4W-7

対象指向型Lisp：EUSlispの他言語インタフェースとウィンドウシステムへの応用

稲葉雅幸（東京大学工学部）
松井俊浩（電子技術総合研究所）

1. はじめに

ロボットシステムには、幾何モデラ、画像処理、軌道生成、認識処理など様々な分野のソフトウェア資産を簡単に取り込めるだけの拡張性が必要である。そこで、ロボット向きに開発している対象指向型lisp(EUSlisp)[1]へ他言語インタフェース機能を付加して、既存のライブラリや他言語で記述されたプログラムパッケージを統合することを行った。さらに統合した機能をEUSlispのクラスとして整理する試みを行った。応用例としては、ロボットシステムの基本ツールとして利用できるもので、クラス化の効果が高く、Lispとライブラリとの相互呼び出しの機能が必要であるウィンドウ・システムとした。

2. 他言語インタフェース

他言語インタフェースは、CやFORTRANなどで記述されたプログラムやデータとの相互利用を実現するためのものであり、以下のような特徴を持つ。

（1）インクリメンタルにオブジェクト・ファイルをロードできる。たとえば、

```
(setq *fm* (load-foreign  "test"
                :ld-option "-lpixrect -lm -lc"))
```

のようにする。

（2）ロード時に自動的にリンクされるため、呼出しアドレスを意識する必要がない。上の*fm*はシンボル表を持つており、その中に登録されたラベルで参照が可能となる。

（3）他言語関数をLisp関数として宣言すれば、それをLisp関数と同様に扱うことができる。

```
(defforeign sprintf *fm* "_sprintf"
        (:string :string) :integer)
```

のように他言語モジュール、外部関数名、引数の型、結果の型を指定して宣言を行う。

（4）必ずしも機械語になっていないlambda形式の関数も他言語からコールできる。たとえば、

```
(defun-c-callable plus (a b) (+ a b))
```

とすれば、(plus 1 3) は4を帰し、

```
(pod-address 'plus)
```

により得られる関数エントリを他言語プログラムへ渡せば、他言語から整数の足算関数として利用できる。

（5）Cの構造体(struct)をlispオブジェクトとして定義して他言語へ渡すことができる。Lispはダイナミックにデータを生成でき、メモリ管理を自動的に行うためC、Fortran、Pascalなどでデータを管理する部分が不必要である。

3. ウィンドウシステムのインタフェース

軌道計画、形状モデラなどでは幾何データの取扱いが多いため、それらの入出力が簡便であることは重要である。画像処理プログラムは各種のパラメタを要求する処理が多く対話的に簡便にパラメタを調節できるものが必要である。Lispの対話性とEmacs等の強力なエディタによってキー入力による方法で簡便性が得られているが、二次元、三次元の数値入力、閾値処理パラメタの連続的変更により処理結果をダイナミックに観察する場合にはウィンドウシステムが強みを発揮する。

ここではEUSLispの他言語インタフェースにより、Sun ViewとSunCoreを利用できるようにした。Sunワークステーション上のLispにそれらのライブラリを組み込んだ例は2~3[2,3]あるが、ここでは(1)クラス化されたモジュールをベースとした他言語インタフェースを設けて実現している、(2)メモリ管理がスタティックであるためデータの共有に特殊な処理を必要としない、(3)対象指向型Lispであるためインタフェースしたライブラリの機能のクラス化が容易であるといった点が特徴である。

SunViewはマウスの状態によりウィンドウに登録されているイベントハンドラを適宜コールする。イベントハンドラをLispで記述できるようにdefun-c-callableマクロを利用できる（図1 Example1参照)。notify_do_dispatchによりlispのreaderが入力待ちの間、イベントの処理がなされる。これによりキーボードからLispを利用することと同時にウィンドウシステムからの入力も可能となり、ウィンドウの使いみちは格段に広がる。

4. ウィンドウのクラス化

外国語インタフェースの機能により、Cで記述できる形式のアプリケーションはEUSlispで実現され、変数宣言もいらないためCより簡便に記述される。しかし、(1)ウィンドウの属性であるイベントハンドラの定義がウィンドウの定義場所から離れて判りにくい、(2)組込みイベントのシーケンスで決まるような新しいイベントに反応するウィンドウを定義するにはイベントバッファなどの記憶域が必要でありそれを個々のハンドラに個別に用意すると繁雑になる、などの問題がありウィンドウをクラス化した。

クラス構成はSunViewの仕様にあわせ、ウィンドウの

Object Oriented Lisp : EUSlisp
- Foreign language interface and its application to window system -

Masayuki INABA[1], Toshihiro MATSUI[2]
1:University of Tokyo, 2:Electrotechnical Laboratory

属性値はメソッドでアクセスしてコピーを持たない。問題はイベント処理であるが、他言語からコールできる関数のための新しいシンボルをハンドラ登録時に自動生成するようにした。そして、イベントを発生した外部ライブラリ内のウィンドウデータとlispのウィンドウデータとの対応付けは、グローバル変数に結合したウィンドウオブジェクトへイベントを送ることで実現している。

図1のExample 2にイベントハンドラを登録するプログラムの例を示す。新しいイベントとして、drag-started,drag-finished,dragging,moving,single-clicked,などを用意し、例のように登録する。:event-handleメソッド内でシステム組込みのイベントが解釈され、新しいイベントであれば登録された処理がコールされる。

