ソフトウェア工学 51－1
（１９８６．１１．２６）

標準ＭＬを用いたプログラム設計支援環境

大林　正晴

㈱管理工学研究所

標準ＭＬを宣言的仕様記述言語とする視覚的ユーザインタフェースを重視したプログラム設計支援環境を提案する。オブジェクト指向的なプログラム設計方法論の１つであるＤＭＣ（概念によるプログラム設計法）を背景にしたものである。その特徴は、次ぎのとおりである。問題を概念やものを中心に整理し、単語という断片として切出し、その単語を２次元平面上でレイアウトすることにより、視覚的なモデル認識の助けを借りながら概念構造図を作成する。このようにして、大きな問題を小さな概念やものの集りに分割した後は、詳細な仕様の記述を宣言的な記述ができる関数型言語標準ＭＬを用いて記述する。記述単位は、部品化、再利用のためのパラメータ化された汎用モジュールとして扱えるように考慮されている。

A program design environment with the standard ML

Masaharu Ohbayashi
Kanrikogaku,Ltd.
2-2-2,Sotokanda,Chiyoda-ku,Tokyo 101,Japan

In this paper, we propose a new program design environment in which the standard ML is used as a declarative specification language and the visiual user interface is emphasized. The basic ideas come from our proposed DMC (Design Method based on Concept) which is a kind of the object oriented programming methodologies.They have following features.

First, we analyze a given problem on the conceptual or objective standpoint and find the set of words which denote some concepts or objects in the conceptual model of a problem. Secondly, we layout these words on a 2-dimentional virtual grid. We built the conceptual structure chart which describe a visiualize image of the model. After we decompose a large scale problem into small size concepts or objects by the above steps, the details of a specification are written by the declarative functional language, i.e. standard ML.

We also consider the reusability of each component, we can manupulate them as a parameterized generic module.

1．はじめに

ソフトウェア工学では、高信頼度ソフトウェアを得るため、あるいは、より柔軟性のあるソフトウェア開発環境を生み出すための研究が進められている。とくに、近年、抽象データ型やオブジェクト指向プログラミングの研究や知識ベースの活用などの研究が盛んになってきた。また、形式的仕様記述言語として、宣言的言語、論理的言語、関数的言語などが注目を集めつつある。また、より数学的な基盤をもつ高度な形式言語を用いたソフトウェア開発手法が提案されるようになった。

しかしながら、これらのソフトウェア工学の最近の成果を取り入れた新しい方式に基づく設計方法論および開発環境は、まだ確立されていない。

高度なソフトウェアの開発には、特に、問題領域の知識情報をもとに試行錯誤を繰り返しながら最適な解を求めていくような場面では、従来の手続きを主体にしたプログラミング言語や方法では、複雑になりすぎ不十分である。一方、宣言的な言語による仕様記述は、問題を事実や規則の集まりとして簡潔に記述することができるのでそのような問題には、より適している。通常使われている多くの言語がアルゴリズムを手続き的に記述するのに対し、本システムのような形式的仕様記述言語では、非手続き的あるいは、宣言的な記述も可能であり、ある意味ではエンド・ユーザにより近い位置をしめる。本稿で提案するシステムは、そのような高度なソフトウェア開発に適用できる環境である。

2．プログラム設計支援環境

マイクロ・コンピュータなどハードウェアの近年の進歩には、目をみはるものがある。ＩＣ，ＬＳＩチップの製造技術の確立により高度な論理装置を安価に手に入れることができるようになり、ディジタル回路を手軽に製造することが可能になったことによるものである。ハードウェアの回路設計では、目的の回路を実現するのに必要なチップを選択し、互いのピンを配線する。その際、チップの特性や仕様などに関する知識を、データ・ブックなどから得ながら設計を行なう。設計でもう一つ重要なことは、信号のタイミングであるが、これらは、タイミング・チャートとして記述する。大まかなことを言えば、回路設計では、回路図、タイミング・チャート、使用するチップの仕様を決定すればよい。

