ソフトウェア工学 53－2
（１９８７．５．２７）

# 複数言語システム下でのユーザインタフェース構築支援環境

松田裕幸　波多野豊治　松永祥嗣　渡辺正信*

日本電気技術情報システム開発（株）
日本電気（株）*

ユーザインタフェース構築支援環境を開発中である。あるシステムのユーザインタフェースを構築しようとする場合、本支援環境が提供するプリミティブを組み合わせることによって、ユーザニーズに合ったようにインタフェースをカスタマイズすることが容易となる。また、プリミティブは特定のプログラミング言語に依存しないプロトコルによって利用できるようにしてあるため、複数のプログラミング言語によって構築した複数のシステムが同一のインタフェースを共有することも可能となる。現在はプロトコルを通してウインドウを操作できる機能までを実現している。

# Language Independent Building Tool for User Interfaces

Hiroyuki MATSUDA, Toyoharu HATANO, Yoshitsugu MATSUNAGA, Masanobu WATANABE*

NEC Scientific Information System Development, Co.

NEC Co.*

We have been developing a user interface building tool. We provide builing primitives, so you can easily customize your systems' interfaces when you use the primitives to develop the systems. These primitives are accessed by language independent protocols, so systems built by more than one programming languages can share the same interface which uses the same protocols. We have realized some window control primitives now.

## １．はじめに

現在、非常に多くの応用システムがマルチウインドウ、マウス、ビットマップディスプレイ等を利用した視覚的なユーザインタフェースを、システムとユーザとの間の円滑なコミニュケーションのために活用している。ユーザインタフェースはユーザがシステムと情報を交換する場だけではなく、”使いやすさ”という人間が主観的には理解しているが必ずしも定量的には定義できない性質も同時に必要としている。むしろ、”使いやすさ”がインタフェースの好し悪しを左右することも多く、ひいてはシステムの稼働率を高める結果にもつながる。しかしながら、”使いやすさ”という概念は多数の論文で議論されてきてはいるが未だに主観の域を出ることはない。　従って著者らは、インタフェースを構築するシステム設計者が”使いやすい”と信じるインタフェースを提供するのではなく、”使いやすい”とユーザが思えるまでインタフェースを段階的に構築、あるいは変更、あるいは再構築するのが容易なインタフェース構築環境を提供することが、現時点での自然な提案であると考える。さらに言えば、この環境はシステム設計者のみならず、システムを利用する一般ユーザにも利用できるような形態になっていればなお望ましい。

本報告ではユーザインタフェース構築支援環境として現在開発中のシステムを紹介する。このシステムは、言語ＬＩＳＰを用いて複数の応用システムとそのインタフェースを構築してきた筆者らの経験を参考にして設計されている。これらの応用システムはほとんどが研究レベルの段階にあり、またユーザとシステム設計者はともに同じ人であるという場合も多い。従って、ユーザインタフェースを言語Ｃなどを駆使して短期間に設計・構築し、場合によってはユーザでもある我々自身の手で再設計や再構築を行なうことは容易ではない。特に、最近普及の著しいワークステーションが提供するウインドウシステムやビットマップディスプレイなどを有効に活用した視覚的なユーザインタフェースを構築するには、理解しなければならない事柄が非常に多い。現在、本システムは言語レベルでのインタフェース構築プリミティブを提供している。これらはプロトコルの形式で利用でき、既成のウインドウシステムを簡便に使いこなすことを目指したものである。なお、この段階でのシステムを本稿中で引用するにあたっては”ＸＸ”という名称を用いることにする。

## ２．システム概要

本章では　１）ＸＸとのインタフェースの基本となるプロトコル体系とその処理プロセス，２）ＸＸを用いる応用システムとＸＸとの間での各種情報のやりとりを制御する機構，３）ＬＩＳＰ環境下でのＸＸの利用方法の３点について述べることにする。

### ２．１プロトコル体系とその処理プロセス

プルトコルは非常に簡単な形式になっており、例えばウインドウを１つ作るとしてそのベースフレームを用意するための形式は次のようになる。

WB:left-top-x:left-top-y:width:height:label!.　　－(1)

ここでWBC は数あるプロトコルを分類するための識別子の役目を果たしている。left-top-xからlabel までには具体的な値が入る。各項目は記号：で区切られプロトコルの終了は記号例！．で指示する。

