A Complete User Interface System

Primary Examples:
MacOS, Visual Basic, NeXTStep, Java, Common Lisp Interface Manager

- **windowing abstraction**
  containers, views

- **display components**
  button, checkbox, choice box, label, list, table,
  scrollbar, textarea, textfield, window, menu, dialog box

- **display tools**
  fonts and points
  color
  graphics system (drawing, clipping, 3D)
  image handling
  layout management

- **temporal data tools**
  time and synchronization model
  sound manager
  video manager
  animation manager

- **interactivity tools**
  event handling and management (mouse, keyboard, arbitraty input devices)
  streams and buffers
  scripting language

- **programming interface tools**
  object-oriented class, instance, and message system (initialize-, make-)
  load, compile, link, and evaluate
  language-specific text editor
  interface construction toolkit
  debugging and exception handling
  namespaces and packages
  foreign function interface

- **operating system tools**
  threads and multitasking
  concurrency, switching, scheduling, and synchronization
  memory management
  file system interface
  network interface and security
  low level: internal data structures, pointers, memory blocks, traps

Extended Examples (Java, CLIM)
using widget interactions as a simple example

Generic object operators/functions:

constructors: make-, initialize-, set-
assessors: get-
queries: ? -
functions: act-on-
relations: constrain-

Turnkey dialog boxes

throw-cancel and catch-cancel <aborts>
message-dialog
yes-or-no-dialog
get-string-from-user-dialog
select-item-from-list-dialog

Windows

nested-views, size, position, scroller, click-handler
title, font, color, active?, layer, zoom, grow, drag

Mac Common Lisp Menu Class structure

menu-element
- menubar (class, variable, function)
  - set-menubar
  - find-menu
  - <color-functions>
  - *default-menubar*
- menu
  - initialize-, set-
    - menu-title, menu-items, menu-colors
    - update-function
    - help-spec (balloon-help system)
  - install, deinstall, installed?
  - enable, disable, enabled?
  - font-style, <color-functions>
  - add-menu-item, remove-menu-item, get-menu-item, find-menu-item
  - menu-item
    - initialize-, set-, get-, query?-
      - owner, title
      - command-key, checked
      - action, action-function (call vs get)
      - disabled?
      - colors, font-style
      - update-function, help-spec
  - window-menu-item
    - close, save, save-as, save-copy-as, revert, hardcopy
    - cut, copy, paste, clear, select-all, undo, undo-more
    - load/evaluate-selection, load/evaluate-whole-buffer

Mac Common Lisp Dialog-items

initialize-, set-, get-, make-
view-size, view-container, view-position, view-nickname, view-font
dialog-item-text, dialog-item-handle, dialog-item-enabled?
part-color-list, dialog-item-action, help-spec, window-pointer
install, activate, activate-event-handler, default

button-dialog-item
press-button, default-button-dialog-item (make-, get-, set-, ?-)
static-text-dialog-item
editable-text-dialog-item
<key-stroke-handlers>
check-box-dialog-item (check-box-check, -uncheck, -checked?)
radio-button-dialog-item (radio-button-cluster, -push, -unpush, -pushed?)
table-dialog-item
<table-constructors>, <cell-contents-handlers>, sequence-dialog-item
pop-up-menu (<handlers>)
scroll-bar (<handlers>)

Interface Toolkit

The toolkit provides a drag-and-drop interface for constructing display interfaces. After selecting and positioning the interface, the toolkit writes the appropriate source code for that interface. Toolkit components:

Menubar Editor
Add Menu
Add Menu Item
Command key, Disabled, Check Mark
Menu Item Action (provide function), Menu Item Colors
Menu Colors
Print Menu Source
Rotate Menubars
Add New Menubar
Delete Menubar
Menubar Colors
Print Menubar Source
Use Dialogs (toggle with Design Dialog)
Design Dialogs
Document
Document with Grow
Document with Zoom
Tool (with title bar and close button)
Single Edge Box
Double Edge Box
Shadow Edge Box
Design Dialog Methods
Include Close Box
Color Window
Add Dialog Item