これに対して、従来のソフトウェア設計は、フローチャートを基本にした考え方でなされることが多々ある。フローチャートの最大の欠点は、プログラムが大きくなると、複雑になりすぎて全体がとらえにくくなることである。これは、モジュールの静的な構造と動的な処理の流れを同じ一つの図にして表わしているところに原因があると考えられる。つまり、回路設計における回路図とタイミング・チャートといったように静的な構造と動的な関係とを分離して説明していないためである。

このような従来のプログラミング設計の欠点を克服するため、プログラムの静的な構造を概念構造図として書き、動的な処理の流れは、概念の仕様として記述するような方法論を導入することにする。ここで、概念構造図が回路設計における回路図にあたる。

実際の設計作業の大半は、試行錯誤と意思決定のプロセスであり、コンピュータを利用した設計、いはゆるＣＡＤが有効であると考えられる。そしてこのようなソフトウェア設計作業のＣＡＤ化においては、ここで説明する概念による設計法が基本的な方法論となりうると考える。

実際のプログラミング環境の構築の仕方には、いろいろなアプローチがあろう。例えば、設計方法論の選択、仕様記述言語の選択、変換方式の選択など、それぞれ異なったアプローチがある。本稿で述べる構想、提案で前提とした、方針はつぎのようである。

・設計方法論を定め、それを土台に環境を構築する。
・形式的仕様記述言語として、宣言的な記述ができるものを選ぶ。
・関数型言語の特長をもったせる。
・視覚的ユーザインタフェースを重視する。

3．設計方法論

プログラム設計方法論には、従来、次のようなものがある。

・ＮＳチャート、ＨＩＰＯ、ジャクソン法、複合設計、ワーニエ法、バックマン図式、ペトリネット、Ｒ－ＮＥＴ、ＨＣＰ、ＰＡＤ

しかし、これらは、抽象化やモジュール化などの考え方を取り入れた方法論ではない。中には、そのような最近のプログラミング技法をある程度考慮したものもあるが、必ずしも十分ではない。

一方、ソフトウェア生産の現場では、巨大化・高度化するソフトウェア製品の品質を向上させ、拡張・保守などの問題に柔軟に対処できるプログラム構造が求められている。このような要請に対し、上にあげた方法をはじめとして、いくつかの方法が提案され効果をあげている。しかし、現実には、これらの方法では、十分に効果をあげることができず、抽象データ型やオブジェクト指向の考えが必要不可欠となる場面も多い。このような場合に有効な手段となるものと考えたのが、ここで提案する”概念による設計法（ＤＭＣ：Design Method based on Concept）”である。

3．1 ＜概念＞と＜もの＞

まず、プログラムを設計する際の指針として新しく＜概念＞という考え方を導入する。＜概念＞とは、一般に、ある性質を所有する一つ一つの具体的な＜もの＞ではなく、その性質だけを所有する抽象的な＜もの＞を指す。もちろん、見方を変えると、この抽象的な＜もの＞、つまり＜概念＞も、一つの具体的な＜もの＞と見なせるわけだから、＜概念＞と＜もの＞とは、相対的な関係にあり、＜もの＞があれば、それに対応する、＜概念＞が必ずあると考えられる。逆に、＜概念＞から具体的な＜もの＞を何個でも生み出すことができるものと考える。

実際の問題から＜概念＞を発見する際には、よく知っている概念だけですませられるとは限らない。そのときには、まず、より具体的な＜もの＞を登場させ、その上で問題を分析し、試行錯誤しながら概念を定めていくことになるだろう。

