

ライブラリの使用法に関する知識を用いた プログラム合成

藤本典士† 永沢勇一†† 今中 武† 上原邦昭† 豊田順一†
†大阪大学産業科学研究所 ††関西大学工学部

プログラム部品の再利用に関する研究の一環として、オブジェクト指向言語 ESP を用いた、ウィンドウ作成プログラムを合成するシステム WINCS を開発した。WINCS は、計算機側から提供されているウィンドウ関連のクラスを継承し、プログラムを合成している。WINCS では、ウィンドウに関する様々な知識を用いて、これまでプログラマがプログラム作成時に行ってきた、マニュアルの検索作業を省略することが可能になっている。また、TMS を利用して、要求仕様に内在する矛盾を取り除いているため、ウィンドウプログラムを作成する際の制約条件に反した要求仕様を与えた場合でも、要求仕様に最も近いプログラムを合成することができるようになっている。

Window Class Synthesizer WINCS

Norio FUJIMOTO†, Yuuichi NAGASAWA††, Takeshi IMANAKA†,
Kuniaki UEHARA† and Jun'ichi TOYODA†

†The Institute of Scientific and Industrial Research

††Faculty of Engineering, Kansai University

†Osaka University, 8-1 Mihogaoka, Ibarakishi 567, JAPAN

††3-3-35 Yamatecho, Suitashi 564, JAPAN

This paper describes a program synthesis system, called WINCS. The aim of WINCS project is to develop a system which automatically synthesizes a window control class in object oriented language ESP. ESP provides *multi-inheritance* mechanism to derive new classes of objects from old ones in a rather convenient way. By using both the knowledge about window and multi-inheritance mechanism, WINCS can synthesize a desirable class from a programmer's requirement, selecting appropriate superclasses from a set of *ESP-provided classes*. Furthermore, TMS (Truth Maintenance System) is introduced in WINCS so as to detect and eliminate inconsistency in the requirement. Therefore, even if the programmer gives requirement which does not meet the constraint of inheritance, WINCS can synthesize an adequate window program.

1. はじめに

近年、ソフトウェア危機が叫ばれる中で、ソフトウェアの生産性を上げるために様々な努力がなされている。プログラムモジュールを部品として捉え、部品を再利用してプログラムを合成する方法もそのうちの1つである。部品を用いてユーザの要求を満たすプログラムを作成する場合、小規模のプログラム部品を数多く保持していなければならない。しかしながら、小規模のプログラム部品を組み合わせで再利用しようとする、部品間のインタフェースを全ての部品に共通なものとしなければならず、部品化が困難になるという問題がある。

このような問題を解決するために、我々は、オブジェクト指向言語を利用したプログラム合成の研究を進めている。オブジェクト指向言語では、プログラムモジュールをオブジェクトに対応させることができるために、部品化が容易であり、オブジェクト間のインタフェースも統一することができるという利点がある。また、オブジェクト指向言語の継承概念を利用し、容易に部品を組み合わせることができるという利点もある。

このような点に注目し、我々は、Prolog をベースとしたオブジェクト指向言語 ESP を利用したウィンドウプログラムを合成するシステム WINCS^[1]を開発している。

一方、ESP を利用してウィンドウを用いたアプリケーションプログラムを作成する場合、予め用意されているウィンドウ関係のプログラムモジュール(クラス)を利用することができる。しかしながら、プログラマは、プログラム作成段階で利用可能部品や、その利用条件を考慮し、マニュアルの中から適切なプログラムモジュールを検索しなければならない。このような問題に対処するために、WINCS では、マニュアルに記述されているウィンドウに関する知識を利用して^[5]、プログラマのマニュアル検索に代わる処理を行ない、プログラマの負担を軽減することが可能となっている。

また、プログラマがプログラムモジュールの利用条件に反した要求を行なった場合でも、適切なウィンドウ作成プログラム(クラス)を合成するために、TMS (Truth Maintenance System)^[2]を導入している。プログラムモジュール組み合わせ時に生じる様々な制約条件を前提とし、あるプログラムモジュールを利用するか否かということを仮定として TMS を起動すると、仮説集合内で矛盾した仮定を棄却し、新たな仮定を設定する操作が行なわれる。この

ような機能を用いて、WINCS は、制約条件を満たす範囲内で、最大限にプログラマの要求を生かしたウィンドウクラスを合成することが可能となっている。

2. ESP によるプログラムの作成

2.1 ESP のプログラミング環境

プログラミング言語 ESP は、逐次型推論マシン PSI 上で稼動する PROLOG をベースとしたオブジェクト指向言語である。一般にオブジェクト指向言語が継承機能を備えているように、ESP も多重継承機能を備えている。ESP では、この多重継承機能を利用して差分プログラミングという手法を用いることができる。差分プログラミングとは、システム側から提供されたものか、ユーザが作成したものかを問わず、既存のプログラム(クラス)をこれから作成しようとするプログラムの一部として継承し、独自に必要なプログラム部分のみを付加するという手法である。また、オブジェクト指向を利用してプログラムをモジュール化して作成することができるために、PROLOG では構築することができるという特徴がある。