Static Text
Editable Text Field (Allow Returns, Allow Tabs, Draw Outline)
Button (Default Button)
Radio Button (Radio Button Pushed, Set Item Cluster)
Checkbox (Checkbox Checked)
Table (Set Cell Size, Horizontal Scroll Bar, Vertical Scroll Bar
Set Table Sequence, Set Wrap Length, Orientation)

Add Dialog Item Methods
Dialog Item Text
Enabled/Disabled
Set Item Action
Set Item Font
Set item Name
Set Color
Print Item Source

New Dialog
Add Horizontal Guide (for alignment during editing)
Add Vertical Guide
Edit Dialog
Print Dialog Source

Java Code for constructing some widgets

Named Button:

```
public void okButton() {
        Button b = new button("OK");
        add(b);  }
```

Unnamed button:

```
add(new Button("OK"))
```

Label:

```
add(new Label("Look at me"))
```

Checkbox:

```
add(new Checkbox("Check here if hungry"))
```

Checkbox Methods:
getLabel(), setLabel(String), getState(), setState(boolean)

Choice Menu:

```
{Choice myClassesMenu = new Choice;
  myClassesMenu.addItem("SE101");
  myClassesMenu.addItem("SE561");
  myClassesMenu.addItem("Special Project");
  add(myClassesMenu); }
```

Choice Menu Methods:
getItem(int), countItems(), getSelectedIndex(),
getSelectedItem, select(int), select(String)

PRODUCTION LISP CODE for a WINDOWING SYSTEM

Unedited, little documentation, good style.
This code is what you would have to write if you were developing an application windowing system without a toolkit or a class library.
Redundant code templates are omitted.

First the **class structure** for the windowing environment,
next the **menu system** with its corresponding **action functions**,
then the **control panel** with its corresponding action functions,
finally the **event handler** for text entry into the control window.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; PARENT-WINDOW

(defclass parent-window (window)
  ((children :accessor children :initarg :children :initform nil)
   (common-data :accessor common-data :initarg :common-data :initform nil)))

(defmethod initialize-instance ((self parent-window) &rest rest)
  (apply #'call-next-method self rest)
  (map-children self #'set-child-parent self))

(defmethod find-parent-child ((self parent-window) type)
  (car (member type (children self) :key #'type)))

(defmethod add-parent-children ((self parent-window) &rest children)
  (setf (children self) (append children (children self))))

(defmethod remove-parent-children ((self parent-window) &rest children)
  (setf (children self) (set-difference (children self) children)))

(defmethod parent-children ((self parent-window) &rest children)
  (apply #'add-parent-children self children)
  (mapcar #'(lambda (child) (set-child-parent child self)) children))

(defmethod map-children ((self parent-window) func &rest args)
  (mapcar #'(lambda (child) (apply func child args)) (children self)))

(defmethod open-child ((self parent-window) type &rest rest)
  (cond
   ((eq type 'entry) self)
   ((find-parent-child self type))
   ((eq type 'database)
    (apply #'make-instance 'database-window :parent self rest))
   (T
    (apply #'make-instance 'display-window
           :type type :parent self rest))))
```

```
(defmethod window-close ((self parent-window))
  (call-next-method self)
  (map-children self #'window-close))

(defmethod set-window-title ((self parent-window) new-title)
  (map-children self #'set-window-title new-title)
  (call-next-method self new-title))

;;;six window subclasses and methods omitted here

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; DISPLAY-WINDOW

(defclass display-window (child-window)
  ((display-view :accessor display-view :initform nil)
   (title :accessor title :initform "Display"))
  (:default-initargs
    :window-type :document-with-zoom
    :view-font '("Monaco" 9)
    :view-size #@(300 150) ))

(defmethod initialize-instance
      ((self display-window) &rest rest &key (type 'display-view))
  (declare (dynamic-extent rest))
  (apply #'call-next-method self :type type rest)
  (let ((view (make-instance type
                :view-container self
                :view-size (subtract-points (view-size self) #@(15 15))
                :view-position #@(0 0)
                :draw-scroller-outline nil)))
    (setf (display-view self) view)
    (setf (title self) (title view))
    (when (parent self)
      (set-window-title self (window-title (parent self))))
    (mapcar #'(lambda (x) (setf (scroll-bar-scroll-size x) 12))
            (view-scroll-bars view))
    (set-common-data view (common-data self))))