3．2 モデル化

ところで、＜概念＞と＜もの＞だけでは、問題の構造を表現することはできない。問題の構造をとらえるために＜概念＞や＜もの＞の間にある相互関係を定める必要がある。つまり、＜概念＞や＜もの＞のもつ機能や作用の構造を明確にすることによって与えられた問題の一つのモデルが作られる。人によっていろいろなモデル化が可能なことはいうまでもないが、重要なことは問題の中にある＜概念＞あるいは＜もの＞を中心にモデルを組み立てることである。

一般に、問題をいくつかの＜概念＞の集まりとしてとらえる。そして各＜概念＞は、メッセージを受け付けると、その機能を実現するのである。また、＜概念＞には、その概念に固有のデータ型があることがある。例えば、＜人＞という概念には、［人］と［人の集合］というデータ型が、＜建物＞には、［建物］というデータ型がそれぞれ付随していると考える（［．．．］はデータ型を表わす）。

3．3 概念構造図

次に、モデルを＜概念＞間の構造図として表わす記法を説明する。一つの＜概念＞は、矩形の中にその概念を識別する名前を書いて表わす。図のように、概念＜Ａ＞が概念＜Ｂ＞にメッセージｆを送ることを矢印で示し、そのメッセージを添える（このとき、メッセージｆは、概念＜Ｂ＞に付随するという）。ｆ以外に、例えば、概念＜Ｂ＞のメッセージｇを使う場合には原則として矢印を別に書くが、煩雑になるときは省略して同じ矢印にメッセージを列記してもよい。

＜概念＞の構造図は、互いに＜概念＞間のメッセージのやりとりだけを表現したもの

で、実際に行なわれる演算の順序などに関しては、何も示してはいない。

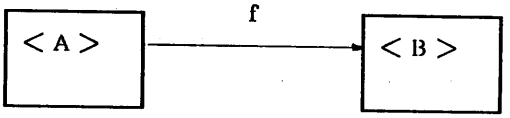


図１．　概念構造図の表現

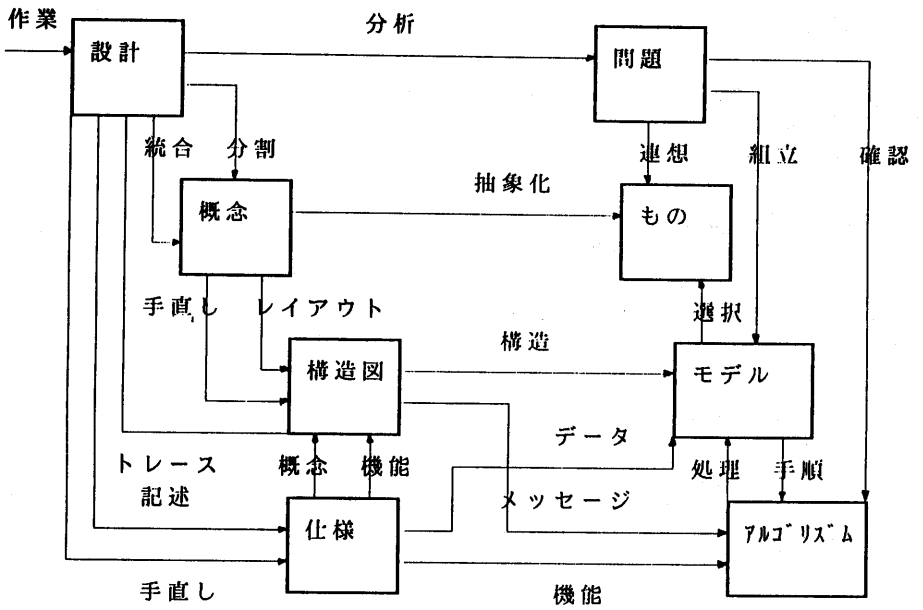


図２．概念構造図の例

３．４　＜概念＞の仕様

概念の構造図は、モデルを明示するために作ったもので、設計者の正確な意図はわからない。そこで、構造図と対応した個々の概念の仕様を形式的仕様として書くことにする。＜概念＞に付随した各メッセージのパラメータとして受け渡される入力データおよび返される出力データを決定し、そのメッセージの機能などと共に宣言的に記述する。

