ソフトウェア工学 58－6
（１９８８．２．４）

概念による設計法（ＤＭＣ）に基づくプログラム開発環境の実現

松浦佐江子　中里淳子　大林正晴

（株）管理工学研究所

標準ＭＬを宣言的仕様記述言語とし、概念による設計法（Design Method based on Concepts）を方法論として取り入れたプログラム開発環境をワークステーション上に試作中である。今回は、この内の記述系のユーザインタフェースおよび構文誘導形エディタについて報告する。モジュール化機構を持たない標準ＭＬをＤＭＣにそって拡張し、この方法論に従って、単語表・概念構造図・インスタンス定義表・構文誘導型エディタといった４つの場面を設定した。この記述系の特徴は、人間の思考過程に合せてこの４つの場面を行き来しながら仕様を記述していくことである。さらに視覚的ユーザインタフェースを重視して開発を行なった。

Implementation of a Program Development Environment with the Design Method based on Concepts

Saeko Matsuura, Atsuko Nakazato, Masaharu Ohbayashi
Kanrikougaku,Ltd.
2-2-2,Sotokanda,Chiyoda-ku,Tokyou 101,Japan

We are implementing a program development environment in which the standard ML is used as a declarative specification language and the "Design Method based on Concepts" is adapted as a methodology.In this paper, we describe the user interface and the syntax synthtic editor in the system .

Since the standard ML doesn't have a mechanism of using modules, we have extended it with DMC, and create four major components which consist of the word table, the structured graph, the definition table of an instance and the syntax synthtic editor.

The feature of this system is that a specification is written by travelling the four components like as a process of human thinking. And we implement laying emphasis on the visual user interface.

１．はじめに

現在、より良いソフトウェアの設計および開発のために様々な方法論、言語、それらを統括した環境等が提案されている。われわれは、方法論として概念による設計法（ＤＭＣ）、言語として簡明かつ豊富な表現力をもつと思われる関数型言語ＭＬを採用して、１つのプログラム開発環境を提案したい。

標準ＭＬはモジュール化機構をもたないので、これをＤＭＣとの対応を旨く取るように拡張した。

現在われわれは、ＤＭＣという方法論による仕様の記述をサポートし、さらに関数型言語により詳細な記述を行なう過程をサポートする環境をＸｅｒｏｘ１１２１ワークステーション上にＩｎｔｅｒｌｉｓｐ－Ｄを用いて試作中である。環境としては解析系、変換系等、統合化したものを目指している。本論文では、その内の記述支援系について報告する。

なお、本研究の一部はＩＰＡの委託による協同システム開発（株）のソフトウェア環境統合化技術開発計画によるものである。

２．ＤＭＣに基づく記述支援系の設計

ＤＭＣにおいてキーとなるのは、問題を分析して抽出された＜概念＞を表わす種々の単語である。仕様を記述していく上でこれらの単語をキーワードとして仕様の構造を組み立てていく。すなわち抽出された＜概念＞の間の関係をつけ、ある種のモデルを構築するわけである。実際に紙の上で記述していく場合にも試行錯誤しながら記述が行なわれる。必要な単語を何度も書かなければならないこともある。ある部分だけ詳細に記述したいこともあると思われる。こうした人間の思考過程に即した環境を作ることがわれわれの第一の設計方針である。

仕様を記述していく場面として、次の４つの場面を設定した。使用したい単語を登録かつ分類する単語表。ＤＭＣに基づいて、問題の大まかな構造を設計する概念構造図。一つ一つの概念において具体的に何をしたいかを、それを表わす単語として登録するためのインスタンス定義表。その単語の行動を詳細に記述するための構文誘導型エディタ。こうした４つの場面を自由に行き来して仕様を記述していくのである。（図２－１参照）以下にそれぞれについて詳しく説明することにする。

また、ワークステーション上にこうした環境を作るのに、使い勝手の良いユーザインタフェースは重要である。試行錯誤の過程においてキーボードからの入力はなるべく少なくし、マウスによる柔軟な操作が行なえることが望ましい。このような、視覚的ユーザインタフェースを重視した環境の構築が第二の設計方針である。

