日本機械学会論文集(C編)
59巻560号(1993-4)

論文 No. 92-1049

対象モデルを適用したウィンドウ環境の構築*

長坂保美*1, 望月靖史*1, 大滝英征*2
石川義雄*2, 綿貫啓一*2

Building of Window based on Object-Model

Yasumi NAGASAKA, Yasushi MOCHIDUKI, Hideyuki OTAKI,
Yoshio ISHIKAWA and Keiichi WATANUKI

With the increasing application of large-scale and complicated systems in practical use, there is a need to develop systems on engineering work stations (EWS), in consideration of memory and speed of processing. However, it is very difficult to develop an application program with a graphical user interface (GUI) on an EWS. The authors have built a model of GUI based on the concept of object-model, and developed a function to create a source code of GUI by using this model. In this paper, knowledge representation of the model, the method for creating a source code of GUI by means of the model, and the method for controlling communication between the application and GUI are described.

Key Words: Model, Expert System, Artificial Intelligence, Object-Model, GUI, Window

1. 緒　　言

シミュレーションシステム (SS) やエキスパートシステム (ES) を開発する場合, エンジニアリング・ワークステーション (EWS) 上でのインタフェースの操作性や, 分散処理系でのデータベース, 知識ベースの有効利用が重要な課題となってくる. 特に, EWS 上のインタフェースは, SS や ES が実用化に向かうか否かの重要な役割を担う.

しかし, 従来のインタフェース (グラフィカル・ユーザ・インタフェース：GUI) 構築のためのツールキット, 例えば OPENLOOK (AT & T) や OSF/Motif (Open Software Foundation, Inc.) などは, ウィンドウ環境 (ウィンドウの画面構成の定義, ウィンドウとアプリケーション間の通信制御の定義, マウスの使い方, メニュー選択方法など), および X-Window から通信制御 (X 通信) に至る幅広い知識と経験が要求される. そのため, アプリケーション構築者(以後ユーザと呼ぶ) は, ウィンドウ環境を常に意識する必要があり, GUI を効果的に取入れたプログラム開発が非常に困難とされていた.

そこで, 著者は, 機械工学分野における EWS 上のシステム (CAD/CAM, AI など) との結合を考慮した場合, インタフェースとして要求されるウィンドウ環境を検討した. その結果, ウィンドウ環境をモデル表現化し, アプリケーション開発が容易で, しかもウィンドウ操作に一貫性があるようにしておくのが効果的であることを見出した. そこで, 知識工学における対象モデル[1] のモデル表現を用いて, ウィンドウ環境をモデル化し, ウィンドウとアプリケーション間の通信制御を規定することにより, 複雑なウィンドウ環境を意識することなく, アプリケーション構築を可能とするようにした.

本稿では, これらの具体例について述べる.

2. 対象モデル表現による GUI システム

2・1 システム構成　図 1 は, 本システムの構成を示したもので, GUI 生成 (GUI-Creation) システム, 通信補助機能 (Sub-Com. Function), および GUI モデル表現(GUI-Modeling)から構成されている. なお, 本システムでは, 複数の EWS (sun/SPARK-stational, news/NWS-1750, etc) がイーサネットを介して結合され, UNIX 上の X-Window 環境下で互い

* 原稿受付　平成 4 年 7 月 9 日.
*1 埼玉大学大学院 (〒338 浦和市下大久保 255).
*2 埼玉大学工学部.

