知識工学と人工知能 39 － 5
（１９８５．３．１９）

知識型ＶＬＳＩ－ＣＡＤへ向けてのセルライブラリシステム－ＯＣＬＳ

金井　直樹，　石塚　満
（東京大学生産技術研究所）

１．はじめに

ＶＬＳＩのＣＡＤは、既存の設計の流れにそって自動化が図られてきたが、複雑度に対処し、高度な支援機能を実現するために知識型システムによる統合化された設計環境が必要になってきている。

我々は、知識型ＶＬＳＩ－ＣＡＤへ向けて、次のような観点から研究、開発を進めている。

- ▷ 知識を内蔵する部品化のアプローチ
- ▷ 構造を持ち深い推論を行なう知識型システムのアプローチ
- ▷ 設計ルールに基づく検証機能の統合
- ▷ コンサルテーション型エクスパート・システムの組み込み
- ▷ 知識表現法
- ▷ グラフィクスとマウスによるユーザ・インタフェース

ここでは、知識型ＶＬＳＩ－ＣＡＤシステムの下部構造に位置づけられ、かつ、独立しても使用できる、支援機能を持った、セル・ライブラリ構築システム ― ＯＣＬＳについて報告する。

２．オブジェクト指向型セル・ライブラリ・システム

知識型ＶＬＳＩ－ＣＡＤに要求される目的を考慮すると、セル・ライブラリには次の様な能力が要求される。

<1>　ある基本パターンを変更して条件や状況に応じたパターンを生成する能力
<2>　ある基本パターンに関する様々な記述レベルの表現を統一的に扱い管理する能力

オブジェクト指向型セル・ライブラリ・システムＯＣＬＳ（Object oriented Cell Library System)[1]は、レイアウト記述レベルの情報を中心に扱い、<1> の要求を満たすセル・ライブラリを構築するシステムである。又、ＯＣＬＳは、オブジェクト指向の考え方を用いることにより、<2> の要求を満たすセル・ライブラリを構築するための拡張性を有している。

ＯＣＬＳの基本的な機能は以下のものである。

(1) 融通性のあるセル・ライブラリ
　ＯＣＬＳでは、セル・ライブラリ中の基本パターンから、設計時の状況や条件に応じたパターンを生み出すことができる。又、新しく設計されたパターンを基本パターンとしてライブラリに登録することができ、同時にその基本パターンから条件にあったパターンを生成するための知識や、仕様などの情報をライブラリに登録することができる。
(2) デザイン・ルール[2] などのチェック
　ＯＣＬＳでは、デザイン・ルールなどに関する知識を持ち、設計時には、それらのルールを満たしているかどうかのチェックを行なうことができる。
(3) 基本パターン選択に関する支援
　ＯＣＬＳでは、基本パターンに関する情報をユーザに提示することにより、基本パターン選択に関する支援を行なう。
(4) 会話型のユーザ・インタフェース
　ＯＣＬＳでは、マウス、メニュー、グラフィクス、スクリーン・エディタなどを用いた会話型のユーザ・インタフェースを持ち、ＯＣＬＳ自体で、ＣＡＤの機能を持つ。

ＯＣＬＳは、オブジェクト指向型のシステムである。ＶＬＳＩの構造化設計手法では、それぞれのブロック（パターン）が、外部からはブラック・ボックスとして取り扱われることを考慮すると、一つのパターンは、通常のオブジェクト指向型言語におけるオブジェクトとしてとらえることができる。基本パターンはクラスに対応する。基本パターンから条件に応じたパターンを生成するのは、クラスにメッセージを送ってインスタンスを生成することに対応する。この様に、ＶＬＳＩの構造化設計手法とオブジェクト指向の考え方は、非常に良くマッチする。

ＯＣＬＳでは、通常のオブジェクト指向型言語と異なり、２種類のオブジェクトを用意している。１つはセル・オブジェクトであり、もう一つはルール・オブジェクトである。セル・オブジェクトは、基本パターンに関する情報を集めたものである。セル・オブジェクトは、従来のオブジェクト指向型言語におけるオブジェクトに対応する。ルール・オブジェクトは、デザイン・ルールなどの設計時にチェックすべき規則を集めたものであり、ルールを階層的に扱うことを目標として作られている。ルール・オブジェクトは、セル・オブジェクトと異なる性質を持つ。

