繰返しを示す特殊な表現である。たとえば、１行目は、「programは";" で区切られたstatement の１回以上の繰返しである」ことを示している。

このような形式で記述された文法を入力して、システムは構文解析と文法案内に必要なデータを生成する。構文解析は、左決定木による汎用パーサ[10]を使用している。

(2) 表示法、プリティプリント

まず構文木とその表示の例を図2.1-2 に示す（文法は図2.1-1 の例のもの）。

=で囲まれた部分はエリア（編集対象部分を示すもので、テキストエディタのカーソルに相当する）を表すものとする。後述するようにSEMACSには文字列モードと構造モードの２つがあるが、構造モード時の表示は、構文木の葉の部分をプリティプリントすることで実現している。エリアは構文木中の１つの部分木であり、白黒反転して表示される。後述するようにSEMACSは文法に従ってデータをトップダウンに作成していく機能を持つが、非終端記号でまだそれ以下の部分木が確定していないものは、

$symbol　（symbolは部分木のトップの非終端記号）
というように表示される。またエリアの部分木の根となっている非終端記号（図2.1-2 の例ではtest）をエリアのトップカテゴリと呼ぶ。

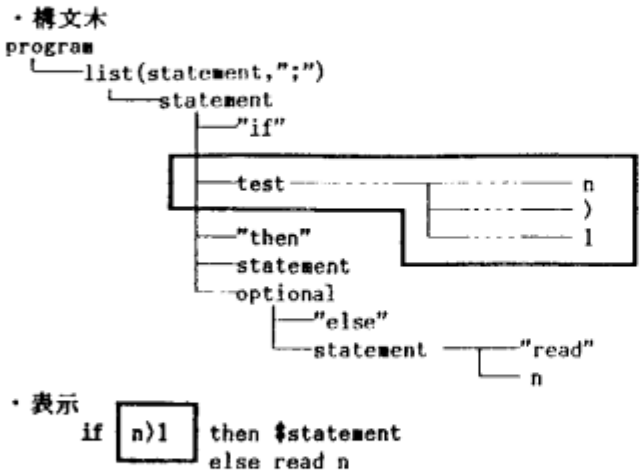


図2.1-2 構文木とその表示の例

プリティプリントに必要な、改行・字下げなどの情報も、構文解析に必要な文法情報と同様にユーザが自由に定義できる。

2．2　構造編集コマンド

SEMACSには、通常のテキストエディタのように文字列を編集の対象とする文字列モードと構文木を直接編集の対象とする構造モードの２つがある。文字列モードではPSI 上のテキストエディタであるPmacs のほとんどすべての編集コマンドを使用することができる。ここでは構造モード時に使用できる特徴的なコマンドを説明する。

(1) モードの切換え

次の４つのコマンドがある。

・構造-)文字列
（エリアを文字列編集の対象とする）　・・・①

・構造-)文字列（エリアの内容を削除）　・・・②

・文字列-)構造
（エリアの内容を構文解析）・・・③

・文字列-)構造
（前回の構造モード時のエリアに復帰）　・・・④

例

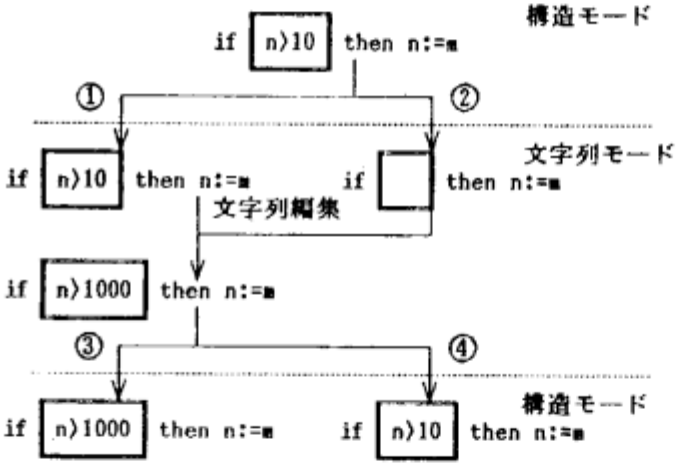


図2.2-1 モードの切換えコマンド

文字列モード時には、エリア以外の部分もエリア内部と同様に編集することができるので、その部分のテキストを参照したり複写したりしてエリア内部のテキストを編集することができる。ただし、エリア以外の部分の変更を行っても、再び構造モードに戻る時には無視され、前の構造モードの時のイメージが復元される。