にメッセージ交換が行われる．

GUIモデル表現は，インタフェースとしてのウィンドウ環境を「機能」と「構造」でモデル化している．このモデル表現は，フレーム表現を基本とするAIツール[(2)~(5)]（「ZES」と呼んでいる）上に構築した．そのため，ZESが有しているエディター機能により，ウィンドウ環境の構築が容易である．

通信補助機能は，アプリケーションによるプロセスとウィンドウ環境によるプロセス間の同期を取るための機能で，2•7節で述べている．

また，GUI生成システムは，GUIモデル表現を基に，アプリケーション側にウィンドウ環境に関するリソース（ソースコードも含む）を提供するシステムである．そして，アプリケーション起動時に，このリソースをウィンドウ側に組込み，アプリケーションとの橋渡しを行う．

なお，GUI生成システムと通信補助機能は，HP社のToolkit（Xw）上に構築されている．

2•2　ウィンドウ環境のモデル表現　図2は，ウィンドウ環境の構成例を示したものである．角枠がウィンドウを，丸枠がそのウィンドウ内のコマンド（処理）を示している．図例では，ウィンドウ「A」がウィンドウ「B」，「C」に移る処理と，「A-1」の処理を行うことを示している．

図3は，フレーム表現の枠組みを用いた対象モデルにより，図2のウィンドウ環境のモデル表現を示したものである．図中の実線枠は，対象モデルの概念[(1)]における「構造」を示しており，2•4節で述べるウィンドウ生成に関する表現が記述されている．一点鎖線枠は「機能」を示し，2•5節で述べるウィンドウ環境のコマンド生成に関する表現が記述されている．

一方，図中の点線枠「Meta-Level Representation」は，ウィンドウ環境の上位概念を示しており，インスタンス・ユニットで定義されたウィンドウ環境を具現化するためのモデル表現が格納されている．これは，2•3節で述べるウィンドウ•テンプレート，マウスの使い方（左右ボタンの定義），2•4，2•5節で述べるモデル表現などが相当する．

図4は，図3のウィンドウ環境のモデル表現を可能にしている「has-window」と「has-command」の関係を示したものである．「has-window」はウィンドウ環境の包含関係を示し，図中のスロット名「window-id」の値で具体的に管理される．

一方，「has-command」は，図中の「command-of」と共に，ウィンドウ環境の「構造」と「機能」を表現するための主従関係を示している．「has-command」は「機能」を，「command-of」は「構造」を示すユニット

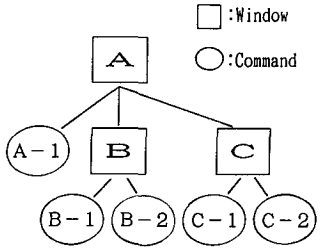


図 2　ウィンドウ環境の構成例

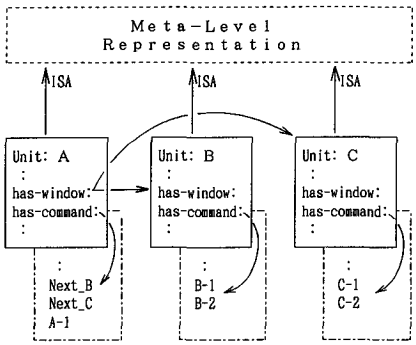


図 3　ウィンドウ環境のモデル表現

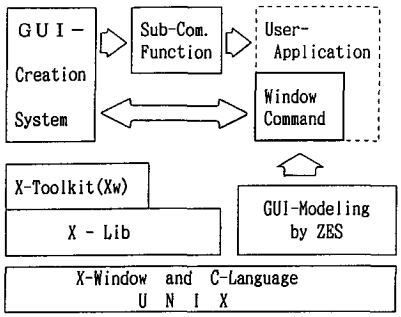
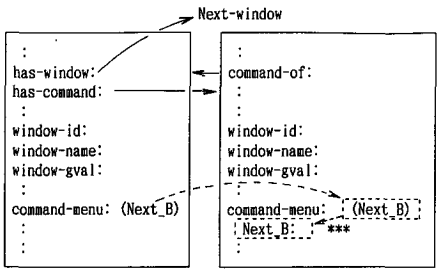


図 1　本システムの構成



（a）「構造」ユニット　　（b）「機能」ユニット

図 4　「has-window」と「has-command」の関係

を指している．例えば，この関係は，(a)ユニット内のスロット名「command-menu」に，値「(Next_B)」が記述されると，(b)ユニット内の点線枠で示すようなスロットの値を自動的に記述し，新たなスロット名「Next-B」も生成する．この新たなスロット名は，「構造」の記述に伴う「機能」を記述する領域である．また，この逆も同様である．つまり，図3のモデル表現は，この包含関係と主従関係によって管理されている．