ＯＣＬＳは、図１の様に、セル・ライブラリ、ルール・ライブラリ、推論システムにより構成されている。セル・ライブラリは、セル・オブジェクトを集めたものであり、ルール・ライブラリは、ルール・オブジェクトを集めたものである。推論システムは、セル・ライブラリとルール・ライブラリを参照して設計を行なう。

３．セル・オブジェクト

３．１．クラス

セル・オブジェクトは、ある一つのパターンに関する情報を集めたものである。セル・オブジェクトのうち、一つの基本パターンに関する情報を定義したものをクラスと呼ぶ。クラスに蓄えられる情報は、以下の３つに大別される。それぞれの情報はクラス名を述語名とするフレームの形で定義されている。

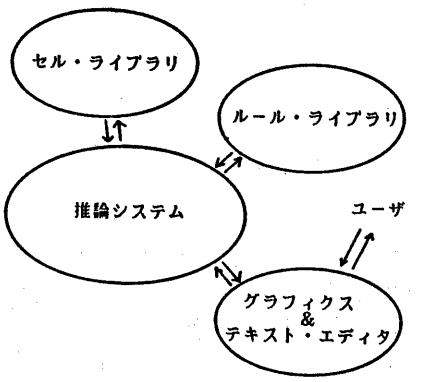

図１．ＯＣＬＳの概念図

<1> 仕様、機能などに関する情報
- ▷レベル：レベルは、ユーザが機能の複雑さに応じて定義する。
- ▷カテゴリー：そのクラスの機能名を表わす。
- ▷デザイナー：設計者名。
- ▷詳細情報：そのクラスの機能などに関する、自然言語で書かれた情報。
- ▷ルール・オブジェクト：そのクラスが満たすべきルールを持ったルール・オブジェクトの名前。
- ▷スーパー・クラス：そのクラスのスーパー・クラスの名前。

<2> パターン情報
- ▷デフォルト：その基本パターンのデフォルト値を定義する。
- ▷パターン：パターン・データ情報。具体的には、他のクラスのインスタンスを組み合わせたもの。

<3> 条件に応じたインスタンスを生成するための知識
- ▷メソッド：メソッドの形で定義される。

クラスは、#で始まる名前を持つ。最上位のスーパー・クラスは、#objectであり、#objectのみは、スーパー・クラスを持たない。

現時点では、レイアウト記述レベルを対象にしているため、マスク・パターン情報のみが扱えるが、セル・オブジェクトのデータ構造をフレームで実現しているため、他の記述レベルのスロットを用意することにより、様々な記述レベルでの情報を統一的に扱うことができる。

３．２．メソッドの定義

メソッドは、C-Prolog上で、

クラス名（method，
　　メッセージ・パターン，
　　メソッド本体）．

の形式で表現されている。

メッセージ・パターンとメソッド本体は、リストの形で表現される。メッセージ・パターンとは、メソッド本体を実行するために、クラスやインスタンスが、受け取るべきメッセージを表わす。メソッド本体は、クラスやインスタンスが、メッセージが受け取った時に行なうべきふるまいを表わす。

図２は、階乗の定義例である。この例では、自分自身に[=,0] というメッセージを送り、答えが真ならば、then以下を実行し、オブジェクト１を返す。偽ならelse以下を実行し、階乗の計算結果を返す。

３．３．パターン定義の形式と　パターン定義用メソッド

３．３．１．デフォルトの設定

デフォルトの宣言は、そのパターンのデフォルト値を設定する他、外部から与えるパラメータの宣言にもなる。デフォルトの設定は、

［［パラメータ名，値］……］

の形式で行なわれる。デフォルト宣言されたパラメータは、外部から指定されないかぎり、デフォルト値が使用される。

```
'#fact'(super,'#number').
'#fact'(method,!,
    [
      [self,
        [=,0],
        [then,[1,return]],
        [else,[a,:=,self,[-,1],!],
              [self,[*,a],return]]]
    ]).
```

図２．メソッドの定義例

```
クラス名（patern,
 [[part,
     [パート名１,:=,クラス１,
        {クラス変更に関するメッセージ}，（*1）
        {パターン・データを展開するメッセージ}，
        {配置、回転等に関するメッセージ}，（*1）
              ⋮
     ],
     [パート名２,:=,クラス２……],
              ⋮
 ],
 [wire,
     [ワイヤ名１,:=,#wire,
        [run_layer,レイヤ名１]，
        [run_width,巾]，　　（*2）
        [start_pt, 場所]，
        {配線指示のメッセージ}，
              ⋮
     ],
     [ワイヤ名２,:=,#wire……],
              ⋮
 ],
 [term,
     [ターミナル名１,:=,場所]，
              ⋮
 ]]).
```