われわれは問題を扱う１つの単位をボードと呼ぶことにした。ボード毎に上の４つの場面が存在する。

ＭＬという言語の特徴について一言述べる。ＭＬは宣言的な言語である。各宣言によって左辺に現れる変数に右辺の値を束縛させることができる。この変数と値の組からなる環境を生成してこの環境下において式を評価するわけである。われわれのいうＭＬは標準ＭＬにモジュール化機構をつけたもので、このモジュール構造が概念構造のインスタンスに対応する。すなわちあるボード内のインスタンスにおける各宣言によりそのボードの環境（変数に関数定義を束縛させた環境）が生成されるわけである。

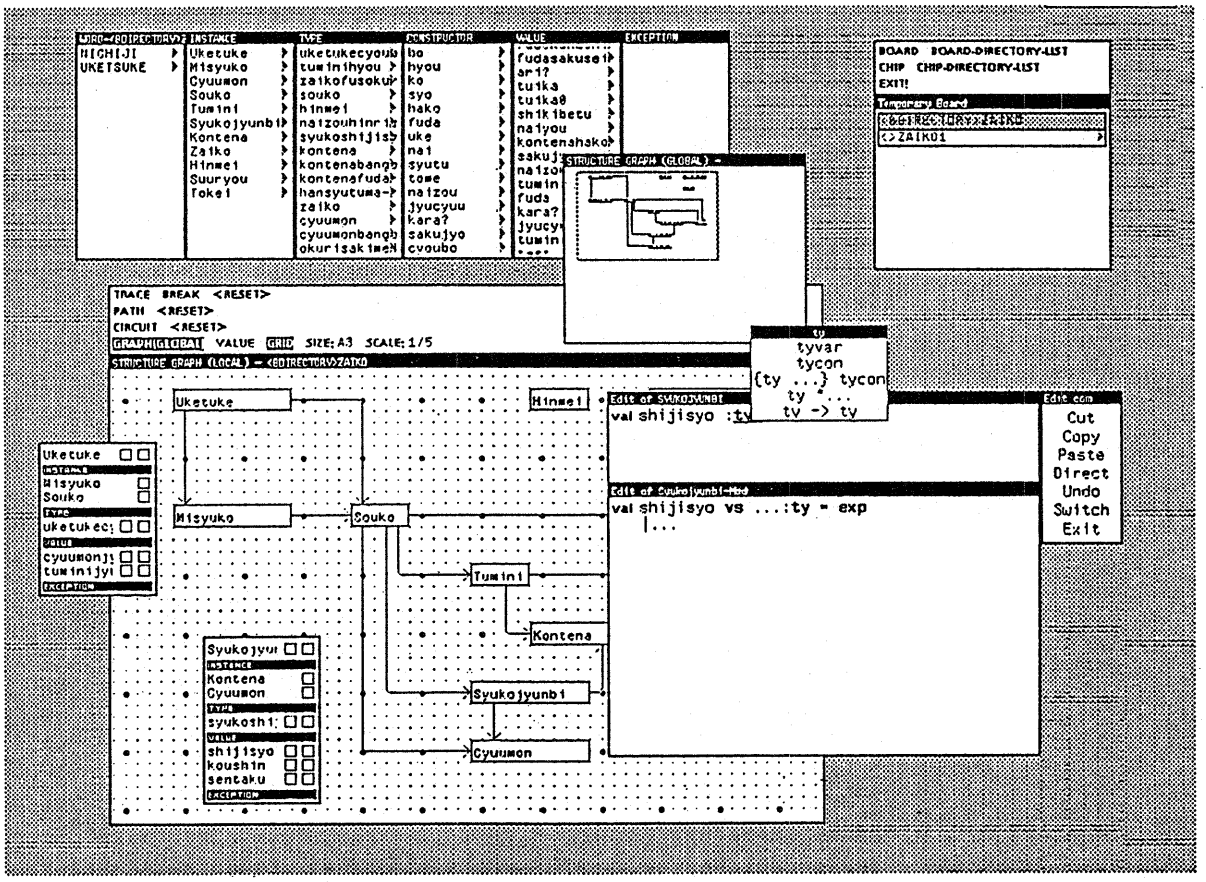
INSTANCE
TYPE
CONSTRUCTOR
VALUE
EXCEPTION
HIGHIJI
UKETSUKE
Uketuke
Misyuko
Cyuumon
Souko
Tumini
Syukojyunbi
Kontena
Zaiko
Hinmei
Suuryou
Tokei
BOARD BOARD-DIRECTORY-LIST
CHIP CHIP-DIRECTORY-LIST
EXIT!!
Temporary Board
TRACE BREAK <RESET>
PATH <RESET>
CIRCUIT <RESET>
GRAPH(GLOBAL) VALUE GRID SIZE:A3 SCALE:1/5
STRUCTURE GRAPH (LOCAL) - <EDITRECTRAY>ZAIKO
Uketuke
Misyuko
Souko
Tumini
Kontena
Syukojyunbi
Cyuumon
Hinmei
Syukojyui
shijisyo
koushin
sentaku
tyvar
tycon
{ty ...} tycon
ty -> ty
val shijisyo :ty
val shijisyo vs ...:ty = exp
Edit com
Cut
Copy
Paste
Direct
Undo
Switch
Exit