このような環境でプログラミングを行なう場合、これまで以上に既存プログラムの管理が重要となる。たとえば、既にどのようなクラスが利用できるか、また、そのクラスを継承した場合にどのような効果が生じるのかなどを熟知していなければ、ESP のプログラム環境を十分に活用することはできない。更に、既存クラスのうち、ユーザの作成したものは別にして、少なくともシステムから提供されたクラスに関する知識は、プログラミングに際して必要不可欠である。しかしながら、ESP の利用可能なクラスは膨大なものであり、分野も広範であるため、一般のプログラマがすべてを熟知することは不可能である。また、プログラマが必要とする既存のクラスを検索するような機能は PSI に備わっていない。このため、プログラマは必要に応じて自分の要求に見合ったクラスをマニュアルの中から探し出さねばならない。このように、ESP によって大規模なプログラムを作成する場合、既存クラスの検索がプログラマの大きな負担となっている。

2.2 ウィンドウプログラム

ESP を用いてウィンドウを利用したアプリケーションプログラムを作成する場合、PSI の OS である SIMPOS によって提供されるウィン

```
T1 construct(one-window) ((T2))
T2 construct(lots-window)
```

T1 は分割されない 1 枚のウィンドウを作成するという仮定であり、T2 は 1 枚のウィンドウを複数のウィンドウに分割するという仮定である。ウィンドウを作成する場合、前者の方が一般的であるため、T2 が真であることが確信されない限り、T1 を真であると確信するということを T1 の支持リストが示している。つまり、仮定 T1 がデフォルトであるとも考えられる。^[4]

“継承競合知識” から導かれた仮定ノードには次のようなものがある。

```
C1 conflict(as-output, as-menu)
  ((T1, S1, select(as-menu))())
```

“as-output” と “as-menu” はウィンドウを分割しない限り競合を起こすという仮定であり、仮定 T1, S1, select(as-menu) が真であると確信されていれば、C1 自体も真であると確信される。

4.3.2 TMS による補足・競合解消処理

TMS では、選択クラスの補足・競合解消処理を TMS ノードの論理的整合性を維持するという処理によって行なっている。選択クラスを補足する場合、仮説群 1 のように“継承補足知識” から得られた仮定 J1 が利用される。J1 は、“mouse-input” と “as-menu” の両方が選択されている時に、真であると確信される仮定である。ここで “as-menu” のみが選択されている場合、J1 の理由づけがなくなり、矛盾 N2 が生じる。この矛盾を取り除くために、TMS は現在明白な支持がないために真であると確信されていない仮定を、真であると確信しようとする。その結果、S3 を真であると確信することになり、S4 が真であると確信されなくなる。その結果、矛盾 N2 を取り除くことができる。このような TMS の処理によって、クラス “mouse-input” が選択されるべきであることが判明する。

仮説群 1

```
J1 connect(mouse-input, as-menu) ((S3, S5))
N2 contradiction ((S4, S5), (C1))
S3 select(mouse-input)
S4 not select(mouse-input) ((S3))
S5 select(as-menu) ((requirement))()
```

```
S6 not select(as-menu) ((S5))
```

競合を解消する場合には、仮説群 2 のように“継承競合知識” から得られた仮定 C1 が利用される。C1 は、“as-output”, “as-menu” が選択され、ウィンドウを分割しないで実現することを真であると確信している時、それ自体真であると確信されることを示している。ここで、“as-output”, “as-menu” がプログラマに選択された場合、C1 が真であると確信され、矛盾 N3 が生じる。これを取り除くために、TMS は現在真であると確信されていない仮定を信じようとする。この候補となる仮定は、T2, S2, S6 であり、T2 を真であると確信した場合、4.1 で述べた共存法をとることになる。また、S2 かまたは S6 を真であると確信した場合、消去法をとったことになる。ただし、どの仮定を選択して真であると確信するかは、その時点での他の仮定の状態に依存している。

仮説群 2

```
C1 conflict(as-output, as-menu)
  ((T1, S1, S5))()
T1 construct(one-window) ((T2))
T2 construct(lots-window)
N3 contradiction ((S4, S5), (C1))
S1 select(as-output) ((requirement))
S2 not select(as-output) ((S3))
N1 contradiction ((S1, S2))()
S5 select(as-menu) ((requirement))
S6 not select(as-menu) ((S5))
N4 contradiction ((S5, S6))()
```