*1　０個以上、いくつでもよい。

*2　省略された場合、層ごとに定義された値を用いる。

図３．パターン・データの形式

３．３．２．パターン定義

パターンの記述方法は、ＭＩＴで開発された高級パターン記述言語ＤＰＬ（Design Procedure Language)[3]を基にしている。パターン・データは、図３の様に、使用するサブ・ブロックに関する情報を扱うパート部、サブ・ブロック間の配線の情報を扱うワイヤ部、端子に関する情報を扱うターミナル部より構成されている。パート、ワイヤ、ターミナルのそれぞれの部分が不要の場合は省略してよい。最もプリミティブなクラスは、#rec であり、四角を表わす。#rec のインスタンスは、レイヤ、巾、長さの情報を持つ。

３．３．３．パート部

▷クラス変更に関するメッセージ

ユーザが、受け手のクラスのメソッドとして定義しておく必要がある。これにより、基本パターンから、条件に応じたパターンを生成することができる。

▷パターン・データを展開するメッセージ

パターン・データは、構造化されていて、内部は他のクラスのインスタンスの組み合わせとして定義されており、グラフィクスに出力する時などには、物理的なデータに展開する必要がある。これは組み込みメソッドexpandを用いて行なわれる。メソッドexpandは、引数を与えることにより、デフォルト値以外の値を使用することができる。

▷配置に関するメッセージ

配置は、メソッドallignを用いて行なわれる。

[allign,移動指定点，移動目標点]

の形で与えられる。移動指定点で、そのインスタンス中の一点を指定し、その点が、移動目標点に移るように、そのインスタンスを移動したものが返される。

▷回転、反転に関するメッセージ

回転は、メソッドrot90、rot180、rot270を用いて行なう。メッセージの受け手に対して、それぞれ、反時計回りに 90°、180°、270°回転したものが返される。

反転は、メソッドneg_x、neg_y、int_pos、int_neg を用いて行なわれる。メッセージの受け手に対して、それぞれ、Ｙ軸対称、Ｘ軸対称、Ｙ＝Ｘ対称、Ｙ＝－Ｘ対称のものが返される。

３．３．４．ワイヤ部

ワイヤ部は、クラス#wireのインスタンスにより、構成されている。#wireのインスタンスは、内部情報として、現在位置座標（カレント・ポイント）、現在のレイヤ、現在のワイヤの巾、それまでに作られたワイヤの情報などを持っている。クラス#wireのインスタンスにメッセージを送ることによりカレント・ポイントが移動し、この移動の軌跡が配線になる。#wireのインスタンスは、#wireに対して、メソッドrun_layer、run_width、start_ptが送られることにより作られる。

３．３．５．ターミナル部

パターン中の座標を名前にわりあて、外部からその名前で座標を参照できるようにする。パターン中の座標は、移動目標点と同様に与えられる。ワイヤ部に関しては、配線中にstore_ptで指定された点でも与えられる。

３．４．条件に応じたインスタンスを生成するための組み込みメソッド

ＯＣＬＳでは、基本パターンから条件に応じたインスタンスを生成することができる。これは、条件に応じて基本パターンをどの様に変更するかを、メソッド本体内に記述することにより実現できる。これが、パート部中の、クラス変更に関するメッセージに対応する。

メソッドの受け手、送り手ともに、パターン・データのオブジェクトである。これらの組み込みメソッドを利用することにより、同一の基本パターンから、ユーザが望

むような変更を加えたパターンをつくりだすためのメソッドが容易に作れる。以下、これらの組み込みメソッドについて述べる。

▷add: メッセージの受け手のパターン情報の指定された場所に、新しいインスタンスに関する情報を加えたパターン・データのオブジェクトが返される。

▷remove: メッセージの受け手のパターン・データから指定されたインスタンスを取り除いたパターン・データのオブジェクトが返される。

▷exchange: メッセージの受け手のパターン情報の指定されたインスタンスを、与えられたものと交換したパターン・データのオブジェクトが返される。

▷ch expand、ch align、ch xfrm: メッセージの受け手のパターン情報における、パート部中の指定されたインスタンスの、メッセージexpand、align のパラメータ、あるいは、回転、反転のメッセージを変更したパターン・データのオブジェクトが返される。