３．５　設計手順

プログラムの設計をどのような手順で進めるべきかについて説明する（図２、および５節を参照）。

① まず、問題を具体的な＜もの＞に即して整理する。当然のことながら、最初から抽象的な概念を見つけるのは困難なので、初めは、具体的なものを登場させて分析し、モデルを作成する。問題の処理過程を検討しながら、入力データや処理アルゴリズムを確認する。（単語の分類、整理）

② 分析・整理した結果、登場した＜もの＞を抽象化して＜概念＞として識別し、全体をいくつかの＜概念＞の集まりに分割する。（具現値のレイアウト）

③ ②で定義した＜概念＞上で処理アルゴリズムが実現されることを確認する。

④ ＜概念＞間の相互関係を考えながら、各＜概念＞に必要な機能およびその機能の実

行を指示するメッセージを決定する。（その名前を具現値表のｖａｌにセット）

⑤ 各＜概念＞に固有のデータがあれば、データ型として定義する。（その名前を具現値表のｔｙｐｅにセット）

⑥ 各機能の入力データ、出力データなどの詳細を決め、＜概念＞ごとに仕様を記述する。（構文誘導エディタで具現値表の記号群、モジュールの詳細を記述）

⑦ 似た＜概念＞や本質的に同じ＜概念＞、一部だけ異なった＜概念＞などを整理・統合する。（記号群名、モジュール名を決定）

4．宣言的仕様記述言語

4．1 核言語

形式的仕様記述の核言語として、将来の変換系や解析系などを考慮して、宣言的な記述ができ、関数型言語の特長をもったものを採用することにした。標準ＭＬは、その様な言語の１つで、ラムダ計算や多様型構造などの研究を背景にして生まれた言語である。

多様型は、型変数を伴った型、つまり、型図式を許す型構造である。型は、仕様記述に対して一種の正しさの検証条件を与えているとも考えられるが、できるだけ簡便であることが望ましい。たとえば、文脈から決定できる型については宣言しないで済ませたい。そこで組織的な多様型をＭＬは採用している。つまり、関数は多様型をもつものとし、引数の型、関数の値の型が文脈から一意的に決定できるときに限り単一型をもつものとする。

標準ＭＬ自体は、モジュール化機構を持たない平坦な言語である。しかし、より大規模な問題を記述するためには、モジュール化機構を付加する必要があることは明らかである。つぎに、いかに自然にＭＬを拡張し、概念による設計方法論との対応をとるかについて述べる。

4．2 モジュール化機構

大雑把に言えば、＜概念＞や＜もの＞は環境（型環境、値環境、例外環境からなる）を定義するための宣言（型束縛、値束縛、例外束縛）の集りに名前をつけたものであり、宣言の評価とは、環境を生成することである。

＜概念＞をパラメータ化されたモジュールとして捉らえたい。そのためには、もっと厳密に対応を定義する必要がある。ここで、つぎのように用語を定義する。

用語	特定の対象（名）	意味
＜もの＞	具現値（名）	モジュールにより具現化された環境
＜概念＞	モジュール（名）	環境を引数にし、環境を値とする関数
記号群	記号型（名）	環境の外部に見えている記号の集合（環境の型）
宣言群		新しい環境を作る束縛の集合

＜概念＞モジュールの具現化で得られた環境に名前をつけることによって、同じ環境をいくつもの場所で使用でき、互いに共有することもできる。＜概念＞モジュールは、環境をパラメータ化してあり、部品化や再利用の便宜を提供する。

環境構造としての＜もの＞具現値は、記号群と呼ぶ一種の型を持つ。基本的には、記号群は＜もの＞具現値を作る各束縛（型構成子、値や例外の型、継承する具現値の記号群）に対して環境の型仕様を与えるものである。一般化すれば、具現値そのものは、型の一般形式、つまり、抽象型とみなすことができる。ここで、型はその値を操作する演算と例外とで１つのものにまとめられている。