(2) 木表示、文法案内

SEMACSには、ICOTで開発されているエキスパートシステム開発支援ツールProton[11] の木表示モジュールに、SEMACS用の機能を加えて拡張した木表示モジュールがある。SEMACSの通常の表示と違い、構文木をそのままの形でグラフィカルに表示する。また、以下に述べる文法案内機能と組み合わせて使用することにより、構文木をこの画面上で展開することができる。

文法案内は、エリアのトップカテゴリの展開の可能性をメニューウインドウの形式で表示し、ユーザの選択により展開を行い、結果を示す機能である。図2.2-2 に例を示す。

またマウスの別のクリックでは、実際の展開は行わず、詳細な情報を見ることができる。図2.2-2 の例では、test,statementに関するウインドウが順に表示されるが、そのウインドウ上でいずれかの項目がクリックされたときには再帰的にその項目に関する情報の処理を行う。

(3) エリアの移動

エリアの移動を行うコマンドには、親、子、兄弟など現在の部分木との構文木中の相対的な位置関係を指定する方法と、画面中で、移動させたい場所を直接マウスにより指定する方法の２通りを用意している。

(4) 探索・置換

各種のコマンドが用意されているが、図2.2-3 に「現在のエリアのトップカテゴリと同じトップカテゴリを構文木の前方に探索し、エリアをその内容に置き換える」という処理を行うコマンドを連続

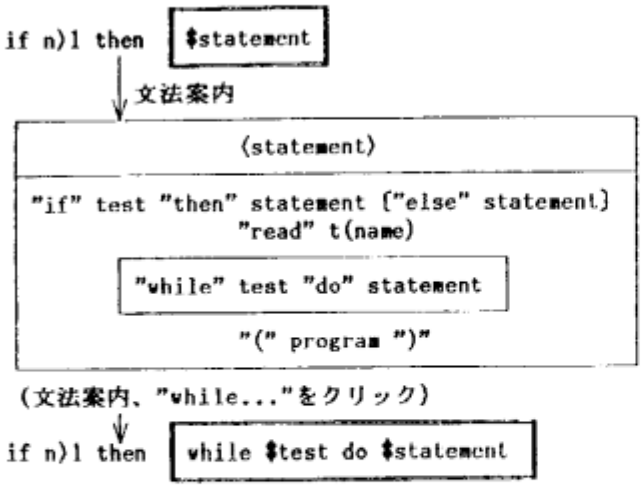
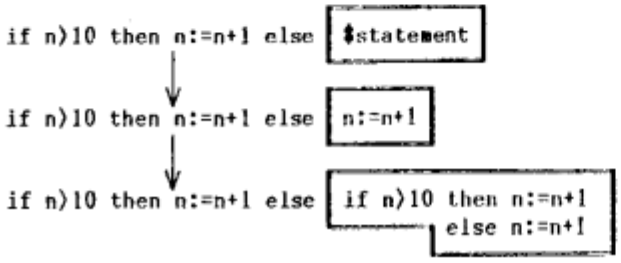


図2.2-2 文法案内の例



(前の文全体のトップカテゴリもstatement)

図2.2-3 置換の例

して用いた例を示す。

(5) その他のコマンド

以上の他にも次のようなコマンドがある。

・エリアの削除、復元
・リスト要素の挿入、削除
・構文木のセーブ、ロード
・ホロフラスティング

3 SEMACSの特長

本節では、1節で指摘した

① 言語、システムへの依存性
② 操作性の悪さ

という従来の構造エディタの問題点がSEMACSにおいてどのように解決され、その結果SEMACSがソフトウエアシステムの開発上どのような特長を持っているかを述べる。

3．1 文法に対する汎用性

文法（が定義されたファイルなど）を入力として、その文法を使用して構文解析を行うパーサを出力するツールは、パーサジェネレータと呼ばれる。SEMACSの文法に対する汎用性は、おもにこのパーサジェネレータにより実現されているということができる。文法に対する汎用性には、まったく別の文法に対してもシステムを同様に使用できるということ以外に、1つの文法に対しても、その文法に変更があった時に容易に対応できるという長所がある。すなわち文法の記述のみを変更すればよいため、言語を開発している場合など、対象としている言語に変更が多い時には便利である。実際、CAP プロジェクトでは証明記述用の言語PDL(Proof Description Language) に対し、その記述力、記述し易さなどを検証する必要があるため細かい実験的な変更を繰り返したが、パーサジェネレータなしでは行えないことであった。