図２－１　記述系の全体像

3．単語の分類と整理

一般に仕様またはプログラムを記述していく際、同じ単語を何度も使用することが多い。そこで初めに思いつく単語を単語表に登録しておいて利用することにした。まず、問題を分析して、仕様を記述していくのに適当と思われる単語をキーボードから入力する。しかし、一次元の単語表では分かりにくいので使用目的にあわせて分類することにした。単語の分類の種類は次の６種類である。

①WORD	使用目的の決まらない単語
②INSTANCE	仕様の構造を記述していく上で大まかな概念となる単語
③TYPE	各インスタンスを記述する際の型構成子を表わす単語
④CONSTRUCTOR	各インスタンスを記述する際の構成子を表わす単語
⑤VALUE	各インスタンスを記述する際の関数名や変数名を表わす単語
⑥EXCEPTION	各インスタンスを記述する際の例外識別子を表わす単語

はじめから目的に合わせて単語を登録しても良いし、WORDの欄から他の欄へ移動したりコピーしたりして分類しても良い。これらの単語は視覚的に分類されているだけではなく、単語表から他の場面である概念構造図、インスタンス定義表、構文誘導型エディタへ移す時の属性としての効果を持つ。すなわち、属性が合わない場合にはメッセージが表示される。

また、INSTANCEとTYPEに登録された単語は使用後は単語表から削除することにした。それは次のような理由による。インスタンス名および型構成子として使用する単語は１つのボードにおいて一意である。そこで、どのインスタンスに属するかを決めた場合、単語表から削除したほうが分かりやすい。ただし、インスタンス定義表から削除した場合にはもとの単語表に戻すこととする。こうしておけば未使用の単語が一目で分かる。（図３－１参照）

表３－１　単語表の機能一覧

機能	コマンド	内容
単語の入力	ADD	登録したい単語表の欄(WORD・VALUE等)のウィンドウ上でこのコマンドを選択すると入力ウィンドウが現れるので単語をキーボードより１語入力すると指定欄に単語が登録される
連続入力	ADD-REPEAT	ADDと同様であるが単語の連続入力ができる
未使用単語の表示	DISPLAY-PARTS	登録したい単語表の欄（WORD・VALUE等）のウィンドウ上でこのコマンドを選択するとその項の未使用の単語が表示される
入力単語の全体表示	DISPLAY-ALL	登録したい単語表の欄（WORD・VALUE等）のウィンドウ上でこのコマンドを選択するとその欄の登録されたすべての単語が表示される
単語表の拡大縮小	SHAPE	単語表全体の大きさを指定して変更する
単語表の移動	MOVE	単語表全体を移動する
単語表のクローズ	CLOSE	単語表全体を閉じる
単語表の縮退	SHRINK	単語表を各欄ごとに重ね合わせる
単語の削除	DELETE	各単語のサブメニューでその単語を削除する
単語の複写	COPY	各単語のサブメニューでその単語を次にマウスをクリックした位置の欄に複写する
単語の移動		単語を選択して次にマウスをクリックした位置の欄にその単語を移動する