5．視覚的ユーザインタフェース

現在、概念による設計方法論を支援する環境をワークステーション上に試作中である。

ここでは、視覚的ユーザインタフェースについて概要を述べる。

5．1 単語の分類、整理

ある程度問題が机上で、分析整理されている場合には、その分析の結果判明した種々の＜もの＞、＜概念＞、＜機能＞、＜データ型＞などを単語で表現したものを順に入力させる。まだ、十分に問題が整理されていなければ、想起した対象を取り敢えず単語で表現していくことになる（不要なものは後で消す）。つぎに、このように、枚挙した単語をモデルを構成して行くときの用途別（具現値名、データ型名、値変数名など）に分類、整理しておく。

実際には、図３の単語表をマウスとポップアップメニューを使って操作しながら、大雑把な単語の分類を行なう。このような単語表を各ライブラリ（問題の単位）ごとにもつことにする。この単語表は、以下に説明する、具現値のレイアウトや定義、詳細な仕様の定義などを行なう際に参照する問題固有の単語集（単語の供給源）の役割を果たす。単語自体を操作（ピックやコピーなど）の対象とすることにより、同じ単語を重複して入力する手間を省いている。

5．2 具現値のレイアウト

一般に、下位の具現値には、２つの種類がある。第１は、望まれる環境を実現するために内部的に参照（使用）するユーティリティ的な役割を果たすものである。これは、モジュールの結果の記号群には影響を与えない、その具現値のユーザにとっては関係ないものである。第２は、その下位の具現値をより増大させた環境にするようなときに使用する。つまり、下位具現値を継承して、その具現値のユーザに対しても継承した記号群を見せる場合である。

このように具現値間の継承と参照の関係は、従属関係の階層を定義するものである。いくつかの具現値は、同じ下位具現値の上に作られる場合もあり得るので、この階層の形式は、木構造と言うよりは、有向グラフになる。具現値間で、下位具現値を共有すると言うことは、二義的なことではなく、共通の下位具現値が、具現値間の通信を可能にすると言う意味で重要である。

具現値レイアウトでは、このような具現値を仮想的な格子平面の上に視覚的に配置し、その参照、継承、共有の関係を結線して行く（図３参照）。このとき、継承の関係を陽に指定することにより情報の可視性を最小にする（情報隠蔽）。レイアウトの原則は、互いに関連性が強い＜もの＞具現値同士は、近い距離に配置することである。全体を整然と配置するには、当然、かなりの試行錯誤を要する場合もありうるであろう。

具現値レイアウトの段階は、大きな問題を小さな概念の集りに分割するというプログラム設計上、最も重要なフェーズであると考えられる。設計者が問題構造をいかに捉えたかが、２次元の平面上に視覚的に表示されるので、別の設計者が見ても全体の構造を理解するのが容易になる。この点が、従来のフローチャートやプログラムテキストだけの場合では、欠落していた。また、この具現値レイアウトは、（具現値定義表を開いて）詳細を検索するときのブラウジング機能のディレクトリィの役目も果たす。

5．3 具現値の定義

各具現値の定義では、インタフェース（記号群）の定義とインプリメンテーション（宣言群）の定義を分離するようにする。この分離は、パラメータ化にとって本質的である。

レイアウト上で下位の具現値（それがない場合でも）は、＜概念＞モジュールの引数となる（パラメータ化されているものとみなされる）。継承する具現値は、記号群にその定義があり、参照する具現値は、宣言群にその定義がある（実際は、レイアウトで指定したものが自動的にセットされる）。