このように、TMS が仮説ノードの中から矛盾を取り除く操作が、選択クラスの補足・競合解消処理になっている。

5. WINCS の実現

5.1 システム構成

WINCS の機能範囲は、単にウィンドウクラスの合成のみにとどまらず、プログラムの合成過程で利用される知識の作成支援、及び合成されたウィンドウを利用する際のプログラムの負担を軽減することができる。このような機能を実現するために、WINCS は図 6 のように知識作成部、プログラム合成部、利用環境支援部によって構成されている。以下、各部の詳細な働きについて述べる。

ジョンプログラムを作成する場合、2章で述べたようなプログラムの負担がある。我々がWINCSを開発した目的は、このようなESPプログラムの負担を軽減することにある。WINCSでは、マニュアルと同等の知識を保持しているために、これらの作業をすべてシステム側で処理し、単に要求するウィンドウの性質のみを入力するだけで、目的のウィンドウを作成するプログラムを合成することができるようになっている。また、WINCSは完成したウィンドウ上で利用できるメソッド（関数）をオンラインでプログラムに提示することができるために、利用可能メソッドをマニュアルから検索する必要がなくなっている。

3.2 プログラム合成過程

WINCSは、図4のように、6段階でウィンドウを作成するプログラムを合成する。第1段階では、プログラムがどのようなウィンドウを要求しているのかをメニュー形式で入力する。第2段階では、要求された性質に見合ったウィンドウ関連のクラスをシステムが提供するクラスの中から選択する。この段階では「システムの提供するクラスがどのような要求を満たすのか」が記述された知識が必要となる。このような知識を「特性ークラス知識」と呼ぶ。第3段階では、第2段階で選択されたクラスだけでウィンドウを形成することが可能であるかを判定し、不足しているクラスを補足する。クラスの補足にはTMSを利用している。この段階では「ウィンドウを形成するには最小限どのようなクラスが必要であるか、また、各継承ク

ラスを単一で継承するだけでその機能を果たすか否か」を記述した知識が必要となる。この知識を「継承補足知識」と呼ぶ。第4段階では、第3段階で補足されたクラスの中で継承時に競合を起こすものがあれば、TMSを利用して競合を解消している。この段階では「どのようなクラス間で競合が生じるのか」を記述した知識が必要となる。この知識を「継承時競合知識」と呼ぶ。第3段階、第4段階におけるTMSの処理については4章で詳細に述べる。第5段階では、第4段階で得られた継承クラス群を指定された継承順序に並び替える。この段階では「どのような順序で継承すべきか」を記述した知識が必要となる。この知識を「継承順序知識」と呼ぶ。第6段階では、第5段階で整列した継承クラス群をESPプログラムの形式に従って組み込み、プログラムが要求するクラス（プログラム）を合成する。この段階ではESPプログラムの形式に関する知識が必要となる。この知識を「プログラム形式知識」と呼ぶ。以上のようにして6段階の処理を通じてウィンドウクラスを合成している。

3.3 知識ベース

3.3.1 特性ークラス知識

「特性ークラス知識」は、プログラムの要求とシステムが提供するウィンドウ関連クラスを結びつけるための知識である。現在、ウィンドウに対する要求はメニューとして用意したものに限定されており、要求とクラスが1対1に対応したものになっている。したがって、これらの知識はマニュアルにおける各クラスの機能に関する説明文を抜粋したものとしている。たとえば、クラス `as-superior` の機能は「子ウィンドウをもつための機能を提供する」と記述されており、これに対応する知識は、「子ウィンドウを持つための機能が必要ならば、クラス `as-superior` を選択する」という形式として表している。

3.3.2 継承時競合知識

同時に継承することのできないクラスを記述した知識が「継承時競合知識」である。「継承時競合知識」は、マニュアルの中で各クラスの継承方法の項目に記述されている制約であり、2通りの記述方法がある。第1は、同時継承できない具体的なクラスを列挙したもので、たとえば、クラス `with-label` では「`with-two-label` と同時に継承することはできない」と記述されている。第2は、同時継承できないクラ

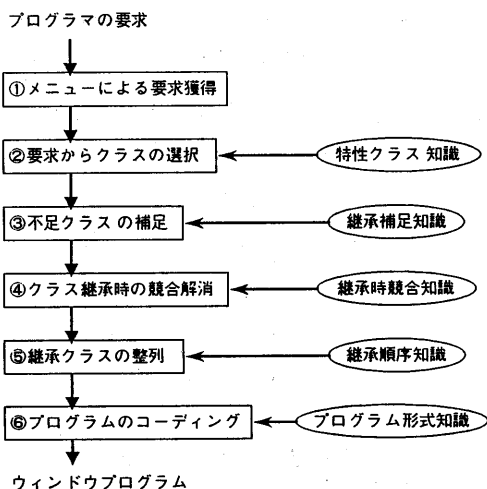


図4. WINCSのプログラム合成課程

ス具体的なクラス名でなく、クラスが属しているグループ名で記述したものである。たとえば、クラス `as-superior` は「出力グループやメニューグループと同時に継承することはできない」と記述されている。クラスグループは形式的に設定されたものであり、実際に継承クラスとして利用されるのは各グループに含まれるクラスであるために、この記述をそのまま知識として利用することはできない。したがって、この記述は知識①のように、出力グループ、メニューグループをそこに含まれる `as-output` などの具体的なクラスに分解した複数の知識として展開して利用している。