パーサジェネレータの性能として、パーサを生成する時間と出力されたパーサの構文解析に要する時間が重要であり、実用的には後者が特に重要となる。この構文解析の問題については4節でさらにふれる。

3．2 習熟度に応じた使用

SEMACSはテキストの入力方式として2つの使い方ができる。その2つの使い方とは、

1 文字列モードでの入力を中心として、ときおり構文解析をかけて構造モードへ移る使い方
2 構造モードで文法案内を行ってはその未展開の部分をうめていくような使い方

である。これら2つの使い方ができることのメリットを以下に考察する。

まず、1 の使い方のメリットを考える。一般に、その文法を熟知している上級者は、すでに従来のテキストエディタでの編集操作にかなり慣れ親しんでいるので、それと同じ要領で入力し必要に応じて構文解析を行ってその文法的な正しさを確認する使い方の方が好まれる。また、上級者には、表示法をエディタに縛られたくない、場合によっては一時的に文法的な正しさを破った記述をしておきたい、という要求もある。SEMACSを1 の使い方で使えるということは、このような上級者の要求に合致するものといえる。

次に、2 の使い方のメリットを考える。一般に、初級者は文法を熟知していないので、エディタ側にガイドしてもらった方が良いということがある。SEMACSを2 の使い方で使えるということは、このような初級者のニーズに合致するものといえる。

そしてまた、このように初級者、上級者それぞれの要求に合う使い方ができるということは、逆から考えれば、その人の習熟度に応じてSEMACSの使い方を変えていくことができるというメリットにもなるわけである。

3．3 構造編集の容易さ

ここでは、構造モード時のコマンドのおもなものが、ソフトウエアシステムの開発上どのような効果があるか説明する。

(1) 構文案内

この機能は、3.2 節で述べたように初級者に有効なだけでなく、文法に精通したユーザにとっても文法の詳細を忘れた場合などに有効な機能である。実際、CAP プロジェクトで使用している言語PDLは、既述のように完成された固定的な言語というわけではなく、細部に頻繁な変更が行われているが、この構文案内の機能によりユーザがその変更に比較的容易に対応することができた。

(2) 部分パース

構文案内の機能だけによっても、構文木の葉の部分を除けばすべての構文木をトップダウンに作成することができる。しかし、3.2 節で述べた上級者の場合のように、文法に精通している箇所などがあればそこは文字列モードで入力、編集を行った方が効率的であろう。SEMACSでは文字列モードで構文木の任意の部分木に対する内容をテキストで入力することができる。また文字列モードでは、2.2 節で述べたように、エリア以外のテキストも、エリアのテキストとまったく同じように参照することができるので、複写などをすることにより利用することができる。構造モードに復帰するときにシステムは、エリアのテキストのみを構文解析しその部分的な構文木を全体の構文木に埋め込むという操作を行う。このように部分的な構文解析が可能であるため、そのために必要な時間が短縮されることのほか、他のウインドウ（バッファ）で部分的な構文木をいくつか作成しそれらを部品として組み合わせるといった操作も可能となる。

(3) リストについて

2.1 節で述べたように、SEMACSでは繰返し構造に対して特殊な記法を用意している。

```
list(Ele,Del)    ・・・①
list(Ele)        ・・・②
```

図3.3-1 リストの記法

図3.3-1 の例では、① は区切り記号付きの１回以上の繰返し、② は区切り記号なしの１回以上の繰返しをそれぞれ表す。通常のBNF による定義では、繰返しを再帰的に定義したり、区切り記号と繰り返される要素とを１つの構造として定義したりするため、生成される構造が直観に反するものになりがちであるので、特に用意した記法である。

この記法で定義された構造をSEMACSでは特にリスト構造と呼び、その要素に対しては挿入・削除が自由に行えるようにしている。区切り記号付きのリストに関しては、その要素の挿入・削除に際してシステムが自動的に区切り記号をそれぞれ挿入・削除するので、ユーザは気にかける必要がない。

また、リストの構文木そのものも、各要素が対等に並んだものになっている。たとえばPrologのように頭部と尾部とがあり、尾部がさらに頭部と尾部とを持つというような構造になっていない。そして、その表示も要素がそのまま並んで表示される。このような構文木や表示もユーザの直観に合った良いものといえる。