このようにして，図3のインスタンス・ユニット内に，目的とするウィンドウ環境が記述（構築）される．そして，上位概念のモデル表現によって，ウィンドウ環境に関するソースコードが生成され，ユーザ・アプリケーションに組込まれる．それゆえ，ユーザは，容易に，かつプログラムミスも少なくアプリケーションを作成できる．

2・3　ウィンドウ構成　図5は，ウィンドウ構成の基本的な枠組み（ウィンドウ・テンプレート）を示したものである．

ウィンドウ・テンプレートは，ウィンドウ名「Window-Title」，アプリケーションからの情報を示すメッセージ表示「Window-Message」，実行可能なコマンドを示すコマンドメニュー「Command-Menu」，テキスト入力およびそのスクロールが可能なテキストエディター「Text-Editor」から構成されている．

アプリケーション側では，ウィンドウ・オープン時に，図6に示したような手順で具体的な構成を逐一決定していく．まず，ウィンドウ・オープンのメッセージをウィンドウ側に送信(1)し，ウィンドウ側からオープンOKのメッセージを受信(2)する．次に，ウィンドウ識別子「Window_ID」を送信(3)する．この識別子は，ウィンドウの呼び出しやクローズなどに利用される．次いで，ウィンドウ構成「Window_Format」がウィンドウ側に送信(4)される．このウィンドウ構成は，以下の構成になっており，2・4節で述べるモデル内部表現のスロットの値で決定されるものである．

（タイトル名　テキストエディター
　　メッセージ表示
　　コマンドメニュー
　　アイコン情報 ...）

このように，本システムのウィンドウの画面構成は，ウィンドウ・オープン時に，アプリケーション側からのウィンドウ構成「Window_Format」によって動的に決定される．

2・4　モデル内部表現とウィンドウ生成　図7は，図5のウィンドウ・テンプレートを決定するモデル内部表現，およびウィンドウ側でのウィンドウ生成の過程を示したものである．ここで，ウィンドウ生成の過程を図中の番号に沿って述べる．

(1)　モデル内部表現：　2・3節のウィンドウ構成「Window_Format」は，スロット名「format」の値「S-format」[4]と，内部表現のスロット値で生成される．そして，この構成は，内部表現のスロット名「window-gval」の値「＊top-menu＊」に結合される．なお，モデル表現内のスロット名「window-message」の値「nil」は，メッセージ表示の枠組みが無いことを指している．また，スロット名「command-menu」の値を「nil」にすると，テキスト表示のみのウィンドウが生成される．スロット名「window_icon」は，複数のウィンドウが混在することを避けるため，不必要なウィンドウをアイコン表示するための情報である．

(2)　ウィンドウ構成の組込み：　(1)により，大域変数「＊top-menu＊」に結合されたウィンドウ構成は，アプリケーション内に組込まれる．これによって，アプリケーション側でのウィンドウ・オープン時に，この大域変数「＊top-menu＊」がウィンドウ側に送信されることとなる．

(3)　リソースの生成：　ウィンドウ側では，ウィンドウ・テンプレート内のウィンドウ名や大きさ，テキストエディターの有無など，ウィンドウ生成のためのリソースが随時決定される．

(4)　ウィンドウの生成：　上記で決定されたリソースは，ウィンドウ構造のデータタイプ「Window

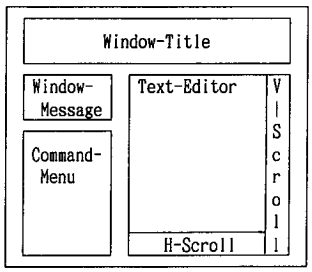


図5　ウィンドウ・テンプレート

	Application side		Window side
(1)	send OPEN_Window	→	receive Message
(2)	receive Message	←	send OPEN_OK
(3)	send Window_ID	→	receive Data
(4)	send Window_Format	→	receive Data
(5)	receive Message	←	send Window_OK