実際の具現値定義表は、つぎのような項目からなっている。

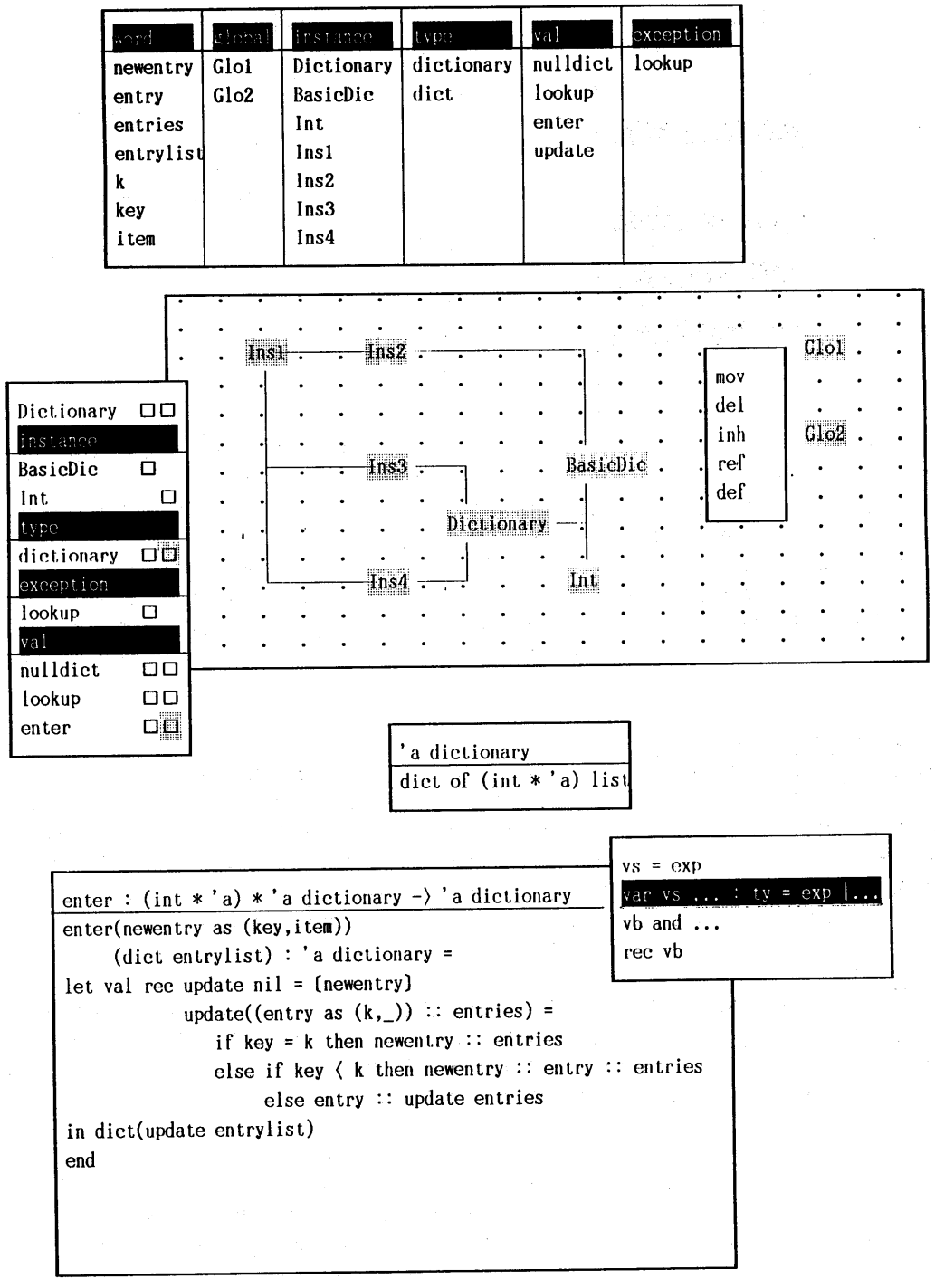
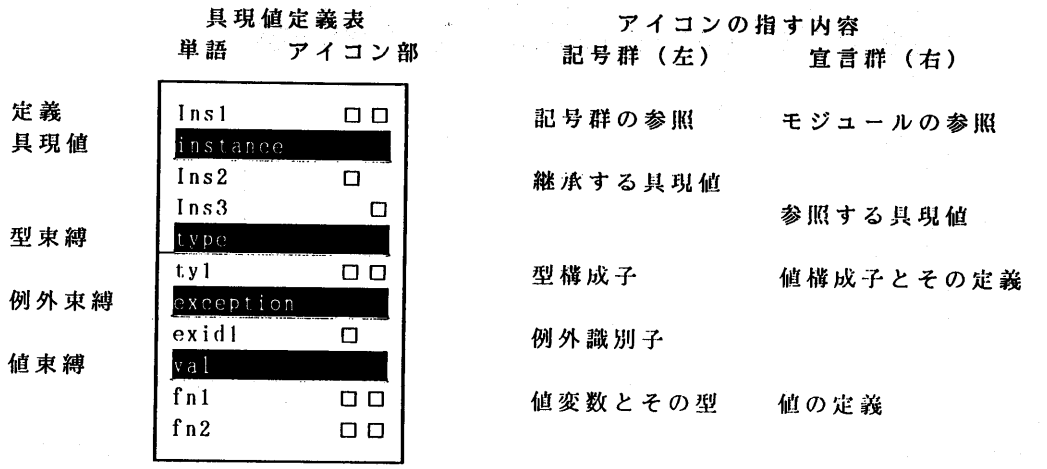