３．４　ソフトウエア開発へのメリット

本節ではプログラム開発にSEMACSを適用した場合のメリットを考察する。

まず仕様書作成にSEMACSを使用すれば、章立てやインデンテーションその他を統一することができる。

実際にプログラムをコーディングする場合のメリットとしては以下のことが考えられる。

① 開発者がより本質的な作業に集中できる。

たとえばPascalのプログラムのifやwhile のような構造上のキーワードは、SEMACSでは文字列モードで編集を行っている時には構造モードへ移る時にチェックされ、構造モードで編集を行っている時にはシステム側から表示されるため、タイプミス等に気を使う必要がない。

② インデンテーションを自動化できる。

インデンテーションは構造モードで自動的に行ってくれる。さらに、そのインデンテーション（プリティプリント）の仕方は、プログラムの形ではあるが、ユーザが決定できるようになっているので便利である。

③ プログラムの書式を統一できる。

文法をBNF で記述できることを生かして、たとえばプログラムのある部分（関数やメソッドの定義部分など）に必ずコメントを記述するような文法を作成してやれば、プログラムの書式の統一まで行うことができる。

④ プログラムの構造化を支援できる。

文法案内とプリティプリントの機能を合わせて使えばプログラムの構造化を支援できる。

さらに、

① SEMACSで作成していた仕様書の詳細処理の部分をバッファの切換えを利用してプログラムのコーディングの方に持ってきて、各部を１つ１つのステップに置き換えていく

② やはりSEMACSで作成していた仕様書のデータ構造を記述している部分をバッファの切換えを利用してプログラムのコーディングの方に持ってきてそのまま使用する

などを行えば、統一したプログラミング環境を構築したり、仕様書とプログラムとの自然な対応付けをしたりすることができる。

3．5　使用例

SEMACSは単独のエディタとしてだけでなく、
①　1節で述べたようにもともと複数のシステムの核となることを目的としていたこと
②　オブジェクト指向型言語で作成されていること
により、容易に他のシステムの開発環境の1要素として組み込まれることができる。本節ではその例としてCAP とKappa で実際に使われている例を示す。

(1) CAP での応用

CAP ではSEMACSはユーザとシステムとのインタフェースの役割を果たす。SEMACSの通常の機能に加えて、証明検査箇所のリアルタイム表示、推論規則を利用した証明の自動展開などのコマンドを付加して使用している。証明の自動展開機能で使用する置換え規則は、PDL の文法に依存していない機能なので他の文法でも使用することができる。現在CAP システムは、証明記述がすべて終了した時点でその内容の妥当性を検査するバッチ的な処理を行っているが、SEMACSの機能を核として対話的な検査を行うシステムへの拡張を予定している。

(2) Kappa での応用[5]

Kappa ではSEMACSはおもに非正規関係などの複雑な構造を持つスキーマなどのメタデータの作成、修正を行うメンテナンスツールに組み込まれて使用されている。意味制約からスキーマの自動設計支援システムやデータベースアクセス用のコマンドを追加した質問応答システムとしての拡張を予定している。

4　ESP による利点

SEMACSの開発に使用した言語ESP は論理型言語Prologにオブジェクト指向の機能を加えたものである。本節ではこの言語の特性がSEMACSそのものとその開発にもたらした利点について述べる。

4．1　論理型言語としての利点

論理型言語としての利点として、本節では、
①　構文解析との関連
②　構造データの扱い
③　その他宣言的な記述ができることなど
について述べる。

4．1．1　文法の汎用性と構文解析の効率化

今回、ユーザ用の文法記述言語として先に述べたようにBNF 風のものを採用し、文法に汎用性を持たせた。一方、そのBNF 風に記述された文法に基づく構造を効率良く構文解析する必要がある。

Prologは一般に構文解析向きの言語と言われ、実際にPrologをもとにしたパーサが研究されてきた。そのなかで、ボトムアップパーサBUP[7] を改良したLDT[10] をSEMACSではパーサジェネレータの出力であるパーサとして採用している。LDT には図4.1.1-1 のような文法をボトムアップに解析する場合、a1 a2 ... anの部分の解析を重複して行わないという効率の良さがある。また、ボトムアップパーサではε-規則が書けないという欠点があるが、{...} という省略可能であることを表す記法を導入したので、ε-規則が書けないことによる不便はほとんど感じなくてすむ。例えば、a ::= b,{c}.という構文規則は内部的に図4.1.1-2 のように展開して処理している。

```
B1 -) a1 a2 ... an T1
B2 -) a1 a2 ... an T2
```

(T1,T2 は終端記号または非終端記号の任意の列)