知識①

`as-superior` は、`as-output` と同時継承不可
`as-superior` は、`as-graphics` と同時継承不可
`as-superior` は、`as-menu` と同時継承不可

:

ただし、`as-output`, `as-graphics`, `as-menu` は、出力グループ、またはメニューグループに含まれるクラス

3.3.3 継承補足知識

この知識には、ウィンドウを作成するために必要最小限の継承クラスを記述した知識と、他のクラスと同時に継承しなければならないクラスを記述した知識がある。以後、前者を“最小クラス知識”、後者を“結合補足知識”と呼ぶ。“最小クラス知識”は必要不可欠なクラスを記述したもので、たとえば、知識②が“最小クラス知識”にあたる。また、マニュアル中の「メニューグループは、クラス `mouse-input` と同時に継承しなければならない」というような記述が“結合補足知識”にあたる。この記述は、“継承時競合知識”と同様に、グループを展開して知識③のような形式で利用される。

知識②

基本グループのうち、`inferior-window`, `popup-window`, `temporary-window` の1つは必要不可欠

知識③

`as-menu` は `mouse-input` と同時継承
`as-on-off-menu` は `mouse-input` と同時継承
`as-turning-menu` は `mouse-input` と同時継承

:

ただし、`as-menu`, `as-turning-menu`, `as-`

`scrolling-menu` は、メニューグループに含まれるクラスである。

3.3.4 継承順序知識

継承順序を規定しているのが“継承順序知識”である。この知識もマニュアルの継承方法に関する記述から抜き出したもので、“継承時競合知識”と同様にクラスグループを基にした記述と各クラスを基にした記述がある。たとえば、「基本グループのクラスは他のどのグループより後で継承する」という記述と「`as-scroll` は `as-output` より前に継承する」という記述がそれぞれ両者に相当している。“継承順序知識”は、“継承時競合知識”と同様にグループが展開され、知識④のような具体的なクラスの記述された形式で利用される。

知識④

`inferior-window` は他のクラスより後で継承
`popup-window` は他のクラスより後で継承
`temporary-window` は他のクラスより後で継承

3.3.5 プログラム形式知識

“プログラム形式知識”は ESP のプログラム形式を記述した知識であり、ウィンドウクラスをコーディングする際に用いられる。ESP のプログラム（クラス）は、図5のような階層形式をなしていると考えられる。木の根にあたる部分が ESP のクラス全体であり、葉の部分にあたるものが ESP における予約語やユーザ定義の記述などのプログラムコードに対応している。このような階層形式を知識⑤のようにファクト表現したものが“プログラム形式知識”の内部表現である。WINCS は、“プログラム形式知識”を根から深さ優先で探索し、葉に至った時点でプログラムコードに変換してい

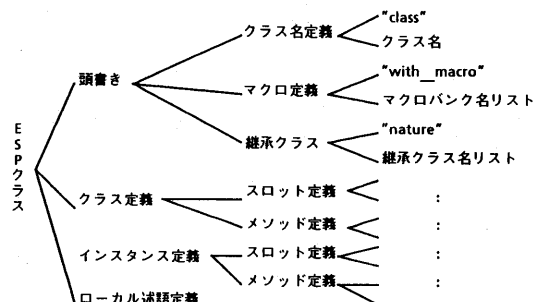


図5. ESPプログラムの階層形式

る。この方法でプログラムをコーディングすると、クラス形式の必要な部分にのみプログラムコードを割り当てることができるという利点がある。WINCSで作成するのは、継承クラスのみを記述したクラスオブジェクトであるため、継承クラスの部分のみが実際のクラスコードに書き換えられる。他の部分については、プログラマが必要に応じて、コードを割り当てることができる。

知識⑤

```
consist(program,[header,class-def,
               instance-def]).
consist(header,[name,macro,nature]).
consist(class-def,[slot,method]).
```

3.3.6 クラスメソッド知識

“クラスメソッド知識”は、各ウィンドウクラスを継承することで利用可能になるメソッドとその利用方法を記述した知識である。たとえば、クラス `as-graphics` では、

```
直線を描くメソッド    “draw-line”,
円弧を描くメソッド    “draw-circle”,
長方形を描くメソッド  “draw-rectangle”,
:
```

などが用意されており、メソッド “draw-line” は、

```
draw-line(対象オブジェクト、端点の座標、
           端点の座標)
```