▷rm align、rm xfrm: メッセージの受け手のパターン情報から、指定されたパート部中のインスタンスのalignメッセージ、あるいは、回転、反転のメッ

```
'#nand'(designer,'Naoki Kanai').
'#nand'(category,nand).
'#nand'(level,gate).
'#nand'(super,'#inverter').
'#nand'(rule,'@rule_object').
'#nand'(patern,
   [
    [part,
      [pulldown],
      [pulldown(1),:=,'#pass_transistor',
        [expand,[parameter,[channel_length,pulldown_length],
                           [channel_width,pulldown_width]]],
        [allign,[>>,top_center,diff_piece],
          [>>,bottom_center,diff,contact,pullup]]],
      [pulldown(2),:=,'#pass_transistor',
        for_nand,
        [expand,[parameter,[channel_length,pulldown_length],
                           [channel_width,pulldown_width]]],
        [allign,[>>,top_right,diff_piece],
          [>>,bottom_right,diff_piece,pulldown(1)]]]],
    [term,
      [input_pt],
      [input_pt(1),:=,[>>,center_left,poly_piece,pulldown(1)]],
      [input_pt(2),:=,[>>,center_left,poly_piece,pulldown(2)]]]]).
'#nand'(method,
  [input,N],
    [
      [N,
        [<=,2],
        [then,[self,return]],
        [else,[m,:=,N,[-,1]],
          [self,[input,m],
            [add,part,
              $,
              [pulldown(N),:=,
                '#pass_transistor',
                for_nand,
                [expand,[parameter,[channel_length,pulldown_length],
                                   [channel_width,pulldown_width]]],
                [allign,[>>,top_right,diff_piece],
                  [>>,bottom_right,diff_piece,pulldown(m)]]],
            [add,term,
              [input_pt(N),:=,
                [>>,center_left,poly_piece,pulldown(m)]]],
            return]]]]).
```

図４．クラス#nand

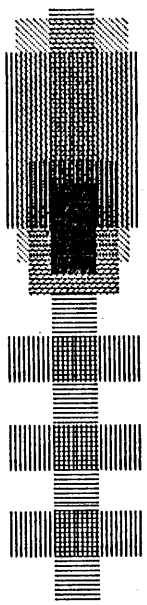

図５．クラス#nandに
メッセージ [input,3]を送った結果

セージを消去したパターン・データのオブジェクトが返される。

３．５．セル・オブジェクトにおけるインヘリタンス

ＯＣＬＳではメソッドのインヘリタンスの他に、デフォルト宣言とパターン・データのインヘリタンスも行なう。現段階では、単純継承を行ない多重継承の機能はない。

メソッドのインヘリタンスでは、通常のオブジェクト指向型言語のインヘリタンスと同じ役割をはたす。

デフォルト値のインヘリタンスでは、スーパー・クラスのデフォルト宣言も、下位のクラスのデフォルト宣言の一部として扱われる。同じパラメータがあった場合には、スーパー・クラスでの、そのパラメータのデフォルト宣言が無視される。

パターン情報のインヘリタンスでは、スーパー・クラスのパターン・データも、そのクラスのパターン・データの一部として用いられる。ただし、クラスとスーパー・クラスの双方に、同じインスタンス名を持ったメッセージ列があった場合には、下位のクラスのデータが有効となる。

３．６．セル・オブジェクトの例

セル・オブジェクト（クラス）の定義例として、NMOSのNANDの例を示す。図４は、クラス#nandの定義である。#nandのスーパー・クラスは#inverterである。図５は、そのパターンである。

４．ルール・オブジェクト

４．１．ルール・オブジェクト

ルール・オブジェクトは、あるクラスのインスタンスが満たすべきルールを定義するための、特別なオブジェクトである。ルール・オブジェクトには、インスタンスはない。このため、ルール・オブジェクトをルール・クラスとも呼ぶことにする。

ルール・オブジェクト内に定義される情報は、スーパー・ルール・クラス名とルールの集合である。ルール・オブジェクトのスーパー・クラスは、ルール・オブジェクトでなければならず、ルール・オブジェクトのスーパー・クラスをスーパー・ルール・クラスと呼ぶことにする。ルール・オブジェクトは、@で始まる名前を持つ。最上位のスーパー・ルール・クラスは、@rule_object である。セル・オブジェクトとルール・オブジェクトの関係の例を図６に示す。