図6　ウィンドウ・オープン時の制御手順

data」に組込まれる．そして，ウィンドウ生成の関数「XtCreateApplicationShell」のリソースとして，実際のウィンドウが生成される．

このようにして，同時に複数のウィンドウ環境を生成することを可能にしている．すなわち，従来のような一つのウィンドウ構成の枠組みに対して，一つのソースコードを必要とすることがなくなる．それゆえ，ともすると大容量化する傾向にあるインタフェースに対し，小容量でのアプリケーション開発が可能である．

2·5 モデル内部表現とコマンド生成 図8は，ウィンドウ環境についてのモデル内部表現，およびコマンド生成の過程を簡略に示したものである．ここで採用しているコマンド（ソースコード）生成過程について，ウィンドウ環境に焦点を絞った場合の過程を追ってみる．

（1） コマンド生成のテンプレートを参照： モデル表現内のスロット名「language」の値は，コマンド生成のテンプレート（ソースコードを生成するための枠組み）が示されている．そして，ISA関係を用いて上位概念に記述されているテンプレートの参照が行われる．

（2） テンプレート内の変数値を参照： テンプレート内の変数（？で始まる文字列）は，「？」を除けばスロット名を示している．そこで，インスタンス・ユニット「top-command」のスロットを参照することによ