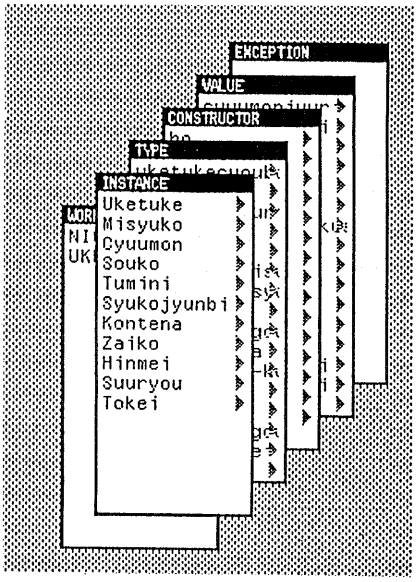

図３－１　単語表

4．概念構造図

さて、問題を分析してその仕様の大まかな概念となる単語がいくつか抽出されたならば、それらを使って構造を作っていくことになる。ここでわれわれはこの構造を概念構造図と呼ばれる２次元のグラフ上に表現する（図４－１参照）。概念構造図上に表現さ

れるのは、ある概念を表わす単語とそれらの間の関係である。構造を構築していく場合、ある概念の具現値であるインスタンスから始まってその下位のインスタンスを定義していく。この時、下位のインスタンスの記号群を単に使用（参照）したいだけならばＲＥＦＥＲ（参照）関係で結べば良い。また下位の記号群も含めてそのインスタンスの環境を構築したいならばＩＮＨＥＲＩＴ（継承）関係で結べば良い。このようにして、構造をレイアウトしていくわけだが、実際には何度も書き直しをしなくてはならないだろう。そこで柔軟なユーザインタフェースが必要になる。そこで、概念構造図におけるユーザインタフェースについて説明する。

まず、画面上にレイアウトしていくうえでの目安になるのがグリッドである。一つのインスタンスからいくつもの関係が結ばれることがあるので、インスタンスを囲む四角形の四方からグリッドを目安として細かく線が引けるようにした。グリッドは消去することもできる。参照・継承関係は上位から下位への矢印で表わされる。

スペースの足りない箇所に新たにインスタンスを挿入したい場合には、インスタンスの位置を移動するか、グリッドをＸ方向あるいはＹ方向へ挿入して画面を引き伸ばせば良い。レイアウトできる画面の大きさは何種類か指定できる。この仮想画面をグローバルビュー、実際に操作している画面をローカルビューと呼ぶ。グロバルビューには縮尺が指定でき、これにより画面のどこをどの位使用しているかがわかるし、ここで見たい位置への移動もできる。

その他、解析系との連係を取る操作もあるがここでは省略する。

表４－１　概念構造図の機能一覧

機能		コマンド	内容
インスタンス編集	インスタンスの削除	DELETE	インスタンス名を選択して現れるメニューからコマンドを選択し（以下これを’インスタンス名を選択して’と略記する）概念構造図からそのインスタンスを削除すると同時にインスタンス名を単語表に戻す
	インスタンスのグローバル化	GLOBAL	インスタンス名を選択してそのインスタンスをグローバルインスタンスにする
	参照関係の表示	REFER	インスタンス名を選択して結びたいインスタンスまでの経路を指定すると矢印で結ばれる
	継承関係の表示	INHERIT	インスタンス名を選択して結びたいインスタンスまでの経路を指定すると太線の矢印で結ばれる
	参照・継承関係の削除	DEL-LINK	インスタンス名を選択して関係を削除したいインスタンスを指定すると矢印が消える
	インスタンスの位置の移動	MOVE	インスタンス名を選択して移動したい位置でマウスをクリックするとインスタンス名と矢印が移動する
	インスタンス定義表の呼び出し	DEF-LIST	インスタンス名を選択してそのインスタンスの定義表を呼び出す
画面編集	グリッドの表示の切り替え	GRID	概念構造図上でコマンドを選択してグリッド表示をＯＮ－ＯＦＦする
	グリッドのＸ方向の挿入	ADDLINE-X	概念構造図上でコマンドを選択して挿入したい位置でマウスをクリックするとＸ方向にグリッドが挿入される
	グリッドのＸ方向の削除	DELLINE-X	概念構造図上でコマンドを選択して挿入したい位置でマウスをクリックするとＸ方向のグリッドが削除される
	グリッドのＹ方向の挿入	ADDLINE-Y	概念構造図上でコマンドを選択して挿入したい位置でマウスをクリックするとＹ方向にグリッドが挿入される
	グリッドのＹ方向の削除	DELLINE-Y	概念構造図上でコマンドを選択して挿入したい位置でマウスをクリックするとＹ方向のグリッドが削除される
画面操作	画面の拡大縮小	SHAPE	画面の大きさを指定して変更する
	画面の移動	MOVE	画面を位置を指定して移動する
	画面のクローズ	CLOSE	画面を閉じる
	画面の再表示	REDISPLAY	画面を再表示する
	画面の最大面積の切り替え	SIZE	画面の最大面積を指定する(A4 A3 B4 B3)
	グローバルビューの縮尺の切り替え	SCALE	グローバルビューの縮尺を指定する（1/2 1/5）