という形式で使うことが記載されている。クラスメソッド知識は、クラス名を基にしてこの記述内容を検索できる形式にしたもので、知識⑥のような形式をしている。

知識⑥

```
method(as-graphics,draw-line,5,
        object,x1,y1,x2,y2)
```

4. TMS を利用した要求の補足と競合解消

4.1 プログラムの要求事項の補足と競合解消

3.2 で述べたように、プログラマの要求を基にして選択したクラスは、ウィンドウを作成するために十分なクラスではなく、補足や競合解消などの処理が必要になる。クラスの補足処理は、WINCSが一方的に補足すべきクラスを決定できる場合と、プログラマに補足すべきクラスを選択させなければならない場合に分けることができる。前者は“結合補足知識”を参照して

補足を行なう場合である。たとえば、“メニュー機能”を要求しているにもかかわらず“マウス入力”という性質を要求していない場合、知識①（“結合補足知識”）を参照すれば、“マウス入力”という性質を要求に付加することができる。後者は“最小クラス知識”を参照して補足を行なう場合である。たとえば、プログラマが基本グループのうちいずれのクラスも選択していない場合、知識②（“最小クラス知識”）を参照すれば、“inferior-window, pupup-window, temporary-window”のうち、いずれか1つを選択しなければならないことが判明する。しかしながら、このうちでいずれのクラスを選択するかはプログラマの要求に依存しており、最終的にはプログラマの選択に委ねなければならない。

補足クラスを最終的に決定する処理は異なるが、いずれの場合でも、補足すべきクラスを検索するためには現在のクラスの選択状況を把握する必要がある。このため、クラスの選択状況全体が管理できるシステムでなければ、補足処理を行なうことはできない。また、補足した結果が再び補足が必要としたり競合を起こしたりすることを考慮することは困難である。これは、補足処理、競合解消処理の連鎖の長さが不明であることから明らかである。このため、補足処理と競合解消処理を独立に行なうことはできず、両処理を統合的に行なわねばならない。

“継承時競合知識”を利用して発見した競合を起こすクラスの組に対して、競合解消を行うには、2種類の競合解消方法があると考えられる。第1の方法は、競合を起こすクラスの組から任意のクラスを削除する方法である（以後、この方法を消去法と呼ぶ）。たとえば、選択されたクラスの中から競合を起こすクラスの組として（`as-output`, `as-menu`）が発見された場合を考える。この場合、消去法で競合解消を行なうとすれば、“`as-output`”か“`as-menu`”のいずれか一方を削除しなければならない。したがって、WINCSはプログラマから最初に獲得した要求を最大限に実現するように、消去すべきクラスを選択する必要がある。つまり、“`as-output`”と“`as-menu`”のうち、いずれかを消去することによって、プログラマの選択した他のクラスへの影響が小さい方を選択しなければならない。たとえば、プログラマが他に“`as-menu`”と同時に継承すべきクラス“`as-single-select`”を選択している場合、“`as-menu`”を消去すれば“`as-single-select`”にも影響が生

じるため、“as-output”を消去すべきクラスとして選択しなければならない。要求の変更を最終的に決定するのはプログラマであるから、WINCS はどちらを消去することが最良であるかを提示し、最終選択をプログラマに委ねるようにしている。

第2の方法は、競合を起こすクラスを残したままで、別の条件を付け加えて、ウィンドウが作成できるように競合解消する方法である（以後、この方法を共存法と呼ぶ）。消去法と同様に、競合を起こすクラスの組として（as-output, as-menu）が発見されたと仮定する。競合の原因は、“出力機能”と“メニュー機能”を1つのウィンドウ内で実現しようとして生じたものであるために、2つのウィンドウで“出力機能”と“メニュー機能”を別々に実現すれば競合は起こらない。共存法では、クラス“as-superior”を継承することによって、1枚のウィンドウを複数の子ウィンドウに分割し、“グラフィック機能”と“メニュー機能”の両方を実現することができる。

上記のように、競合を解消するには2通りの方法がある。どちらの方法を選択すれば、要求を満たすものになるのかは、プログラマの真の要求に依存している。このため、WINCS は2通りの競合解消を行った結果をそれぞれ提示し、最終選択をプログラマに委ねるようにしている。

4.2 TMS の導入

クラスの補足・競合解消処理では、

- ①クラスの選択状況全体を検索して、補足・競合解消の対象となるクラスを発見しなければならない。
- ②補足処理が次の競合解消処理を引き起こし、競合解消処理がさらに補足処理を引き起こすことがある。
- ③競合解消のためのクラスの消去は、他の選択クラスに最も影響の小さいものを選択しなければならない。
- ④クラスの消去によって、連鎖的に他のクラスに影響を及ぼすことがある。