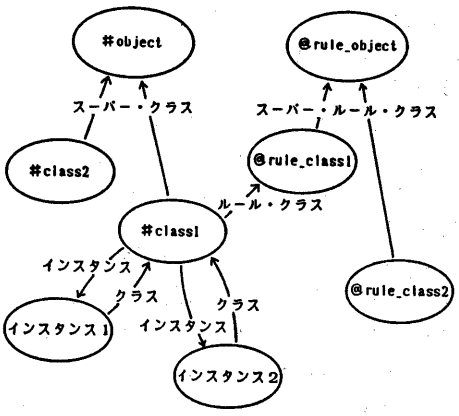

図６．セル・オブジェクトとルール・オブジェクトの関係の例

ルール・オブジェクトは、ルール・クラス名を述語名とするフレームの形で定義される。ルール・オブジェクトでは、メソッドあるいは、メッセージという概念はなく、組み込みルールを用いて表現されたルールが定義されているだけである。組み込みルールを用いることにより、様々なデバイス技術に応じたルールを定義することができる。組み込みルールには、２種類のタイプがある。

▷タイプ１・ルール

タイプ１・ルールは、クラスのインスタンスが、自分自身のみで満たさなければならないルールである。タイプ１・ルールは、クラスが展開されインスタンスが生成される時に適用される。

▷タイプ２・ルール

タイプ２・ルールは、タイプ・１ルールと異なり、複数のクラスのインスタンス間でチェックされるべきルールである。タイプ２・ルールは、そのインスタンスを使うクラスが展開されインスタンスを生成する時に適用される。

４．２．組み込みルール

▷ルール　width:レイヤの最小線巾を定義する。

▷ルール　space:レイヤとレイヤの間隔の最小値を定義する。

▷ルール　connect:ルールconnect は、ルールwidth のタイプ２版である。二つのインスタンス中の同じレイヤが接したり、交わった時の交わりの最小巾を定義する。

▷ルール　enclose:あるレイヤが、他のレイヤで囲まれて覆われていなければならないことを表わす。

▷ルール　extend: レイヤとレイヤがクロスする時、ある長さ以上飛び出していなければならないことを表わす。

４．３．ルール・オブジェクトにおけるインヘリタンス

ルール・オブジェクトにおけるインヘリタンスでは、セル・オブジェクトのインヘリタンスと異なり、同名のルールでも、下位のルールが優先されるということはない。つまり、インヘリタンスにより、スーパー・ルール・クラスの全てのルールも、チェックすべきルールと解釈される。

４．４．ルール定義の例

ＮＭＯＳの 2.5μルールのデザインルールを図７に示す。ルール width、enclose はルール・クラス@rule_recに定義されている。その他のルールは、ルール・クラス@rule_object に定義されている。

```
/*------------------*/
/*   rule object    */
/*------------------*/

'@rule_object'(rule,2,
        space,
           [[diff,diff,3],[diff,poly,1],
            [diff,ion,1.5],[poly,poly,2],
            [metal,metal,3],[diff,cut,2],
            [cut,cut,2]]).

'@rule_object'(rule,2,
        connect,
           [[diff,2],[poly,2],[ion,5],
            [metal,3],[cut,2]]).

'@rule_object'(rule,2,
        extend,
           [[diff,poly,2],[poly,diff,2],
            [ion,diff,1.5],[ion,poly,1.5]]).


/*----------------------*/
/*   rule rec object    */
/*----------------------*/

'@rule_rec'(super,'@rule_object').
'@rule_rec'(rule,1,
        width,
           [[diff,2],[poly,2],[ion,5],
            [metal,3],[cut,2]]).

'@rule_rec'(rule,2,
        enclose,
           [[diff,cut,1],[poly,cut,1],
            [metal,cut,1],
            [[diff+poly,1],cut,[2*2,1*2,1]],
            [[poly+diff,1],cut,[1*2,2*2,1]]]).
```