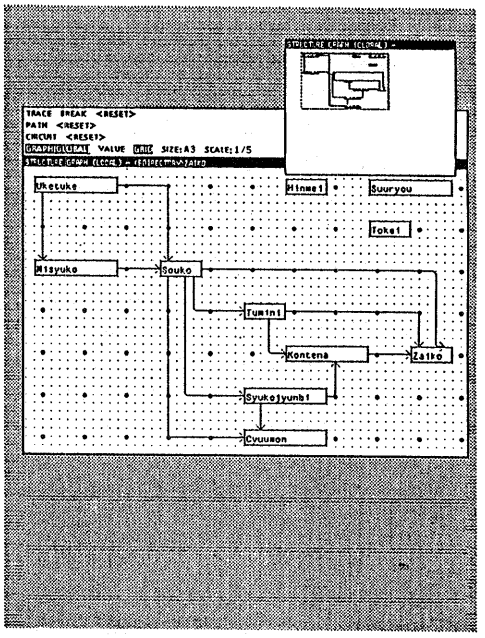

図４－１　概念構造図

5．インスタンス定義表

各インスタンスは記号群と宣言群から構成される。記号群とはいわゆるインスタンスの型を表わすものであり、宣言群はインスタンスの環境を生成するものである。概念を表わすインスタンスに対してそこでどの様な環境を作るかを、関数名や型名に使用したい単語を各インスタンスに振り分けることにより行なうのである。記号群と宣言群の名前は既定値としてインスタンス名からとった名前がつけられている。（図５－１参照）

ここではインスタンスで定義されている宣言の列をテキスト形式でみることができる。現在は記述系に関する機能のみがあるが、この定義表にインスタンスの部品化などの呼び出し機能をつけたいと考えている。

表５－１　インスタンス定義表の機能一覧

機能	コマンド	内容
エディタ呼び出し		インスタンス定義表の各宣言名の横の左のアイコンをクリックすると記号群のエディタが右のアイコンをクリックすると宣言群のエディタが呼び出される
単語の選択		エディト中に宣言名の単語を選択するとそれが埋め込み可能ならば構文要素として埋め込む
単語の削除	DELETE	宣言名の単語を選択するとメニューが表示されるのでコマンドを選択すると、インスタンス定義表からその単語を削除すると同時に単語表に戻す
インスタンス定義一覧	DEF-LIST	インスタンス名を選択するとメニューが表示されるのでコマンドを選択すると、そのインスタンスで定義されている宣言の列がテキスト形式で表示される

