Chapter 8

Implementation Support
Elements of windowing systems

• **Windowing System:**
  – windowing system is a system for sharing a computer's graphical display resources/ GUI among multiple applications at the same time.

• **Windowing Systems are:**
  – Device/Hardware Independent
  – Support Multi tasking
  – Management of independent but simultaneously active applications
  – A windowing system will have a fixed generic language which is called **Imaging Model**.

• Only one program **device driver needs to be written for a particular hardware device**
Windowing System Architecture

The client-server architecture
Toolkits

- A toolkit provides the programmer with a set of ready-made interaction objects – alternatively called interaction techniques, gadgets.

- These objects have a predefined behavior as that described for the button:

  ![Diagram of button interaction](image)

  **Figure 8.8** Example of behavior of a button interaction object

- The Java toolkit for developing windowed applications is called the Abstract Windowing Toolkit, AWT.
Toolkits (ctd)

- Toolkits provide Consistency and Generalizability for an interactive system.
  - One of the advantages of programming with toolkits is that they can enforce consistency in both input form and output form by providing similar behavior to a collection of widgets.
  - This consistency of behavior for interaction objects is referred to as the look and feel of the toolkit.
Toolkits

• To provide flexibility, the interaction objects can be modified
  – These objects are modified by ------------------

• Instantiation?
• Inheritance
• multiple inheritance?
• instance attributes?
Programming the application - 1
read-evaluation loop

repeat
  read-event(myevent)
  case myevent.type
    type_1:
      do type_1 processing
    type_2:
      do type_2 processing
    ...
  type_n:
    do type_n processing
  end case
end repeat
void main(String[] args) {
    Menu menu = new Menu();
    menu.setOption("Save");
    menu.setOption("Quit");
    menu.setAction("Save", mySave)
    menu.setAction("Quit", myQuit)
    ...
}

int mySave(Event e) {
    // save the current file
}

int myQuit(Event e) {
    // close down
}
User Interface Management Systems (UIMS)

• The set of programming and design techniques which provide more development support for interactive system design beyond the toolkits.
  – Examples of UIMS are Serpent & Picasso

• The UIMS should support:
  – **Conceptual Architecture:**
    • for the structure of an interactive system which concentrates on a separation between application semantics/logics and presentation;
  – **Techniques:**
    • for implementing a separated application and presentation and preserving the intended connection between them;
  – **Support techniques:**
    • For managing, implementing and evaluating a run-time interactive environment
UIMS: Conceptual Architecture

- Separation between application semantics/logic and presentation improving:
  - **Portability**
    - runs on different systems and provides device independent interface
  - **Reusability**
    - Reusability of components reduces development costs
  - **Multiple interfaces**
    - Supports development of multiple interface to access same functionality
  - **Customizability**
    - by designer and user without altering core application
UIMS: Conceptual Architecture

Seeheim Model
UIMS: Conceptual Architecture

MVC: model - view - controller
UIMS: Conceptual Architecture

Multi-Agent Architecture

PAC
presentation - abstraction - control
## MVC & PAC Differences/Issues

### PAC
- Groups input and output together.
- Secondly, PAC provides an explicit component whose duty it is to see that abstraction and presentation are kept consistent with each other.
- Not linked to any programming environment, though it is certainly helpful to an object-oriented approach.
  - It is probably because of this last difference that PAC could so easily isolate the control component; PAC is more of a conceptual architecture.

### MVC
- whereas MVC separates them
- MVC does not assign this important task to any one component, leaving it to the programmer/designer to determine where that chore resides
- Cannot so easily isolate the control component
Implementation Considerations

• Dialog Modeling Techniques in UIMS:
  – Menu Networks:
  – Grammar notations
  – State transition diagrams
  – Event Languages
  – Declarative languages
  – Constraints
  – Graphical specification
Case study

Worked exercise:
Scrolling is an effective means of browsing through a document in a window that is too small to show the whole document. Compare the different interactive behavior of the following two interaction objects to implement scrolling:

1). A scrollbar is attached to the side of the window with arrows at the top and bottom. When the mouse is positioned over the arrow at the top of the screen (which points up), the window frame is moved upwards to reveal a part of the document above/before what is currently viewed. When the bottom arrow is selected, the frame moves down to reveal the document below/after the current view.

2). The document is contained in a textual interaction object. Pressing the mouse button in the text object allows you to drag the document within the window boundaries. You drag up to browse down in the document and you drag down to browse up.

The difference between the two situations can be characterized by noticing that, in the first case, the user is actually manipulating the window (moving it up or down to reveal the contents of the document), whereas, in the second case, the user is manipulating the document (pushing it up or down to reveal its contents through the windows). What usability principles would you use to justify one method over the other (also consider the case when you want to scroll from side to side as well as up and down)? What implementation considerations are important?