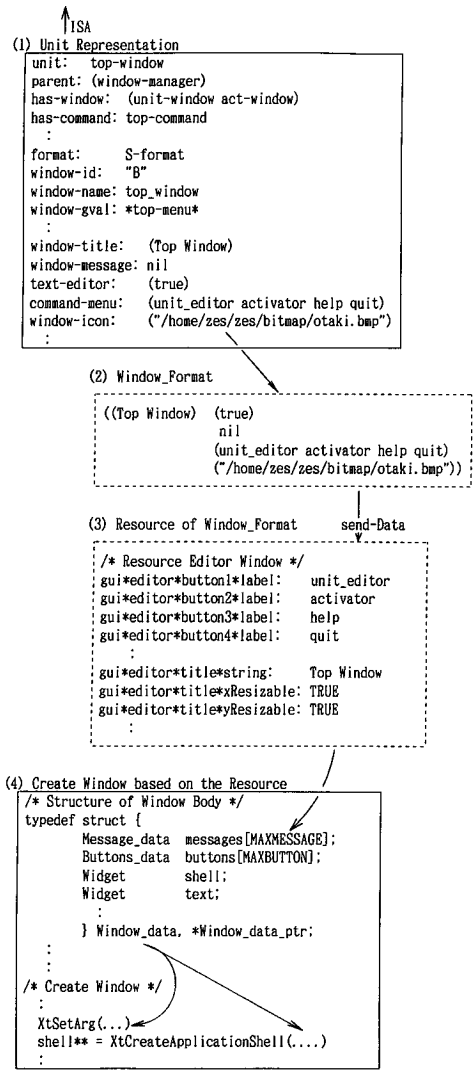


図 7 モデル内部表現とウィンドウ生成の過程

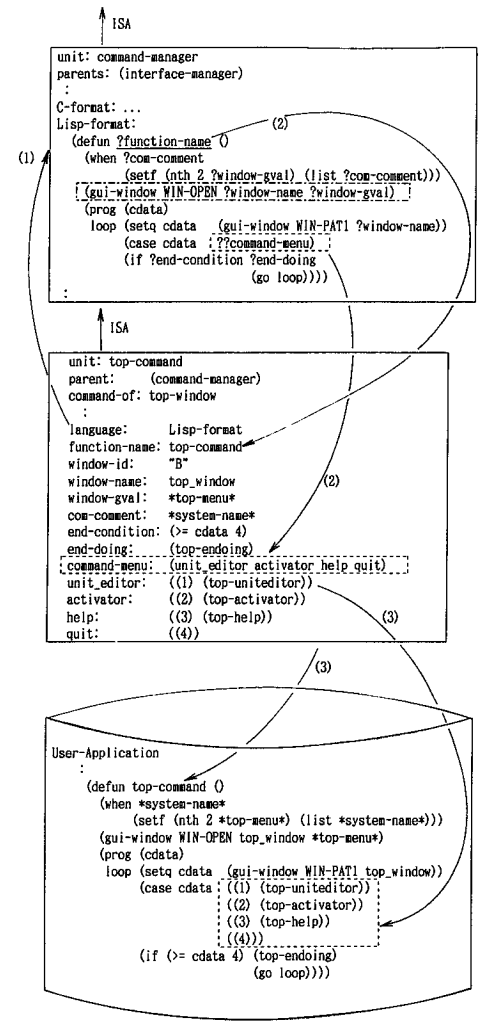


図 8 モデル内部表現とコマンド生成の過程

り，テンプレート内の変数が決定される．ただし，テンプレート内の点線枠の変数「？？command-menu」は，
スロット名「command-menu」のスロット値「(unit_edit 「(unit_editor activator…)」 or activator..)」を，再度参照するスロット名として決定される．なお，一点鎖線枠の関数「gui-window」は，2・6節で述べる．

（3） ソースコードの組込み： ソースコードが生成され，アプリケーション内に組込まれる．

このようにして，該当する言語（例ではLisp言語）にソースコード化される．なお，本システムでは，C言語とLisp言語が用意されているので，UNIX上での大方の言語が用意されていることになる．また，スロット名「function-name」は，アプリケーション内で呼び出すための関数名を示している．

このように，ウィンドウ環境に関する知識は全く意識することなく，必要に応じた言語にソースコード化されるので，ユーザは従来の入出力命令を呼び出すように，指定した関数名を呼び出せばよい．

2・6 ウィンドウ制御機能　図9は，アプリケーション側に組込まれるウィンドウ制御機能に関する関数「GUI-WINDOW」を示したものである．ここでは，Lisp言語の場合を示しているが，C言語の場合も同様な関数が提供される．

関数「GUI-WINDOW」の引数「Msg_ID」によって，ウィンドウ側との制御条件が設定される．例えば，「WIN-OPEN」の場合は，2・3節で述べたウィンドウ・オープンの制御を実現する．また，「WIN-PAT1」は，ウィンドウ識別子「Wind_ID」をウィンドウ側に送信し，ウィンドウ側のマウス・クリック入力の制御を実現する．この他，ウィンドウ環境外のアプリケーションを直接起動する制御「WIN-PAT2」，テキストエディターからの入力の制御「WIN-PAT3, WIN-PAT4」，他のシステムなどとの分散処理の制御「WIN-PAT6～WIN-PAT9」（例えば，ウィンドウ画面から市販のCADシステムの起動やデータ入力[5]など）も組込まれている．なお，図10の制御方法は，2・7節で述べる通信制御に基づいて，ソースコード化されている．

```
(defun GUI-WINDOW (Msg_ID &optional (Win_ID nil) (S_Data nil))
  (prog ((Rdata nil))
LOOP (send-message Msg_ID)
     (when (/= Msg_ID (receive-message))
           (print "??? system error ???")
           (return nil))
     (cond ((eq Msg_ID WIN-OPEN)  (send-data Wind_ID)
                                  (send-data S_Data)
                                  (setq Rdata (receive-message)))
           ((eq Msg_ID WIN-CLOSE) (send-data Wind_ID)
                                  (setq Rdata (receive-message)))
           ((eq Msg_ID WIN-PAT1)  (send-data Win_ID)
                                  (setq Rdata (receive-message))
                                  (when (= Rdata 0)
                                        (go LOOP)))

           ((eq Msg_ID WIN-PAT2) ..) ;Perform Application Program
           ((eq Msg_ID WIN-PAT3) ..) ;Input Text (paragraph)
           ((eq Msg_ID WIN-PAT4) ..) ;Input Text (not paragraph)
           ((eq Msg_ID WIN-PAT5) ..) ;Recognize Current-ID
           ((eq Msg_ID WIN-PAT6) ..) ;Send Data to other application
           ((eq Msg_ID WIN-PAT7) ..) ;Send Message to other application
           ((eq Msg_ID WIN-PAT8) ..) ;Receive Message from other application
           ((eq Msg_ID WIN-PAT9) ..) ;Receive Data from other application
           (t   (setq Rdata 0)))
     (return Rdata)))
```