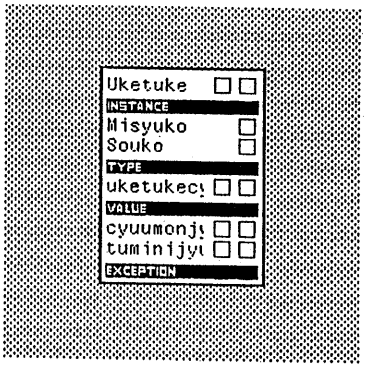

図５－１　インスタンス定義表

6．構文誘導形エディタ

概念構造図のレイアウトやインスタンス定義表への単語の分配をしながら、実際の型や関数の定義を構文誘導型エディタで行なう。（図６－１参照）

このエディタはＭＬの構造エディタである。すなわちＭＬの構文要素がメニューで表示されるのでユーザはメニューより構文要素を選択することでプログラムを記述できる。こうした構文誘導型エディタの利点は構文の誤りがないことや、段階的詳細化が可能なことである。しかし、目的の式を記述するのに何回もメニューを選択しなければならない煩わしさもあるし、汎用性も失われる。なぜ、こうしたエディタを採用したかというと、段階的詳細化が行なえるので、我々の設計方針である４つの場面の中での試行錯誤に適していると考えたからである。ここではＭＬという言語を採用したが、他の言語を当てはめることも可能と考える。そこで今回はＭＬの特徴を生かしたエディタを考えた。

一般の構造エディタと同様に次の機能がある。エディト・ウィンドウ内でのマウスによるクリックでその構文要素に対応するメニューが表示される。メニューを選択することで構文要素が埋め込まれる。Ｃｕｔ・Ｃｏｐｙ・Ｐａｓｔｅ・Ｓｗｉｔｃｈ・Ｕｎｄｏ・Ｄｉｒｅｃｔコマンドにより構文要素単位での編集ができる。これらに加えて、次のようなことをシステムがサポートすることにした。

まず、単語表やインスタンス定義表から単語を拾って埋め込むことができることである。特に関数名および構成子として使用している単語はその関数と引数のapply形式で埋め込むことができる。例を上げて説明しよう。ここでval,if等の強調文字はＭＬの予約語を、con,exp等の下線のついた語は非終端記号を表わす。

例　Instance1
　　INSTANCE1
　　val func1:type1 * type2 -〉 type3

インスタンスInstance1の記号群INSTANCE1においてfunc1が上のように定義されてい

るとする。

```
Instance2
Instance2-Mod
val func2 (arg1 ,vs ,...) = if exp then exp else exp
```

現在インスタンスInstance2宣言群Instance2-Modの値宣言func2の右辺のif式のthen部のexpをエディト中とする。
この時Instance1がInstance2の下位のインスタンスで参照関係・継承関係にあるか、継承関係をはさんで参照関係にあれば、インスタンス定義表または単語表のfunc1を指定することで次のように埋め込むことができる。

```
Instance2
Instance2-Mod
val func2 (arg1 ,vs ,...) = if exp
                            then func1(type1 ,type2,...)
                            else exp
```

これは構造的に埋め込まれているのでＣｕｔコマンドによってtype1をexpに戻して別のパターンを埋め込むこともできる。
すなわちＣｕｔコマンドにより

```
val func2 (arg1 ,vs ,...) = if exp
                            then func1(exp ,type2,...)
                            else exp
```

とし、式にメニューよりcon avsを選択し続けて単語表の構成子constructor1を指定するとこの構成子の定義が存在すれば次のように埋め込まれる。

```
val func2 (arg1 ,vs ,...) = if exp
            then func1(constructor1(int1,string1,...) ,type2,...)
            else exp
```

ただし、constructor1は型宣言

```
type type1 = constructor1 of (int * string)
```

より constructor1 : int * string -〉 type1なる型を持つ。

視覚的ユーザインタフェースの観点から、どの位詳細化が行なわれているかが視覚的にわかるように、非終端記号・終端記号・予約語・ユーザ定義語またはシステム関数をフォントを変えて表示した。

ＭＬは型付けされた言語である。型をもつことはある種の検証になるが、すべてに型をつけていては記述が面倒である。そこで記述する際には型をあまり意識せずにシステム内で型を決め、検証を行なうこととする。型を使ったチェックにはボード内のインスタンスの接続の整合性のチェックと、各インスタンスの定義内の宣言の整合性のチェックがある。
ボード内のインスタンス間の接続関係のチェックを行なうことによって、型構成子の使用、関数名の使用における単語の誤った使用を防ぐことができる。すなわちＯＭＣという方法論によって規定されている事項をシステムがチェックし、メッセージを提示することにより自然と方法論の学習にもなるわけである。