図4.1.1-1 右辺に重複のある文法の例

```
a ::= b.
a ::= b,c.
```

図4.1.1-2 文法の内部での展開の例

BUP やLDT （あるいはそのもとになった文法記述言語DCG ）は、Prologのような論理型言語を基礎として設計されたものであり、このような書きやすい文法記述言語と効率の良いパーサが容易に得られるという点で論理型言語（ESP ）を採用したことは有利であった。

4．1．2　構造データの扱い

SEMACSの中で構造データが、ESP の論理型言語Prologとしての利点により、扱いやすかった例をいくつか述べる。

まずタームが扱えることにより、たとえば図2.1-1 のようなBNF 風の記述をそのままの形で読み込み、かつ、タームとして処理できた。すなわち、BNF 風の記述を構文解析する機能を開発する必要がなく（省略可能の記法を図4.1.1-2 のように内部的に展開するなどの本質的でない部分を除いて）便利であった。

また、SEMACSでは木構造を内部的にPrologリストで表現している。そしてPrologのリスト処理の機能をフルに生かすことにより、
①　エリアの部分、それ以外の部分を内部的に容易に切り分けて持っていることができる
②　①により、エリアの変更を高速に処理できる
③　木表示での、内部表現のPrologリストの他のデータ構造への変換も効率的に実行できる
などのメリットがあった。

４．１．３　その他宣言的な記述ができることなど

論理型言語による利点として他に挙げられることは、１つ１つの節が宣言的で論理的な意味を持つことがまずある。実際、SEMACSは実質的な開発メンバー５人全員それぞれがフルパワーをさけない状態で、かつ、開発期間も１年に満たないなかで、２節で説明した機能、３節で述べた特長を実現したわけであるが、それには宣言的、論理的な意味を持つことによる、

①　開発者にとっての記述のしやすさ
②　言語自身の記述力のよさ

が要因として挙げられる。またそのような意味を意識しながらインプリメンテーションすることにより、アルゴリズムの誤りを１つの節の論理的な意味のおかしさとして気づくこともできて、信頼性の高いプログラムを作成するのにも役立った。

またLDT 他構文解析の手法や、論理プログラミングについての研究例、プログラミング例が豊富にあったことも、ESP で開発を進めていく環境として非常に良いものであった。

４．２　オブジェクト指向型言語としての利点

オブジェクト指向型言語としての利点として、本節で、

①　内部状態を持てること
②　データ抽象
③　クラスの汎用性
④　生産効率

について述べる。

４．２．１　内部状態を持てること

オブジェクトは内部状態を持っており、あるクラスのインスタンスオブジェクトを複数生成することで同一機能で異なる内部状態のものを簡単に実現することができる。オブジェクト指向型言語のオブジェクトのこの利点をSEMACSではバッファというものに生かしている。

SEMACSのバッファはインスタンスオブジェクトとして実現されており、編集対象の

①　テキスト情報
②　構造情報
③　編集モード
④　文法名

を内部状態として保持している。こうすることにより、

①　バッファを切り換えるだけで、任意の文法に対する構造を編集できる
②　編集モードも含めて、切り換える直前の状態からそのバッファに対する編集操作を再開できる

などのメリットをもたらしている。

４．２．２　データ抽象

４．２．１節で述べたようにSEMACSのバッファは内部状態として構造情報を持っているが、その構造情報も１つのインスタンスオブジェクトになっている。そして、その構造情報を変化させたり参照したりしたいときは、そのインスタンスオブジェクトにメッセージを送ることにより実現している。このようにして、オブジェクトの持つデータ抽象の機能を生かしている。また、「バッファは構造情報を持つ」というようなデータ構造をオブジェクト間のhas_a の関係で人間にとって自然な形で実現しているわけである。

４．２．３　クラスの汎用性

一般に、オブジェクト指向型言語では、新しいオブジェクトのクラスをシステムに加える時も、そのクラスを、既存のクラスに対して使われていたのと同じメッセージパターンによって同様の機能が引き出せるように定義しておけば、もとのシステムを何ら変更・拡張する必要がない。すなわち、オブジェクト指向型言語で書かれたシステムは拡張性に富むわけである。[13]

この長所をSEMACSでは、文法の汎用性や扱える文法の数をいくつでもふやせるという点に生かしている。SEMACSでは、あるバッファで文字列モードから構造モードへ一番最初に移る時に初めてユーザが文法名を指定する。これを受けてSEMACSが構文解析のクラスを初めて指定する。ユーザが記述した文法に対応して生成された構文解析のクラスはすべて同じメッセージパターンで起動されるようになっているので、いつでもその時にユーザが扱いたい文法のクラスを、文法名の指定を通して選べるわけである。SEMACSをインストールした時になかった文法の構文解析のクラスもこのときに呼び出されることができる。また、異なる文法ごとに異なる構文解析のクラスが１つずつ対応していることは、人間にとって自然なソフトウエアの構成と言える。