図 9　ウィンドウ制御機能に関する関数

2・7 通信制御の方法　本システムの通信制御では，ウィンドウとウィンドウ内のコマンドとの同期を図り，さらに2・6節で述べたように他システムとの同期も図る必要がある．そこで，本システムの通信制御の方法は，ソケット通信（クライアント・サーバモデル）を用いた非同期によるメッセージ伝達の方法[6]を用いることにより，他のシステムとの同期を可能にした．

図10は，本システムで採用した通信制御（ソケット通信とXプロトコル通信）の概要を示したものである．点線枠はXプロトコル通信を，二点鎖枠はソケット通信による伝達を示している．

本システムの具体的な通信制御は，非同期のメッセ

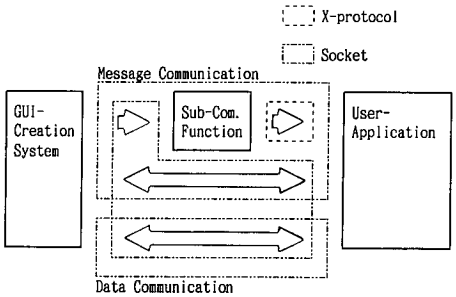


図 10　本システムの通信制御の概要

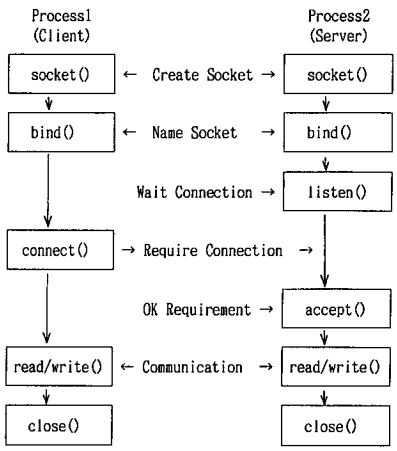


図 11　ソケット通信の生成と送受信の手順

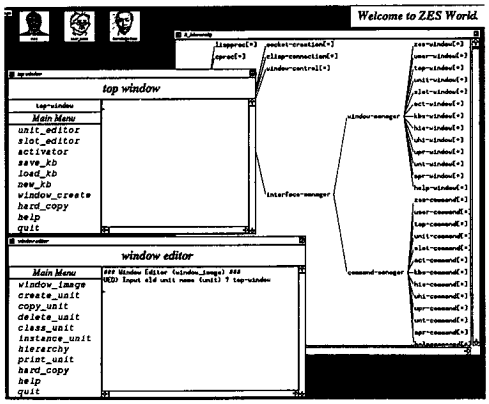


図 12 本システムの実例(モデル表現の構築例)

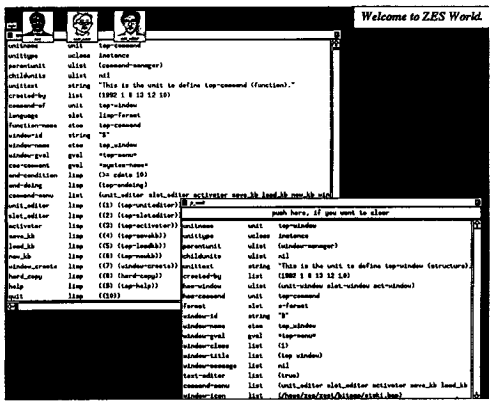


図 13 本システムの実例(モデル内部表現)

ージ伝達(通信補助機能「Sub-Com. Function」:X プロトコル通信)でプロセス間の同期を図り,実際のデータの送受信(ソケット通信)を行っている. 例えば,2・2 節で述べたウィンドウ・オープン命令を参照してみると,通信(1),(2)のメッセージ通信で同期を取り,通信(3),(4)でプロセス間のデータ通信が行われ,最後に通信(5)で通信の終了を示している.