このプロトコルを実際に特定の言語から用いる時はどうするか？

ＬＩＳＰを例にとってみると、XX-CREATE-BASE-WINDOW なる関数を用意する。しかも、この関数は以下に示すように(1) のプロトコルと１対１に対応する簡単なコーディングとなる。

```
(defun XX-CREATE-BASE-WINDOW (x y w h label)        -(2)
    (XX-string-eval
        (XX-send-recv (XX-set-protocol "CBW"  x y w h label))))
```

プレフィックスXXがついた関数はシステムＸＸが用意するLISP向け関数である。例えば関数XX-set-protocol は全ての引数から文字列"CBW:0:0:100:50:test-windowl."を作り出す。ここでは、x からlabel までの具体値として0 から"test-window" を想定している。この関数はＬＩＳＰで２－３行のプログラムである。

先にも述べたようにシステムＸＸのプロトコル体系は非常に簡単であり従って、従来の同様なシステムにも多く用いられてきたと考えられる。しかしながら複雑なデータ構造を多用し，かつ独立したプロセスとして存在するウインドウシステムを単純なプロトコルだけで制御することは容易ではない。しかし(2)の関数定義からもわかるようにプロトコルの処理プロセスは以下の３スッテプのみで済んでいる。

（i）　プロトコルの用意　　(XX-set-protocol)

（ii）プロトコルをシステムＸＸに送り、返事を待つ　　(XX-send-recv)

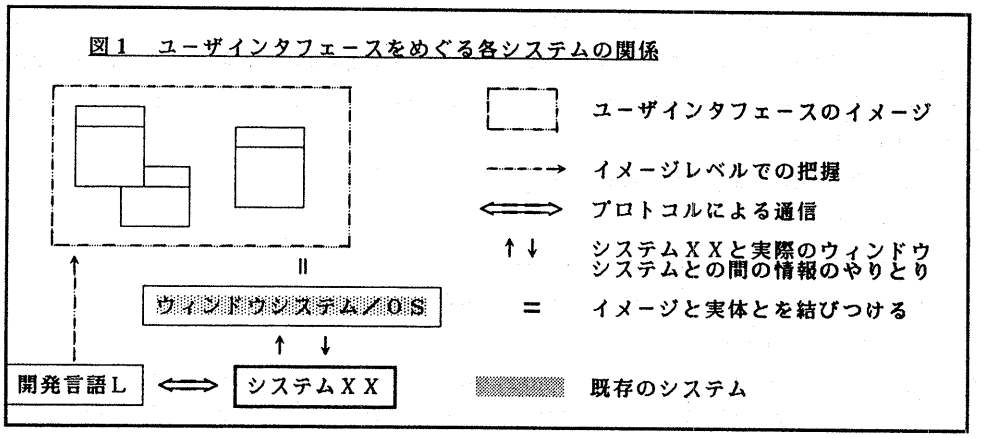
（iii）返値を評価する　　(XX-string-eval)

これはすべてのプロトコルに対して共通である。プロトコル送信後、返事を待たなくてよい時はXX-sendという関数を用い、従ってこの時は(iii）の返値を評価する部分は不要である。

今はＬＩＳＰの場合について述べたが、(2）に示したようなプログラミング言語レベルでのプロトコル記述の簡便さは他の言語の場合でも大きく損われることはないと考えている。

