Detailed Event Handling

Reading #4: "Chapter 4.3-4.6 Basics of Event Handling" by Dan Olsen, Developing User Interfaces, 1998, pp. 89-104.

Part I

How are events managed by the UIMS?

• Events are typed
  – What kind of event is it?

• Events are filtered and processed
  – Who has to deal with it?
    • Either windowing system or to application or none

UIMS Event Types

• Input Events
  – Mouse Buttons
  – Modifier Keys (Shift, Control, Meta, Option, etc.)
  – Double-Clicking, triple-clicking
  – Function Keys
  – Mouse Movement
  – Mouse-Enter & Exit
  – Keyboard

• Windowing Events
  – Create, Destroy, Open, Close, Iconify, Deiconify, Resize

• Redrawing Events

• Pseudo-Events: communication between objects
How are events managed by the UIMS?

• Events are filtered
  – Either windowing system or to application or none
• Event priority queue managed by OS
• Ordered by
  – Priorities pre-set by OS for event types
  – Timestamp
• Older Mac OS and Microsoft Windows have only one queue
• Multi-tasking OS (e.g. X Window, Mac OS X) has a queue for each process

Macintosh Event Priority Queue

1. Activate event: activate specific window
2. Mouse, keyboard, disk events
3. Auto-key event
4. Update event: repeated key
5. No event, continue to loop

How are events managed by the UIMS?

• Events are records sent by the windowing system to the application
  – name of event
  – timestamp
  – event-specific fields such as XY location for pointing device
  – widget object or window ID
UIMS Event Processing

Event Record

Event = Record
    EventCode: Integer;
    MouseX, MouseY: Integer;
    EventValue: Integer;
    Time: Integer;
    WindowID: Integer;
End;

where EventCode “1” for mouse button;
    EventValue “2” for down

How are events managed by the UIMS?

cont.

How does the windowing system associate the event with a window?
Called “event dispatching”

– Hierarchy of windows
  • bottom-first processing

– Input focus
  • Currently selected window receives all key & mouse events
Event Dispatching
Hierarchy of windows

Part II
Event management within the program

• Main Event loop
  – Procedural languages
    • Explicit main event loop
    • Procedure name, event table, callbacks
  – Object-oriented languages
    • Implicit main event loop
    • Event handlers

Explicit Event Handling in the Application Program

• Trap calls to ROM-based Toolbox code
  – Example: Macintosh Pascal would use “case” statement

• Event-table
  – Each window has a pointer to an event table for each possible event
  – Event table has addresses for procedures to handle various event types
  – Example: Applications written completely in C
The Main Event Loop

Explicit Main Event Loop

Explicit Main Event Loop cont.
Big Problem: Hooking the UIMS and application back together

- How does the UIMS send the application the information to process the correct semantics for an event?
  - Can associate application procedures directly by name
    - Kernel models
  - Can associate application procedures through callbacks
    - Client-server models, e.g. Motif

UIMS to Application Semantics

Associating Procedure Names or Addresses

UIMS to Application Semantics Example: Simple Drawing Application

Application Semantics
if line-icon,
  DrawLine(X1,Y1,X2,Y2)
if rect-icon,
  DrawRect(X1,Y1,X2,Y2)
if poly-icon,
  do:
  get X,Y points
  StartPoly(X,Y)
  AddPolyPoint(X,Y)
if poly-complete,
  EndPoly()
**UIMS to Application Semantics**

**Associating Procedure Names**

- In the application program, the command is associated with a procedure name and event record

```plaintext
Procedure DoSemanticCommand(CommandNum: Integer; Evnt: EventRecord);
Begin
  Case CommandNum Of
    0: DeleteLine(Evnt);
    1: DrawLine(Evnt);
    2: DeleteCircle(Evnt);
    3: DrawCircle(Evnt);
    4: QuitProg(Evnt);
  End;
End;
End; { DoSemanticCommand }
```

**An Event Record**

```plaintext
Event = Record
  EventCode: Integer;
  MouseX, MouseY: Integer;
  EventValue: Integer;
  Time: Integer;
End;
```

where EventCode "1" for mouse button;
EventValue "2" for down

---

**UIMS to Application Semantics**

**Callbacks**

- Application Source
- Libraries
- Dialog Description
- UIMS Compiler
- Dialog Description Binary
- Run-time Program
- UIMS Run-time Routines
- Links
- Object Code
- Compile
- Executable Program
- Link
- Run-time Program
UIMS to Application Semantics