図７．2.5μルール定義例

５．ユーザ・インタフェース

５．１．ユーザ・インタフェースの実現

ＯＣＬＳは、高機能の個人用ワークステーション上で、高速なグラフィクス機能やポインティング・デバイス（マウスなど）を使用して稼働することを、念頭において設計されている。現在は、この様な設計環境のプロトタイプとして、VAX11/780 をホスト・コンピュータ、PC-9801 をフロント・エンドとして用いている。ＯＣＬＳ本体は、VAX11 上にC-Prologを用いて実装されている。グラフィクス機能、ウィンドウ機能、マウス制御は、PC-9801 上にBASIC を用いて実装されている。VAX11 とPC-9801 は、電話回線を用いてつながれる。

5．2．メニュー

ＯＣＬＳでは、セル・オブジェクトのメンテナンスを、マウスを用いたメニュー方式と、テキスト・エディタ（スクリーン・エディタ）を利用して行なう。セル・オブジェクトのメンテナンス時には、その時に行なうことができるコマンドを、メニュー・ウィンドウに提示し、マウスを用いてその中からコマンドを選択する。コマンド以外の情報は、キーボードより直接入力される。クラス・メンテナンス・モード・メニューの様子を図８に示す。

6．おわりに

ＯＣＬＳでは、知識型ＶＬＳＩ－ＣＡＤに適したセル・ライブラリを作るために、通常のオブジェクト指向型言語と異なり、性質の異なる２種類のオブジェクトを用意した。セル・オブジェクトとルール・オブジェクトを用いることにより、柔軟で、様々な情報を統一的に扱え、多くの特質を持ったセル・ライブラリが容易に実現できる。さらに、ＯＣＬＳの考え方は、ＶＬＳＩ設計を超えて、様々な分野に応用できる可能性を持っている。

参考文献

[1] 金井，坪井，石塚,"知識型VLSI-CADのためのセル・ライブラリ・システム ― ＯＣＬＳの概要",情報処理学会第30回全国大会,1985.

[2] C.ミード ,L.コンウェイ,"超ＬＳＩシステム入門",培風館,1981.

[3] J.Batali et al.,"The Design Procedure Language Manual", MIT A.I.Memo no.598,sep.,1980.

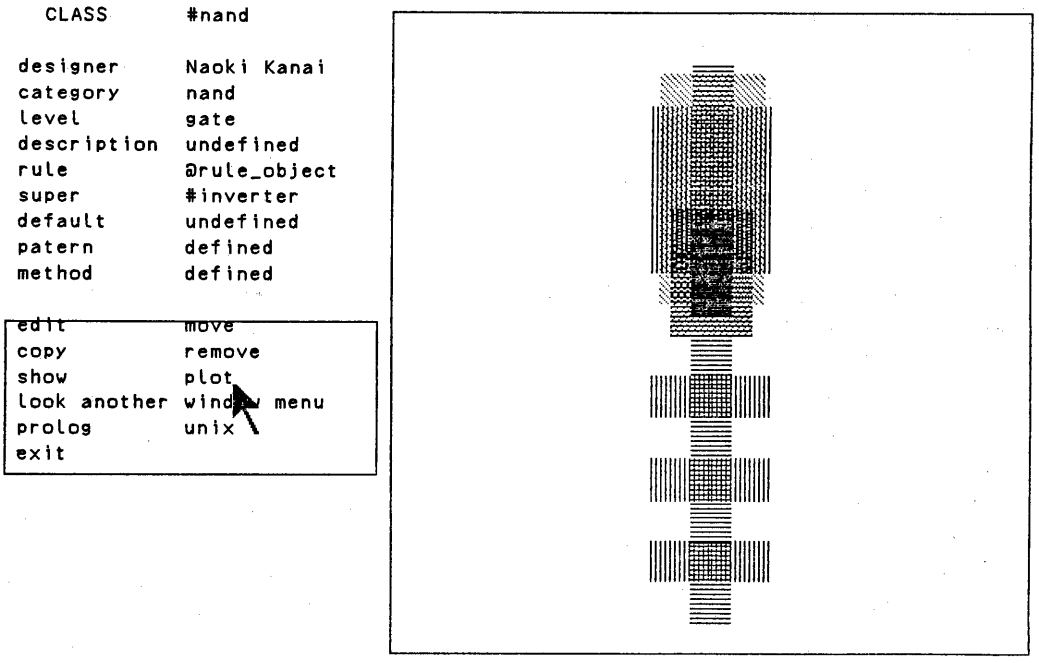

図８．クラス・メンテナンス・モード・メニュー