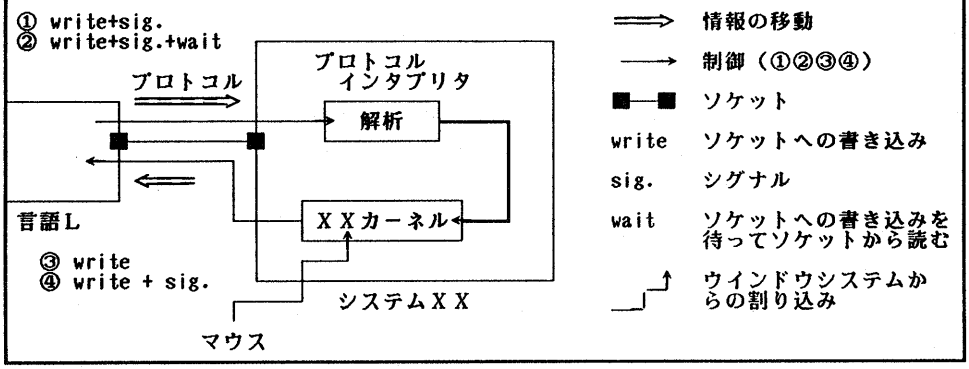
## ２．２情報制御機構

ユーザインタフェース開発言語（例えば言語Ｌ）とシステムＸＸとの間の関係について説明する（図１参照）。ウインドウを例にとってみると、ウインドウを生成するには言語ＬからシステムＸ Ｘに対して生成のためのプロトコルを送る（=>）。それによって画面にウインドウが現われるのが見えるが（--->）、実際にはシステムＸＸとそれを支えるウインドウシステムとの間にウインドウ生成のための処理がなされる（↑↓）。ウインドウにはウインドウシステムが管理するＩＤが付与されているので、このＩＤはシステムＸＸを通して言語Ｌ側に返される（<=）。また、言語Ｌの関数を起動するための文字列がメニュー項目として事前に登録してあれば、そのメニューが選ばれた時には、システムＸＸは何らかの方法でその文字列を言語Ｌ側に渡してやらなければならない。そして言語Ｌでこの文字列は評価、実行のために使われ



る。以上述べたことを正確に表現するために、言語ＬとシステムＸ Ｘ間の情報のやりとりを両プロセス間の通信方式で説明する（図２）。

図２　言語Ｌとシステム Ｘ Ｘとの通信




システムＸＸは言語Ｃで実現されており、それ自体が１プロセスである。システムＸＸを利用する言語は別プロセスとしてＸＸとの間に通信経路（ソケットを利用）を言語で構築するアプリケーションごとに設ける。言語Ｌからシステム ＸＸを利用する場合には①プロトコルをＸＸへ送り、②場合によってはその結果を受け取る。①の場合はソケットにプロトコルを書き込み(write)、書き込んだことをＸＸに知らせる(sig)。②の場合にはさらに結果がソケットに書き込まれるのを待ち、そこから読みとる(wait)。　逆に、システムＸＸから言語Ｌに対する情報としては、　③言語Ｌから送ったプロトコルの返事を待つものと、④マウス等からの割り込みによって言語Ｌに送られる情報とに分けられる。
ここで注意をしなければならないことが一点ある。いま言語ＬがＬＩＳＰだとすると④の時に送られて来たデータは単なる数字の場合もあるが、関数名の時もある。　関数の時は評価・実行しなければならないが、システムＸＸの中ではこれはすべて単なるデータとして保持されており、言語Ｌのセマンティックスは知らない。但し、ＸＸで保持されている関数名等はプロトコルを用いて言語Ｌからシステム ＸＸにデータとして渡されたものである。従って、複数の異なる言語からシステムＸＸを利用しようとする際、システムＸＸに各言語ごとに対応するセマンティックスをあらかじめ与えておく必要はまったくない。

## ２．３　ＬＩＳＰ環境下でのシステムＸＸの利用

本節では、ＬＩＳＰ環境下でシステムＸＸを利用する場合の手順を示す。前節で説明した機構は内部的な話であり、システムＸＸをＬＩＳＰから利用する場合に考慮しなければならない規則は非常に少ない。以下に機能別の関数を全て列挙する。

システムＸＸとの間の通信経路の確立／解除：　XX-connect, XX-receive-signal, XX-disconnect

プロトコル作成：　XX-set-protocol

プロトコルの送出：　XX-send-recv、XX-send

ＸＸ側から送られてくるデータを解析／分類／評価するもの：

XX-decode, XX-string-eval

これらのうち、XX-set-protocol とXX-string-evalはＬＩＳＰで数行のプログラムであり、他の関数は全てＸＸが提供するＣ関数をただリンクして用いる。しかも、各関数が扱う引数と値は数値か文字列のみ

である。従って、ＬＩＳＰ以外の言語例えばＦｏｒｔｒａｎ等でもサブルーチン、あるいは関数とＣ関数とをリンクする機構があればＬＩＳＰの場合同様、容易にシステムＸＸを利用する事が出来る。
ＬＩＳＰがシステムＸＸを利用する場合の手順は以下の通りである。