(defmethod set-view-size ((self display-window) h &optional v)
  (declare (ignore h v))
  (without-interrupts
   (call-next-method)
   (let* ((new-size (subtract-points (view-size self) #@(15 15))))
       (set-view-size (display-view self) new-size))))

(defmethod window-zoom-event-handler ((self display-window) message)
  (declare (ignore message))
  (without-interrupts
   (call-next-method)
   (let* ((new-size (subtract-points (view-size self) #@(15 15))))
       (set-view-size (display-view self) new-size))))

(defmethod clear ((self display-window))
  (call-next-method self))
```

```
(defmethod save-to-eval ((self display-window))
  `(make-instance 'display-window
     :type ',(type self)
     :window-title ,(window-title self)
     :view-position ,(view-position self)
     :view-size ,(view-size self)  ))

(defmethod window-close ((self display-window))
  (when (parent self)
    (remove-parent-children (parent self) self))
 (call-next-method self))

(defun make-trace-output-window (parent)
  (make-instance 'display-window
    :type 'trace
    :parent parent
    :close-box-p    nil
    :window-title   "Trace Output Window"  ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; LOSP-MENU

(defvar *losp-menu* nil)
(defvar *db-edit-menu* nil)

(defun initialize-losp-menu ()
  (menu-install (setq *losp-menu* (make-losp-menu)))
  (menu-install (setq *db-edit-menu* (make-db-edit-menu))))

(defun make-losp-menu ()
  (MAKE-INSTANCE 'MENU
    :MENU-TITLE  "Losp"
    :MENU-ITEMS
    (LIST (MAKE-INSTANCE 'MENU-ITEM
            :MENU-ITEM-TITLE    "About..."
            :MENU-ITEM-ACTION   #'make-losp-ABOUT-WINDOW)
          ;(MAKE-INSTANCE 'MENU-ITEM
          ;  :MENU-ITEM-TITLE    "Load"
          ;  :MENU-ITEM-ACTION   #'menu-load-losp)
          (MAKE-INSTANCE 'MENU-ITEM
            :MENU-ITEM-TITLE    "Entry Window"
            :MENU-ITEM-ACTION   #'menu-make-entry-window)
          (MAKE-INSTANCE 'MENU-ITEM
            :MENU-ITEM-TITLE    "Control Panel"
            :MENU-ITEM-ACTION   #'make-losp-CONTROL-PANEL
            :COMMAND-KEY        #\=
            :MENU-ITEM-CHECKED  nil)
          (MAKE-INSTANCE 'MENU-ITEM
            :MENU-ITEM-TITLE    "Test Minimizer"
            :MENU-ITEM-ACTION   #'run-min-test)
          (MAKE-INSTANCE 'MENU-ITEM
            :MENU-ITEM-TITLE    "Quit Losp"
            :MENU-ITEM-ACTION   #'close-LOSP
            :command-key        #\Q)))        )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; MENU ACTION FUNCTIONS
;;;
;;;   Menu items:             Activation function:
;;;     About...                 make-losp-about-window
;;;     Load                     menu-load-losp    [in initialize file]
;;;     Entry Window             make-entry-window
;;;     Control Panel            make-losp-control-panel
;;;     Test Minimizer           run-min-test
;;;     Quit Losp                close-losp

(defun menu-make-entry-window ()
  (setq *current-entry-window* (make-entry-window))
  ;(make-losp-control-panel))

(defun close-losp ()
  (if *current-entry-window* (window-close *current-entry-window*))
  (menu-deinstall *db-edit-menu*)
  (menu-deinstall *losp-menu*))