図3．設計支援環境の画面イメージ



レイアウト上の１つの具現値が選択されたとき、該当する具現値表が表示され、さらにアイコンを選択することによって各項目の詳細な定義が表示される。同時に、つぎに述べる構文誘導形エディタで修正が可能になる。アイコンは、その項目の状態を絵文字で表示したもので、たとえば、定義の完成度などを表現する。

先頭の定義項目は、自己の具現値名と、記号群と宣言群のアイコンからなる。このアイコン部は、既定値として、自己名（記号群名は、具現値名を大文字にしたもの、モジュール名は、具現値名にＭｏｄを付けたもの）が最初に設定されていると考える。他の記号群やモジュールを参照するときには、これらのアイコンを選択して定義することも可能である。

５．４ 構文誘導形エディタ

具現値定義表の各項目の詳細を定義する際に、この構文誘導形エディタを使用する。これは、構文規則に関するメニューがガイダンスとして表示され、メニューの項目の１つの構文を選択して行きながら、段階的に仕様記述を行なって行くものである（図３参照）。直接入力形式も部分的に許すが、原則として、構文規則の雛形、単語表や具現値定義表の単語をマウスで拾いながら構文エラーのないテキストを合成していくことになる。修正、複写、転送などの操作を構文単位ごとに行なえるようにする。

６．おわりに

本論文では、標準ＭＬを宣言的な仕様記述言語としたときの支援環境について構想と提案を行なった。特に、記述支援環境を中心に述べた。今後は、ワークステショーン上での記述支援系の試作とその評価を行ない、機能の充実とユーザインタフェースの改善を図り、実用化に耐え得るものにしていく考えである。

参考文献

〔1〕 R.Burstall,"Programming with Modules as Typed Functional Programming" Proceedings of The International Conference on FGCS 1984,pp 103-112.

〔2〕 R.Milner,"A proposal for standard ML" Conference Record of 1984 ACM Symposium on LISP and Functional Programming,ACM Aug. 1984,pp 184-197.

〔3〕 D.MacQueen,"Modules for standard ML" Conference Record of 1984 ACM Symposium on LISP and Functional Programming,ACM Aug. 1984,pp 198-207.

〔4〕 関根，大林，"新しいプログラム設計法－－－概念による設計法（ＤＭＣ）"，ｂｉｔ 1982 6月号 PP 102-114.