（i）　システムＸＸを起動する（ＵＮＩＸの実行形式）。　－Ｓｈｅｌｌレベル
（ii）　システムＸＸとの通信路を確立する。　　|
（iii）プロトコルを作成してＸＸへ送る。　　|－ＬＩＳＰレベル
（iv）（iii）を必要なだけ繰り返す。　　|

現在、用意してあるプロトコルには、ウインドウ、メニュー、マウス、ターミナル、グラフィック操作あわせて７５種類ある。

## ３．ユーザインタフェース構築例　－　オブジェクト階層エディタを例として　－

本章では、ＬＩＳＰからシステムＸＸを利用し構築したユーザインタフェースの実例を紹介する。図３に示したハードコピーには４つのウインドウが見える。このうち、ＬＩＳＰとシステムＸＸを起動するためのシェルウインドウと、システム自身を実現するウインドウ以外の２つのウインドウ、図形を表示している”ＴＥＳＴ＿ＷＩＮＤＯＷ”とオブジェクト階層エディタ”ＯＢＪＥＣＴ＿ＥＤＩＴＯＲ”がすべてＬＩＳＰ関数によってつくられており、かつ操作可能である（図４参照）。

次に、オブジェクト階層エディタ作成のためのＬＩＳＰプログラムの一部をP.７に掲載する。これ以外には、Ｓ式から木構造を生成、表示、探索する数十行の関数が３つと、各ポップアップメニューから起動されるオブジェクト操作関数６つと、数行の補助関数がある。このうち、木構造を表示する関数でテキストと線を書くためのＸＸ用の関数が２つ使用されている。

プログラム中XXのプレフィックスがついた下線の引いてある関数は、すべてシステムＸＸに対するプロトコルを生成する関数である。このレベルでは２．３節で説明した関数はまったく意識する必要がない。プログラムのうち、代表的な機能を記述するものを取り上げ説明する。

（１）オブジェクト階層エディタのウインドウを作り、オブジェクト関係を現わすＳ式のファイルを指定するためのテキストパネルメニューを用意する。

図４に示したように３つの関数によって必要な部品は揃う。但し、ウインドウとメニューは表示の可否を指定でき、画面上に表示するためにはあと２つの関数XX-DISPLAY-WINDOW 、XX-SHOW-PANEL-ITEMが必要である。この例で特記しておくことは、XX-CREATE-TEXT-PANELの最後の引数”show-object-hierarchy” についてである。メニュー部[Load-File] にファイル名が入力されると、　この項目とファイル名が一定のデータ形式でＬＩＳＰに送られる。ＬＩＳＰ側はこのデータを解釈し、項目を関数名とし、ファイル名を引数として関数を評価する。そしてオブジェクト階層図が表示される。

階層図のウインドウに付加されたポップアップメニューに対する起動関数の指定も、同様な形式になっている（XX-CREATE-MEMU）。

（２）オブジェクト階層図中のオブジェクト名をマウスの中ボタンで指示するとその名前を反転し、続いて右ボタンを押すとオブジェクト操作のポップアップメニューが現われる。

マウスの中ボタンを押すとその位置にある名前を反転するためには、XX-SET-MOUSE-EVENTによってマウスボタン押時に起動すべき関数を指定する。この場合はsearch-object である。マウスカーサの位置はXX-GET-MOUSE-POSによって得られる。その位置から階層図の内部データ形式を利用して選ばれたオブジェクト名を探しだし、反転して表示する。反転はXX-WRITE-REVERSED-TEXTで行なう。このプログラムでは反転