という点を考慮して処理を行なわねばならない。このような処理を行なうために、Jon Doyle の提案した TMS (Truth Maintenance System) を導入している。TMS は、与えられた仮説集合全体を管理し、仮説に変化が起こった場合、仮説集合内の論理的整合性を維持するため、非単調な推論を行なうシステムである。このような機能は、クラスの選択状況全体を管理

しながら統合的に行なわねばならない補足・競合解消処理に適している。WINCS では、“クラスを選択している”ということを仮定とし、“継承補足知識”、“継承時競合知識”を前提として TMS を起動している。その結果、前提や他の仮定に矛盾する仮定を棄却するという処理によって、プログラマが任意に選択したクラス群から、継承時の制約条件を満たすクラス群を決定することができる。

4.3 TMS による要求の修正

4.3.1 TMS ノードの生成

TMS 内部では、仮定や前提を TMS ノードで表現している。さらに、TMS ノードには仮定や前提を支持した理由を同時に記述している。TMS で利用される仮定には3種類あり、第1はクラスが選択されているか否かを仮定したもので、第2はウィンドウの実現方針を仮定したものである。また、第3は“継承時競合知識”、“継承補足知識”を仮定したものである。これらは本来、常に真理を保つものであるから前提となるべきである。しかしながら、たとえば、共存法を採用する場合のように、特定の条件下で真とならないこともあり得るため、これらの知識を仮定としている。

ここで3種類の仮定について例を示す。たとえば、クラス“as-output”が選択されているという事実からは、クラスの選択状況に関した以下の4つの仮定ノードが生成される。

```
S1 select(as-output) ((requirement)(M1))
S2 not select(as-output) ((S1))
N1 contradiction ((S1,S2)())
M1 wrong programmer
```

S1 は“as-output”が選択されているという仮定ノードである。さらに、プログラマの選択が正しくないという仮定 M1 が真であると確信されていないことと、プログラマの要求(requirement)とによって支持されていることを示している。S2 は“as-output”が選択されていないという仮定ノードであり、S1 が真であると確信されない限り S2 自体が真であると確信されることを示している。N1 は、S1 と S2 が同時に真であると確信されると、矛盾が起こることを示している。現時点では、S1 が真であると確信されているので S2 は真であると確信されていない状態にある。

ウィンドウの実現方針に関する仮定ノードは次のようなものである。

```
T1 construct(one-window) ((T2))
T2 construct(lots-window)
```

T1 は分割されない 1 枚のウィンドウを作成するという仮定であり、T2 は 1 枚のウィンドウを複数のウィンドウに分割するという仮定である。ウィンドウを作成する場合、前者の方が一般的であるため、T2 が真であることが確信されない限り、T1 を真であると確信するということを T1 の支持リストが示している。つまり、仮定 T1 がデフォルトであるとも考えられる。^[4]

“継承競合知識” から導かれた仮定ノードには次のようなものがある。

```
C1 conflict(as-output, as-menu)
  ((T1, S1, select(as-menu))())
```

“as-output” と “as-menu” はウィンドウを分割しない限り競合を起こすという仮定であり、仮定 T1, S1, select(as-menu) が真であると確信されていれば、C1 自体も真であると確信される。

4.3.2 TMS による補足・競合解消処理

TMS では、選択クラスの補足・競合解消処理を TMS ノードの論理的整合性を維持するという処理によって行なっている。選択クラスを補足する場合、仮説群 1 のように“継承補足知識” から得られた仮定 J1 が利用される。J1 は、“mouse-input” と “as-menu” の両方が選択されている時に、真であると確信される仮定である。ここで “as-menu” のみが選択されている場合、J1 の理由づけがなくなり、矛盾 N2 が生じる。この矛盾を取り除くために、TMS は現在明白な支持がないために真であると確信されていない仮定を、真であると確信しようとする。その結果、S3 を真であると確信することになり、S4 が真であると確信されなくなる。その結果、矛盾 N2 を取り除くことができる。このような TMS の処理によって、クラス “mouse-input” が選択されるべきであることが判明する。

仮説群 1

```
J1 connect(mouse-input, as-menu) ((S3, S5))
N2 contradiction ((S4, S5), (C1))
S3 select(mouse-input)
S4 not select(mouse-input) ((S3))
S5 select(as-menu) ((requirement))()
```

```
S6 not select(as-menu) ((S5))
```

競合を解消する場合には、仮説群 2 のように“継承競合知識” から得られた仮定 C1 が利用される。C1 は、“as-output”, “as-menu” が選択され、ウィンドウを分割しないで実現することを真であると確信している時、それ自体真であると確信されることを示している。ここで、“as-output”, “as-menu” がプログラマに選択された場合、C1 が真であると確信され、矛盾 N3 が生じる。これを取り除くために、TMS は現在真であると確信されていない仮定を信じようとする。この候補となる仮定は、T2, S2, S6 であり、T2 を真であると確信した場合、4.1 で述べた共存法をとることになる。また、S2 かまたは S6 を真であると確信した場合、消去法をとったことになる。ただし、どの仮定を選択して真であると確信するかは、その時点での他の仮定の状態に依存している。