;;;several menu functions omitted here

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; ABOUT-LOSP

(defun make-losp-about-window ()
  (modal-dialog
   (MAKE-INSTANCE 'COLOR-DIALOG
     :WINDOW-TYPE      :DOUBLE-EDGE-BOX
     :WINDOW-TITLE     "about-losp"
     :VIEW-POSITION    #@(426 60)
     :VIEW-SIZE        #@(370 185)
     :CLOSE-BOX-P      NIL
     :VIEW-FONT        '("Chicago" 12 :SRCOR :PLAIN)
     :VIEW-SUBVIEWS
     (LIST (MAKE-DIALOG-ITEM
            'STATIC-TEXT-DIALOG-ITEM   #@(58 8)  #@(260 16)
            "Losp Boolean Minimization Engine 1.0"   'NIL)
           (MAKE-DIALOG-ITEM
            'STATIC-TEXT-DIALOG-ITEM   #@(146 31)  #@(73 16)
            "May 1995"   'NIL)
           (MAKE-DIALOG-ITEM
            'STATIC-TEXT-DIALOG-ITEM   #@(8 58)  #@(345 32)
            "Copyright  (C)   1995,    OZ...International, Ltd.  and Interval
Research Corporation, All Rights Reserved."   'NIL)
           (MAKE-DIALOG-ITEM
            'STATIC-TEXT-DIALOG-ITEM   #@(20 102)  #@(345 16)
            "Authored by William Bricken and Jeffrey James."   'NIL)
           (MAKE-DIALOG-ITEM
            'BUTTON-DIALOG-ITEM   #@(130 140)  #@(114 23)
            "OK"
            #'(lambda (item) (declare (ignore item))
               (return-from-modal-dialog t))
            :DEFAULT-BUTTON    T)))   ))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; CONTROL-PANEL

(defun make-losp-control-panel ()
  (setq *problem-number-comtab* (make-comtab))
  (comtab-set-key *problem-number-comtab*
                  '(#\Newline) 'accept-problem-number-text-entry)
  (setq *isolate-variable-comtab* (make-comtab))
  (comtab-set-key *isolate-variable-comtab*
                  '(#\Newline) 'accept-isolate-variable-text-entry)
  (setq *losp-control-panel*
        (MAKE-INSTANCE 'control-panel-window
          :WINDOW-TYPE         :TOOL
          :WINDOW-TITLE        (format nil "Losp Control Panel")
          :VIEW-POSITION       '(:TOP 208)
          :VIEW-SIZE           #@(230 254)
          :VIEW-FONT           '("Chicago" 12 :SRCOR :PLAIN)
          :parent              *current-entry-window*
          :VIEW-SUBVIEWS       (losp-control-panel-subviews)))
  (set-radio-buttons-when-opened)
  (set-logic-check-box-when-opened)
  (set-circuit-check-box-when-opened)
  (set-trace-check-box-when-opened)
  (set-database-check-box-when-opened))

(defun losp-control-panel-subviews ()
  (LIST (MAKE-DIALOG-ITEM
         'STATIC-TEXT-DIALOG-ITEM   #@(10 5)  #@(56 16)
         "Analysis"
         'NIL)
        (MAKE-DIALOG-ITEM
         'BUTTON-DIALOG-ITEM   #@(70 3)  #@(60 18)
         "Apply"
         #'(LAMBDA (ITEM) (apply-button-action item))
         :VIEW-FONT                  '("Courier" 12 :SRCOR :PLAIN)
         :view-nick-name             'apply-button
         :DEFAULT-BUTTON              NIL)
        (MAKE-DIALOG-ITEM
         'RADIO-BUTTON-DIALOG-ITEM   #@(10 28)  #@(110 16)
         "Transcribe"
         #'(LAMBDA (ITEM) (transcribe-radio-button-action item))
         :VIEW-FONT                   '("Geneva" 12 :SRCOR :PLAIN)
         :view-nick-name              'transcribe-radio-button
         :RADIO-BUTTON-PUSHED-P       nil)
        (MAKE-DIALOG-ITEM
         'EDITABLE-TEXT-DIALOG-ITEM   #@(160 222)  #@(52 15)
         ""
         #'(LAMBDA (ITEM) (case-variable-text-action item))
         :VIEW-FONT              '("Geneva" 12 :SRCOR :PLAIN)
         :view-nick-name         'isolate-variable-text-box
         :comtab                 *isolate-variable-comtab*
         :ALLOW-RETURNS          T)  ))

;;;18 other dialog-item specifications omitted here
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; ACTIONS
;;;
;; see process file for usage of the globals
;;   *valid-analysis-levels*  *current-analysis-level*
;;   *active-displays*  *most-recent-analysis-result*
;;   *current-entry-window*