宣言はエディタをＥｘｉｔする時に解析系に渡す内部形式に変換されている。このとき、まだ詳細化が終了していない構文については未定義として内部形式を生成している。これを使って未定義仕様の実行ということも考えている。

表６－１　構文誘導型エディタの機能一覧

機能	コマンド	内容
段階的詳細化		エディタ内でマウスをクリックすると指定された構文要素に対応する構文メニューが表示されるのでそれを選択する
エディトコマンド	Cut	コマンドを選択してから取り除きたい要素を指定すると、その部分を反転表示してもとの構文要素に戻る。カットした構文単位はバッファに保持および表示される
	Copy	コマンドを選択してからコピーしたい要素を指定すると、その部分を反転表示のみする。その構文単位はバッファに保持および表示される
	Paste	コマンドを選択して構文要素を指定するとCutまたはCopyによりカットされた構文を指定された構文要素に埋め込む
	Direct	コマンドを選択して構文要素を指定すると終端記号または単純な式を直接入力できる
	Switch	コマンドを選択して構文要素を指定するとCopyによりカットされた構文を指定された構文要素とを入れ替える
	Undo	コマンドを選択すると１回前の状態に戻る
	Exit	コマンドを選択するとエディトを終了する
複数エディタのオープン		インスタンス定義表からエディタを呼び出すと２つめからはユーザが位置を指定してエディタをオープンする
エディタ画面の移動	Move	右マウスのクリックにより表示されるメニューよりコマンドを選択してエディト・ウィンドウを移動する
エディタ画面の拡大縮小	Shape	右マウスのクリックにより表示されるメニューよりコマンドを選択してエディト・ウィンドウの大きさを指定して変更する
単語表およびインスタンス定義表からの単語の埋め込み		構文要素メニューから選択しないで単語表またはインスタンス定義表から単語を選択するとチェック後単語を埋め込む

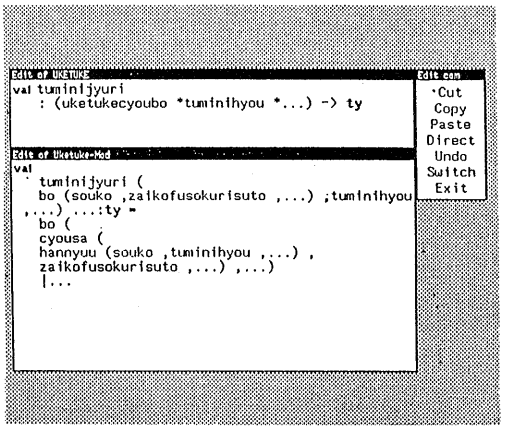

図６－１　構文誘導型エディタ

７．おわりに

問題を扱う単位としてボードを設定した。ボードの大きさは視覚的にも人間が扱える単位が望ましい。しかし、問題が大きくなると１つのボードだけでは表現できなくなるだろう。そこでボードの階層化ということが必要になってくる。これについては現在検討中である。

一方、こうしたボードを管理する機構、すなわちディレクトリイ等も必要になる。従来の単なるツリー構造ではなくネットワーク構造を考えている。

また、インスタンスをボードから切り放して部品化し、それを再利用することも検討中である。すなわちインスタンスの記号群と宣言群をそれぞれ必要な情報とともに切り放しチップと呼んで管理する。インスタンスを定義する時にその記号群や宣言群としてチップを再利用するわけである。

さらに以上のような記述支援系を使って記述した仕様を実際に動かすための解析系を現在試作中である。１つのボードに対して、それを実行するためのトップレベルの環境を定義し評価を行なう。この環境を幾つも用意することで様々な実行ができる。さらにこれらの図式的表現のデバッガや、先に述べた未定義仕様の実行についても考案中である。

参考文献

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

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

〔3〕T.Teitelbaum,"The Cornell Program Synthesizer:A Syntax-Directed Programming Environment"ACM Vol.24,No.9 1981 pp 563-578

〔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 pp 102-114

〔5〕大林，"標準ＭＬを用いたプログラム設計支援環境",ソフトウェア工学 51-1 1986.11.26

〔6〕大林，"概念による設計法（ＤＭＣ）による在庫管理システムの記述",ソフトウェア工学 51-5 1987