**Callbacks**

- **XWindow Code**
  ```c
  void EnterCallBack(CmndName, CmndProc)
  {
    SemanticCommand CmndProc;
    char * CmndName;
    LookUpCallBack(CmndName);
  }
  ```

- **Application Code**
  ```c
  EnterCallBack("DeleteLine", DeleteLine);
  EnterCallBack("DrawLine", DrawLine);
  EnterCallBack("DeleteCircle", DeleteCircle);
  EnterCallBack("DrawCircle", DrawCircle);
  EnterCallBack("QuitProg", QuitProg);
  ```

Implicit Main Event Loop

- No explicit main event loop: no "case" or "switch" or callback statements
- Abstract class called, for example, "WinEventHandler"
  - has methods which associate all windowing system events
    - SetCanvas, MouseDown, MouseMove, Redraw
  - O-O program creates a sub-class, an event handler object, for each window created
    - NewWindow(EventHandler)
- Each widget inherits its event processing from its parent
  - Example: Java, Tcl/Tk

Implicit Event Loop in Application (LISP CLOS)

```
(SETQ WorkWindow (CreateWindow 205 307 185 295 2))
(while (InRegionP (MouseCoords)
  (fetch ImageRegion AndGateDescr)
  and not (KEYDOWNP 'LSHIFT)
  do
  (replace CurrentCursorCoords (MouseCoords))
  (if (EQ (BUTTONSTATE) 'LEFT) then
    (RETFROM 'Tracker)
  ))
```
Implicit Main Event Loop

Tcl/Tk

- Each Tk widget is a window
- Each widget has pre-defined event handlers
  - Example: Button widget responds to mouse button
- Can attach a Tcl script to an event handler to process application semantics for widget
  - Example: Bind command
- Other events in event queue
  - “after” generates timer event (used for animation, etc.)
  - “fileevent” when file descriptor becomes readable or writable
  - Process redraws after input events

Tcl/Tk Example

dialog . (File Modified) (File "tcl.6" has been modified since the last time it was saved. Do you want to save it before exiting the application)? warning D (Save File) \ (Discard Changes) (Return To Editor)

Tcl/Tk Program

Dialog Box example

```
proc dialog {title text bitmap default args} {
    global button
    # 1. Create the top-level window and divide it into top # and bottom parts.
    toplevel $w -class Dialog
    $w title "$title"
    $w iconname $w Dialog
    frame $w::top -relief raised -bd 1
    pack $w::top -side top -fill both
    frame $w::bot -relief raised -bd 1
    pack $w::bot -side bottom -fill both
```
Tcl/Tk Program
Dialog Box example cont.

# 2. Fill the top part with the bitmap and message.
message $w.top msg -width 31 -text Start
-font -Adobe-Times-Medium-R-Normal-*=16-
pack $w.top.msg -side right -expand 1 -fill both
-pack $w.top.msg -side left -padx 3m -pady 3m

Tcl/Tk Program
Dialog Box example cont.

# 3. Create a row of buttons at the bottom of the dialog.
set 1
foreach but $buttons {break but -command
-apply but -text but -command
if (0) {
-apply but -text but -command
frame $w.but.default -relief sunk -bd 1
raise $w.but.default
-pack $w.but.default -side left -expand 1
-pack $w.but.default -side left -expand 1
}
else {
-pack $w.but.default -side left -expand 1
}
incr 1
}

Tcl/Tk Program
Dialog Box example cont.

# 4. Set up a binding for <Return>, if there's a default.
# set a grab, and click the focus too.
if ($default == 0) {
-bind $w -command "$w but button default flash" -apply $w -command "$w but button default flash"
}
-set $w.focus
grab set $w

# 5. Wait for the user to respond, then restore the focus
# and return the index of the selected button.
wait_variable button
disable $w

return $button
Summary

• All UIMS systems use an event model
• Events are typed
  – input, output, pseudo
• Events are filtered
  – Either windowing system or to application or none
• Events are stored in a priority queue
  – associated with a specific window in a hierarchy
  – passed to the application through an event record
• Application programs process these events
  – explicitly with a main event loop
  – implicitly in O-O languages with event handlers