図 11 は,本システムにおけるクライアント・サーバモデルを実現するソケット通信の生成と送受信の手順を示したものである.まず,ソケットの生成,ソケット名称の定義が行われる.次に,クライアント側では,相手プロセスとのソケット結合を要求する.サーバ側では,ソケット結合要求の待ち状態「listen」に入り,要求を受け取ると,その要求結果を返す「accept」.そして,具体的なデータの送受信が行われる.

このように,本システムの通信制御の方法は,プロセス間の同期を図る他に,同期型メッセージ伝達における処理の中断や,非同期型におけるデータ欠落といった問題,さらにデータ通信のためのバッファ(容量的な)不足という問題も解決している.

2・8 本システムの実例 図 12 は,ZES 上に提供されている GUI モデル表現の構築例を示したものである.図の右部は図 3 のモデル表現の階層構造を示したものであり,図の下部はモデル表現のためのエディターを示したものである.そして,図の左上部がこのエディターで構築されたモデル表現によるウィンドウの画面である.

図 13 は,図 12 のエディターで構築している際のモデル内部表現を示したものである.図の左部はウィンドウ環境のウィンドウ構成の枠組み「機能」を,右部がウィンドウ環境のコマンド「構造」を示している.

図 14 は,図 9 の分散処理の制御を用いて,市販の

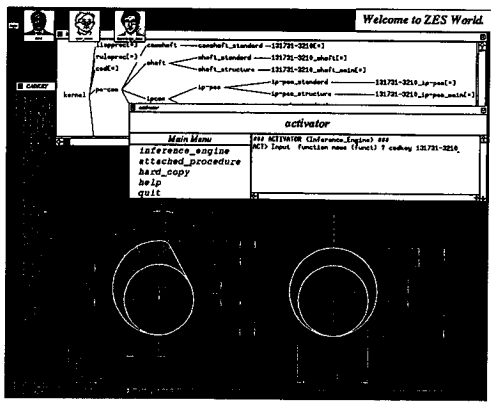


図 14 本システムの実例(他システムとの起動)

CAD システムを AI ツールのチャイルド・プロセスとして起動している例を示したものである.CAD システム中の転送コマンドにより,CAD の図面情報が AI ツール内に読み込まれ,他システムとの同期が取れるようになっている.

3. 緒 言

ここで,本研究をまとめてみると以下のようになる.

(1) ウィンドウ環境のモデル表現により,GUI のツールキットに関する複雑な知識を有せずとも,ウィンドウ環境を容易に生成することができる.そのため,アプリケーションの構築性が大いに増す.

(2) ウィンドウ環境のリソースの提供で,同時に複数のウィンドウ環境を生成することを可能にした.これにより,大容量化の傾向にあるインタフェースの開発の問題や,そのデバッグの難しさなどに対する解決策の一助となるものと期待される.

（3） モデル表現では，C（手続き型）言語と Lisp（宣言型）言語を対象としている．それゆえ，言語にとらわれることなくウィンドウ環境を生成できる．

文　献

（1） 上野，電子通信学会技法，AI 86-4(1986)，21-28.
（2） 長坂・ほか2名，機論，58-547，C(1992)，975.
（3） 長坂・ほか2名，機論，58-549，C(1992)，1680.
（4） 長坂・ほか3名，機論，58-552，C(1992)，2560.
（5） 長坂・ほか3名，設計工学会会誌，27-12（1992），537-542.
（6） 三田，応用C言語，アスキー出版，(1991)，279-294.