図2には実際に利用しているウィンドウ例として画像処理ハードウェアの操作パネルの例を示す。ある決まった処理を起動するためのbutton，モード切り換えのためのchoice，連続的に数値を入力するためのslider，ファイル名などの入力のためのtextといったパネル項目を多数用い、抽出された線画や形状モデルを表示するために2つのcanvasを用いている。図2のウィンドウ定義をCと同じようにLisp関数で記述した場合には約1000行ぐらいであったがクラス化により、約500行程度になった。

```
;;; Example 1.
(defun-c-callable quit-proc (i e)
  (window_destroy w))
(defun simple nil
  (notify_do_dispatch)
  (setq bold
   (pf_open
  "/usr/lib/fonts/fixedwidthfonts/screen.b.12"))
  (setq hpixrect
   (icon_load_mpr
    "/usr/include/images/hello_world.icon"
    (setq errbuf (make-string 32))))
  (setq hicon (icon_create icon_image hpixrect))
  (setq w
   (window_create_frame
    null frame_label "simple" frame_icon hicon))
  (setq p (window_create_panel w win_font bold))
  (panel_create_message p
   panel_label_string "Hit button to quit.")
  (panel_create_button p
   panel_label_image
    (panel_button_image p "quit" 5)
   panel_notify_proc (pod-address 'quit-proc))
  (window_fit p) (window_fit w)
  (window_set w win_show true))

;;; Example 2.
(defun test nil
  (notify_do_dispatch)
  (setq F (instance sunview-frame :init nil))
  (setq P (instance sunview-panel :init F))
  (setq B
   (instance panel-button-item
    :init P :name "Button"
    :nproc  #'(lambda (item event)
                (format t "Event=~S~%" event))))
  (send P :fit_height)
  (setq C
   (instance sunview-canvas
    :init F :x 0 :width 300 :height 300
    :click-handler
    '(lambda (button pos time)
       (if (eq button 'middle) (print 'm)))
    :drag-started-handler
    '(lambda (button pos time)
       (print (v+ pos #f(10 10))))
    :eproc
    #'(lambda (canvas event)
        (send C :handle-event event))))
  (send C :fit) (send F :fit)
  (send F :show)
  )
```

図1　プログラム記述例

5．　おわりに

他言語インタフェースを用いて既存のソフトウェアを有効利用することを考え、EUSlispへのウィンドウシステムの組込みを行った。さらにEUSlispの対象指向機能を用いてウィンドウのクラス化を行った。他言語インタフェースの開発、SunView,SunCoreのインタフェースに要した期間は約2カ月である。他言語の資産を取り込み、EUSlispの対象指向機能により簡単にウィンドウシステムの機能を吸収できた。

lisp-machineのようにウィンドウシステムをEUSLispで最初から記述することと比べると開発に要した手間は格段に少なく、lisp以外の言語で記述されたパッケージも組込み可能である点が特長である。既存のウィンドウシステムのクラス化による統合は、イベント処理の起動が外部ライブラリ内で行われるため、イベントをLisp内部のウィンドウオブジェクトへ伝達する部分が必要であるが4．で説明したようにすれば簡単に実現できる。

ウィンドウシステムを組み込んだことで、だれでも利用できて拡張が容易なアプリケーション・プログラムの蓄積が容易になった。今後はEUSlispのモデリング機能[4]と併せてロボットシミュレータへの応用を行う。

謝辞　共同研究の機会を与えて頂いた白井良明元電総研制御部長、井上博允東大工学部教授に感謝する。

参考文献

[1]松井、塚本：「EUSlisp:対象指向による型拡張性を有するlispの実現」、情報処理学会第35回全国大会、2R-5、1987.

[2]山本他：「AS3000用Kyoto Common Lisp」、情報処理学会知識工学と人工知能研究会資料、49-5、1986.

[3]Franz Inc. : Extended Common Lisp User Manual、1986.

[4]松井、稲葉：「対象指向型Lisp：EUSLispを用いたロボット用幾何モデリングシステム」、情報処理学会第37回全国大会、1988.

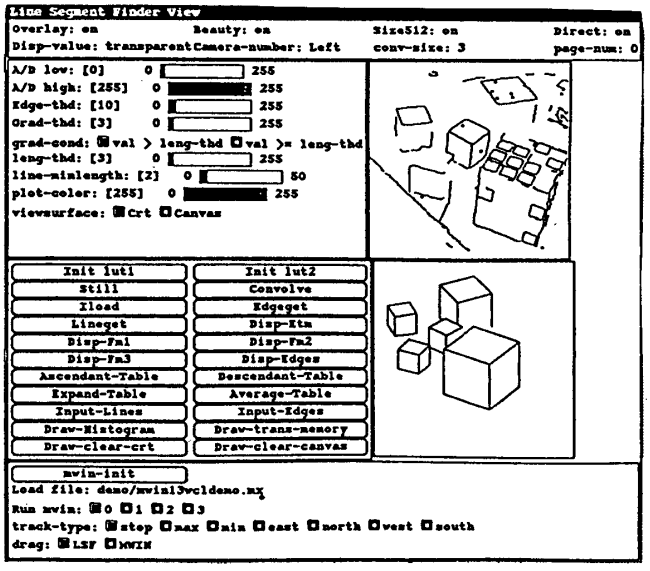

図2　ウィンドウ・パネルの例