仮説群 2

```
C1 conflict(as-output, as-menu)
  ((T1, S1, S5))()
T1 construct(one-window) ((T2))
T2 construct(lots-window)
N3 contradiction ((S4, S5), (C1))
S1 select(as-output) ((requirement))
S2 not select(as-output) ((S3))
N1 contradiction ((S1, S2))()
S5 select(as-menu) ((requirement))
S6 not select(as-menu) ((S5))
N4 contradiction ((S5, S6))()
```

このように、TMS が仮説ノードの中から矛盾を取り除く操作が、選択クラスの補足・競合解消処理になっている。

5. WINCS の実現

5.1 システム構成

WINCS の機能範囲は、単にウィンドウクラスの合成のみにとどまらず、プログラムの合成過程で利用される知識の作成支援、及び合成されたウィンドウを利用する際のプログラムの負担を軽減することができる。このような機能を実現するために、WINCS は図 6 のように知識作成部、プログラム合成部、利用環境支援部によって構成されている。以下、各部の詳細な働きについて述べる。

5.2 知識作成部

WINCS で利用する全ての知識を手手によってマニュアルを検索し、その内部表現を作成するにはかなりの負担がかかる。また、マニュアルの記述が不十分なこともあり、内部に矛盾を含まない知識を作成することは困難である。このため知識作成部では、プログラム合成部、利用環境支援部で利用する知識の内部表現を生成することを支援している。知識作成部では、

- ①マニュアルの記述を WINCS へ入力、
- ②入力ミス及び、マニュアルの不備による記述中の矛盾の発見とその修正、
- ③修正されたマニュアルの記述を内部表現に展開すること

が行われる。①では、WINCS の管理者が、マニュアル中のウィンドウ関連のクラスの記述を、WINCS と対話を進めながら投入していく。②では、PSI のライブラリ検索機能を利用して、ウィンドウ関連クラスを検索し、マニュアル記述の入力ミスがないかどうかの検査を行なう。また、マニュアル自体にも記述の省略があり、そのまま WINCS の知識として利用できないことがある。たとえば、メニューグループの各クラスのマニュアル記述には、「出力グループのクラスと同時継承不可能」という記述があるが、出力グループの各クラスのマニュアル記述には、「メニューグループのクラスと同時継承不可能」という記述はない。内容としては、両者とも「メニューグループの各クラスと出力グループの各クラスは同時継承不可能」という意味を持っている。しかしながら、4章で述べたように、TMS である特定のクラスの選択の可否を判定する場合、その他のクラスの選択状況が

判断の材料となっている。したがって、どのクラスを基にしても他のクラスとの関係が明白であるような知識が必要となる。このために、②では、マニュアルの記述を補足して、たとえば、

「メニューグループの各クラスは出力グループのクラスと同時継承不可能」、

「出力グループの各クラスはメニューグループのクラスと同時継承不可能」

というような、方向性を持った2つの知識に分解する処理を行なっている。この処理の詳細は他稿^[3]に譲る。③では、3.3 で述べたように、各知識のうちで、クラスグループで記述された知識を、そこに含まれる各クラスを基にした記述に展開し、それぞれの内部表現に変換する。①, ②, ③によって、WINCS のシステム管理者は、容易にマニュアルの記述から WINCS に必要な知識の内部表現を得ることができる。

5.3 プログラム合成部

プログラム合成部は WINCS の中核的機能である。この部分では、プログラムの要求獲得からウィンドウクラスの合成までを6段階に分けて処理する。①の要求獲得段階では、WINCS 側で予め用意したウィンドウに対する要求事項を、プログラマにメニューとして提示し要求を選択させる。たとえば、要求には「ウィンドウにラベルを付ける」、「メニューをつくる」などがある。この段階では、プログラマ自身の要求を取り入れるだけであるから、選択された要求には様々な矛盾が考えられる。②のクラス選択段階では、獲得した要求に見合ったクラスを選択する。この処理は、「特性-クラス知識」