図３　システムＸＸとＬＩＳＰによるユーザインタフェース構築例

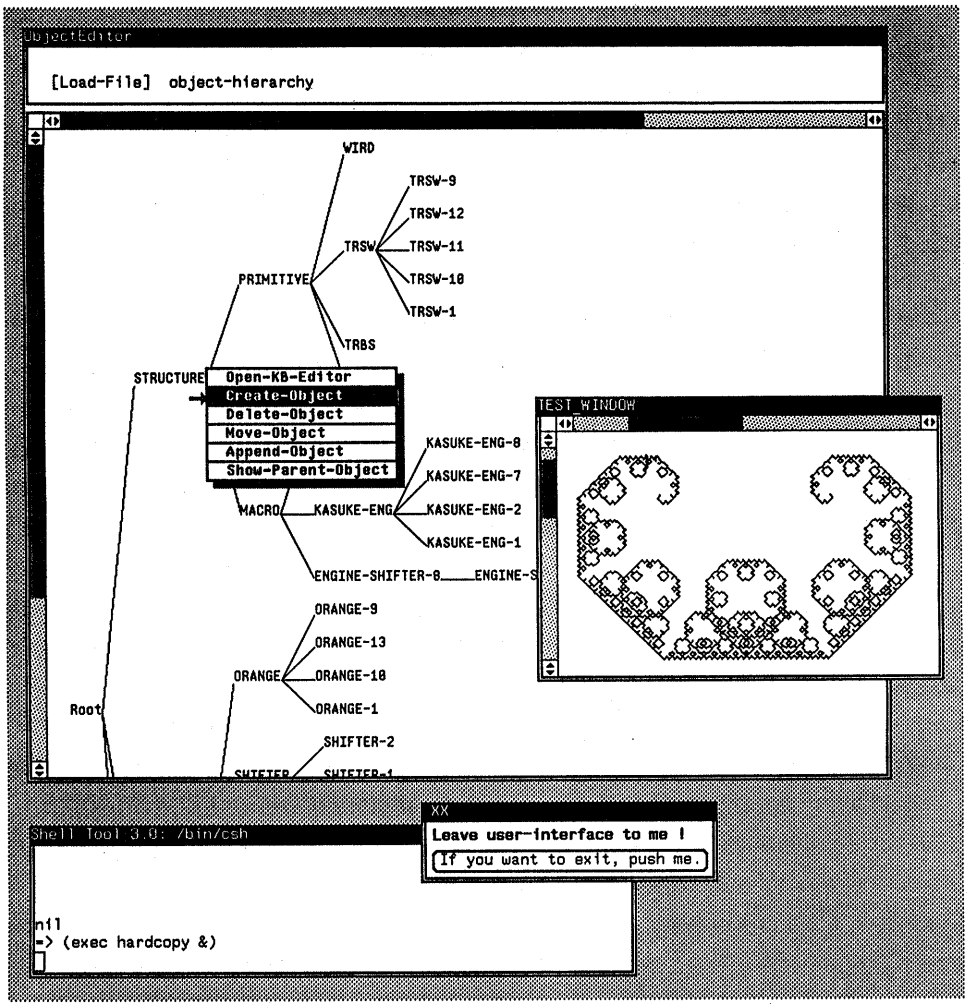
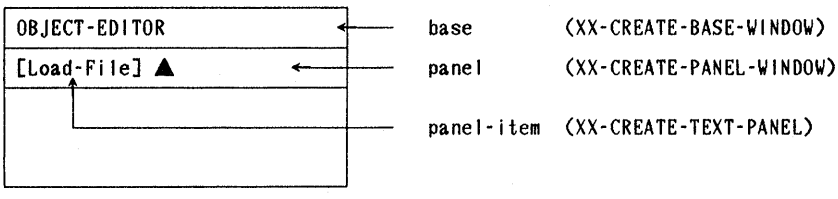