４．２．４　生産効率

SEMACSは、基本的にPSI 上のテキストエディタであるPmacs をもとにして作られている。本節では、オブジェクト指向型言語の継承機能などを利用した生産効率の良さを示す。

表4.2.4-1 、表4.2.4-2 にSEMACSとPmacs のクラス数、ライン数の対比をそれぞれ示す（カッコ内はProtonの木表示モジュールからの流用分）。

クラスの流用は、クラス名を変更する程度でほぼ100%流用できている。このときPmacs に加えた修

表4.2.4-1 クラス数の対比

修正内容	Pmacs	SEMACS
無修正	81(12)--	-)81(12)
クラス名変更	12 --	-)12
クラス追加	--	64
合計	93(12)	157(12)

表4.2.4-2 ライン数の対比

修正内容	Pmacs	SEMACS
無修正	約14,000(1000)--	-)約14,000(1000)
修正	約 300 --	-)約 300
新規作成	--------	約 7,000
合計	約14,300(1000)	約21,300(1000)

正ライン数は約300 ラインで、これはPmacs 全体の約2%にすぎない。

これに対して、純粋に構造編集機能として加えたクラス数とライン数はそれぞれ64クラス、7,000 ラインである。これは、SEMACS全体の約30% を占める。

このように付け加えた機能の割にPmacs の流用のための修正量が少なくてすんだのは、Pmacs の基本設計の良さとともにESP のオブジェクト指向型言語としての良さによるものと考えられる。

5 結論

構造エディタSEMACSは、種々の従来の構造エディタの問題点を解決しさまざまなソフトウエアシステムの開発効率を向上させるものである。また、その開発に使用した言語ESP の論理型言語としての特性とオブジェクト指向型言語としての特性とを生かしたものである。

今後の課題としては、

① ユーザ用の記述言語をBNF からDCGとし、ユーザによる文法の記述力を増すこと
② ホロフラスティングを自動的に行い、ユーザにとって常に意味のある構造が表示されるようにすること
③ ワイルドカードなどを使用した構造のパターン探索機能の充実
④ 木表示の最適化
⑤ ユーザがプログラムで定義しなくても構造に沿って自動的にインデンテーションを行う、文字列モードのイメージをそのまま使用するなどのプリティプリントの充実

などが考えられる。

謝辞

Pmacs 関係でご助言いただいたICOT第４研究室の佐藤裕幸氏と、Proton関係でお世話になった開発担当者の皆様に感謝致します。

［参考文献］

〔1〕 Bertrand,M.:CEPAGE:Towards Computer-Aided Design of Software,Interactive Software Engineering,Inc,1987.
〔2〕 原田賢一：構造エディタ，共立出版，1987.
〔3〕 ICOT:SIMPOS 文書編集機能説明書,1987.
〔4〕 ICOT:SEMACS 使用説明書,1988.
〔5〕 河村，丸山他：知識ベース管理システムKappa ―利用者インタフェース―，第35回情処全国大会論文集,pp.1615-1616,1987.
〔6〕 小久保，吉武他：逐次型推論マシンPSI 上の汎用構造エディタSEMACS，日本ソフトウェア科学会第５回大会論文集,pp.137-140,1988.
〔7〕 松本，田中：Prologに埋め込まれたbottom up parser:BUP，情処学会自然言語処理研究会34-6,Dec,1982.
〔8〕 溝口，横田他：知識ベース管理システムKappa ―データベースから知識ベースへ―，第35回情処全国大会論文集,pp.1603-1604,1987.
〔9〕 Sakai,K.:Toward Mechanization of Mathematics -Proof Checker and Term Rewriting System-,ICOT Technical Report TR-348,1988.
〔10〕 坂井 公：DCG のボトムアップ型構文解析の新方式について,Proceedings of The Logic Programming Conference'86,pp.13-18,1986.
〔11〕 澤本他：PSI 上のエキスパートシステム開発支援ツール(1)～(4),第33回情処全国大会論文集,pp.1115-1122,1986.
〔12〕 横田一正：知識ベース基本ソフトウェアKappa ，第５回5Gシンポジウム,June,1987.
〔13〕 米澤明憲：オブジェクト指向プログラミングについて，コンピュータソフトウェア,Vol.1,No.1,pp.29-41,Apr,1984.