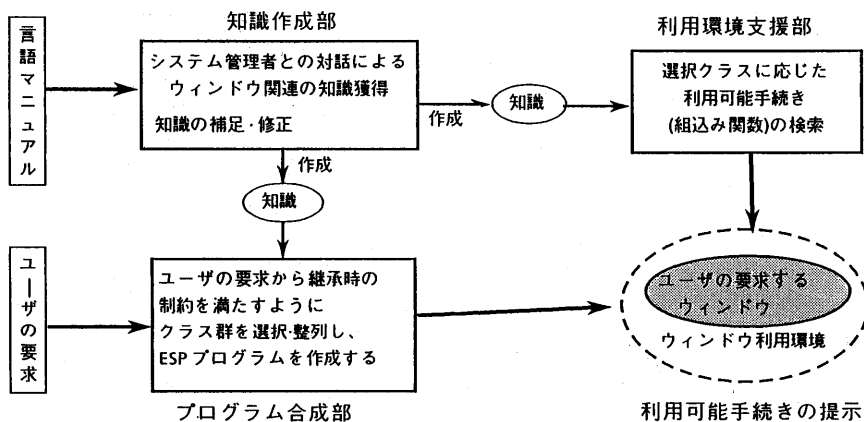


図 6. WINCS の全体構成

を参照して、要求から1対1にクラスを決定することが可能である。たとえば、「ウィンドウにラベルを1つ付ける」という要求に対しては、特性・クラス知識を用いてクラス `with-label` を選択する。③の TMS ノード設定段階では、4.3 で述べたように、ウィンドウの選択状況から仮定を生成し、それらに対応する TMS ノード（仮定ノード）を作成する。また、“最小クラス知識”、“結合補足知識”、“継承競合知識”に対応する TMS ノード（前提ノード）を作成する。④の TMS による選択クラスの修正段階では、③で生成された TMS ノードを基にして矛盾を取り除く処理を行い、理論的に整合性のとれた仮説集合を決定する。言い換えれば、プログラムの要求に最も近く、補足・競合解消処理が必要のないクラス集合を決定することになる。⑤のクラス整列段階では、“継承順序知識”を参照し、④の段階で決定したクラスを整列する。たとえば、`as-output`, `inferior-window`, `with-label` というクラスを継承することが決定した場合、知識④（継承順序知識）を利用してこれらのクラスを

- 1.with-label,
- 2.as-output,
- 3.inferior-window,

という順序に整列する。⑥のプログラムコーディング段階では、プログラム形式知識（知識⑤）を利用して、⑤の段階で決定した継承クラスをプログラムに埋め込む。その結果、図7のようなプログラムを合成することができる。

```
class my__window__class
  nature with__label,
         as__output,
         inferior__window;
end.
```

図7. WINCS で合成されたウィンドウクラス

5.4 利用環境支援部

システムの提供しているウィンドウクラスを継承してアプリケーションプログラムを作成した場合、作成されたウィンドウオブジェクトに対してどのようなメソッドが発行できるのかは、継承したウィンドウクラスに依存している。たとえば、`as-graphics` を継承して作成したウィンドウに対しては、“直線を描く”、“円弧を描く”、“長方形を描く”などのメ

ソッドが利用可能である。しかしながら、これらのメソッドは利用方法を含めすべてマニュアルに記載されているために、プログラマはメソッドの利用に際して再びマニュアルを検索する必要が生じる。利用環境支援部では、合成過程においてどのようなウィンドウクラスを継承したのかを基にして、クラス・メソッド知識を検索し、作成されたウィンドウに対して発行できるメソッドを提示することができる。たとえば、合成過程で `as-graphics` が継承された場合、`as-graphics` に関するクラス・メソッド知識（知識⑥）を参照して、利用可能メソッド `draw-line`, `draw-circle`, `draw-rectangle` を表示し、マウスを用いてこれらのメソッドをウィンドウに発行できるようになっている。

6. おわりに

本報告では、ウィンドウクラス合成システム WINCS の機能とその実現手法について述べた。ESP プログラマは、WINCS を利用することによって、計算機側からどのようなクラス（プログラム部品）が提供されているかを意識することなく、要求したウィンドウ作成プログラムを合成することができる。その結果、プログラマはマニュアルの検索などの煩わしい作業から解放される。また、TMS を用いてプログラマの要求を論理的に整合性の保たれた要求に修正することが可能である。このため、プログラマが論理的に整合性のとれない要求を行なった場合にも、要求を最大限に満たすようなウィンドウ作成クラスを合成することができる。

【参考文献】

- [1] 藤本他：ESP 言語上のウィンドウ制御プログラムの開発を支援するシステム WINCS, 情報処理学会第37回全国大会, pp.906-907 (1988).
- [2] J.Doyle: A Truth Maintenance System, Artificial Intelligence, Vol.12, No.3, pp. 231-272 (1979).
- [3] 永沢, 藤本他：プログラム合成システム WINCS のための知識獲得サブシステム, 情報処理学会第38回全国大会（発表予定）.
- [4] R.Nado and R.Fikes: Semantically Sound Inheritance for a Formally Defined Frame Language with Defaults, Proc. of AAAI-87, pp. 443-448 (1987).
- [5] 武田正之：知識ベースに基づく Language-oriented Editor, 情報処理学会論文誌, Vol.28, No.11, pp.1154-1161 (1987).