(defun set-radio-buttons-when-opened ()
  (cond
   ((eq *current-analysis-level* '*TRANSCRIBE*)
    (radio-button-push
     (view-named 'transcribe-radio-button *losp-control-panel*)))
   ((eq *current-analysis-level* '*CLEAN*)
    (radio-button-push
     (view-named 'clean-radio-button *losp-control-panel*)))
   ((eq *current-analysis-level* '*SORT*)
    (radio-button-push
     (view-named 'sort-radio-button *losp-control-panel*)))
   ((eq *current-analysis-level* '*EXTRACT-LITERALS*)
    (radio-button-push
     (view-named 'extract-literals-radio-button *losp-control-panel*)))
   ((eq *current-analysis-level* '*CANCEL-BOUNDS*)
    (radio-button-push
     (view-named 'cancel-bounds-radio-button *losp-control-panel*)))
   ((eq *current-analysis-level* '*INSERT-BOUNDS*)
    (radio-button-push
     (view-named 'insert-bounds-radio-button *losp-control-panel*)))
   ((eq *current-analysis-level* '*MINIMIZE*)
    (radio-button-push
     (view-named 'minimize-radio-button *losp-control-panel*)))
   (T nil)))

(defun transcribe-radio-button-action (self)
  (setq *current-analysis-level* '*TRANSCRIBE*)
  self)

(defun clean-radio-button-action (self)
  (setq *current-analysis-level* '*CLEAN*)
  self)

(defun sort-radio-button-action (self)
  (setq *current-analysis-level* '*SORT*)
  self)

(defun database-display-box-action (self)
  (let ((win (when *current-entry-window*
               (find-parent-child *current-entry-window* 'database))))
    (if win
      (window-close win)
      (make-database-window *current-entry-window*)))
  self)

;;;20 other action specifications omitted here
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; ENTER LOSP-ENTRY-WINDOW
;;;
;; Controls the behavior of the 'return' key in the entry buffer.
;; If the cursor is not on the last line of the buffer, 'return' copies
;; the current line (without the prompt) to the end and moves the cursor
;; there too.  No error handling is provided here.

(defun accept-entry (entry-window &optional force)
  (let* ((bmark (fred-buffer entry-window))
         (end (buffer-line-end bmark))
         (eob (buffer-size bmark))
         (entry (string-left-trim *entry-prompt*
                  (buffer-substring bmark (buffer-line-start bmark) end)))
         (symbol-entry (string2symbol-boxed entry)))
    (cond                              ;; accept input
     ((or force (= end eob))
      (set-mark bmark eob)
      (ed-insert-char entry-window #\Newline)
      (if (null symbol-entry)
        (buffer-insert-at-end bmark "return")
        (let ((logic-type
               (intersection (flat symbol-entry) *valid-logic-functions*))
              (assertion-type (member *assertion-token* symbol-entry)))
          (cond
           (logic-type
            (buffer-insert-at-end bmark *multiple-form-message*))
           (assertion-type
            (buffer-insert-at-end bmark "Assert:  ")
            (buffer-insert-at-end bmark
              (assert-entry entry-window (remove-assert-mark symbol-entry))))
           (T (let ((result (process-entry entry-window symbol-entry)))
                (buffer-insert-at-end
                 bmark (prepare-text-out result)))))))
      (ed-insert-char entry-window #\Newline)
      (buffer-insert-at-end bmark *entry-prompt*))
     ;; otherwise copy entry to the end of the buffer
     (T (buffer-insert-at-end bmark entry)))))

(defun force-accept (entry-window) (accept-entry entry-window T))

(defun buffer-insert-at-end (bmark string)
  (buffer-insert bmark string (buffer-size bmark))
  (set-mark bmark (buffer-size bmark)))

(defun prepare-text-out (form)
  (cond
   ((null form)  "")
   ((marked form) "()")
   (T (let ((string-form (symbol2string form)))
      (remove #\) (remove #\( string-form :count 1)
              :count 1 :from-end t)))))

;;;many other handlers and functions omitted here
```