図４　ウインドウの各部とＬＩＳＰ関数の対応



```
;;;
;;; OBJECT EDITOR
;;;

(defun object-editor ()
  (set-frame object-editor-base
             (XX-CREATE-BASE-WINDOW 10  ;x-coor
                                    10  ;y-coor
                                    800 ;width
                                    800 ;height
                                        ;label
                                    "ObjectEditor"))
  (set-frame object-editor-panel
             (XX-CREATE-PANEL-WINDOW (get-frame
                                       object-editor-base)
                                     0    ;x-coor
                                     0    ;y-coor
                                     790  ;width
                                     50)) ;height
  (set-frame object-editor-panel-item
             (XX-CREATE-TEXT-PANEL (get-frame
                                     object-editor-panel)
                                   20  ;x-pos from left
                                   25  ;y-pos from left
                                   "[Load-File]" ;menu
                                   ;invoked function
                                   "show-object-hierarcy"))
  (set-frame object-editor-canvas
             (XX-CREATE-CANVAS-WINDOW (get-frame  ;FRAME ID
                                        object-editor-base)
                                      0    ;x-pos from left
                                      60   ;y-pos from left
                                      0    ;width
                                      0    ;height
                                      0    ;canvas width
                                      0))  ;canvas width
  (set-frame object-editor-menu
             (XX-CREATE-MENU
                        ;<menu label>        <invoked function>
                        ;-------------------------------------
                        "Open-KB-Editor"     "kb-editor"
                        "Create-Object"      "create-object"
                        "Delete-Object"      "delete-object"
                        "Move-Object"        "move-object"
                        "Append-Object"      "append-object"
                        "Show-Parent-Object" "show-parent-object"))
  (XX-DISPLAY-WINDOW (get-frame object-editor-base))
  (XX-SHOW-PANEL-ITEM (get-frame object-editor-panel-item))
  (XX-SET-MENU (get-frame object-editor-canvas)
               (get-frame object-editor-menu)
               2)     ;Right Button
  (XX-SET-MOUSE-EVENT 1  ;Middle Button
                      ;invoked function
                      "search-object")
  )

(defun show-object-hierarcy (file-name)
  (tree (get-frame object-editor-canvas)
        (read-sexp-from-file file-name)
        20
        20))

(defun search-object ()
  (let ((current-pos
         (decode (XX-GET-MOUSE-POS
                  (get-frame object-editor-canvas)))))
       (setq *current-node* (search-tree
                             *current-tree*
                             (x-coor current-pos)
                             (y-coor current-pos)))
       (XX-WRITE-REEVERSED-TEXT
        (get-frame object-editor-canvas)
        (name-of current-node)
        (pos-x-of current-node)
        (pos-y-of current-node)
        1)))
```

したままである。続いて右ボタンを押すとポップアップメニューが現われるのは、このウインドウにメニューをXX-SET-MENU によってあらかじめ割りつけておいてからである。

以上の例からも分かるように、ユーザインタフェースとしてはよく用いる機能に対してＬＩＳＰで簡潔に記述できた。これはシステムＸＸが言語依存部分を極めて少なくすると同時に、提供するプロトコルの機能の上での抽象度が高いためである。

## ４．まとめ

今回はユーザインタフェース構築のための支援環境として、プロトコルレベルのウインドウ操作プリミティブとそれらを管理するシステムＸＸの報告をおこなった。ＸＸはウインドウシステムそのものではなく、既存のウインドウシステムを簡便に使えるためのツールの一つである。

システムＸＸの特長は、［１］抽象的な命令セットであるプロトコルを採用、［２］ＸＸを利用する際の言語依存部分が少ない、［３］ＸＸをサーバとし、これを利用する側をクライアントとする通信方式を採用、の以上３点が組み合わさることによって得られる。

・ウインドウシステムの多くの補助的な関数名や複雑なデータ構造を意識しなくてもよい。

・複数の異なる言語が同一のインタフェースを共有することが可能である。但し、現在は複数のシステムからの同一のアクセスに対する排他制御機構はまだ取り入れていない。

・インタフェース構築の際の記述量が少なくしかも記述性に優れる。従って、言語レベルではあるがインタフェースのカスタマイズ、変更等に対しては容易に対処できる。

・言語ごとによる、システム自身の修正はまったく不要である（言語依存部分はシステムＸＸ自身には含まれず、関数として各言語ごとに提供される）。

・ＸＸに対する依頼に対して、結果を待たずに実行を続けることできる。

・クライアント側の負荷（コーデング量、オブジェクトサイズ）は軽くてすむ。

・特に、ＬＩＳＰのような言語からＸＸを利用できると、ウインドウレイアウトや、図形作成等がインタラクティブに行なえ、毎回修正やコンパイルが必要となるコンパイラ言語と比較して、　生産性が向上する。また、階層図表示などでは、木探索アルゴリズムはＬＩＳＰで実現し、表示のみＸＸを利用すればよく開発が非常に容易である。特に、３章で紹介したオブジェクト階層エディタは木操作を含めてわずか１週間程度で作成された。

ＸＸはユーザインタフェースを構築するために機能を言語レベルで提供したが、今後はウインドウレイアウト、メニュー設定等自身を設計できるウインドウ操作環境が必要となってくる。その段階に至って初めて、インタフェースを使うユーザ自身がインタフェースを自由に設計、カスタマイズすることが可能となってくるであろう。

ＸＸは、現在、我々自身が開発している応用システム上のユーザインタフェースを構築・支援するために使用されている。