IMPLEMENTATION SUPPORT IN HCI

Overview

- programming tools provide levels of services for programmers
- windowing systems as core support for separate and simultaneous user-system threads
- programming the application and control of dialogue
- interaction toolkits bring programming closer to level of user perception
- user interface management systems help to control relationship between presentation and functionality of objects

Introduction

- Up to now, our concern has been slanted away from concerns of the actual programmer.
- Advances in coding have elevated programming from hardware-specific to interaction techniquespecific.
- Layers of development tools
  - windowing systems
  - interaction toolkits
  - user interface management systems

Elements of windowing systems

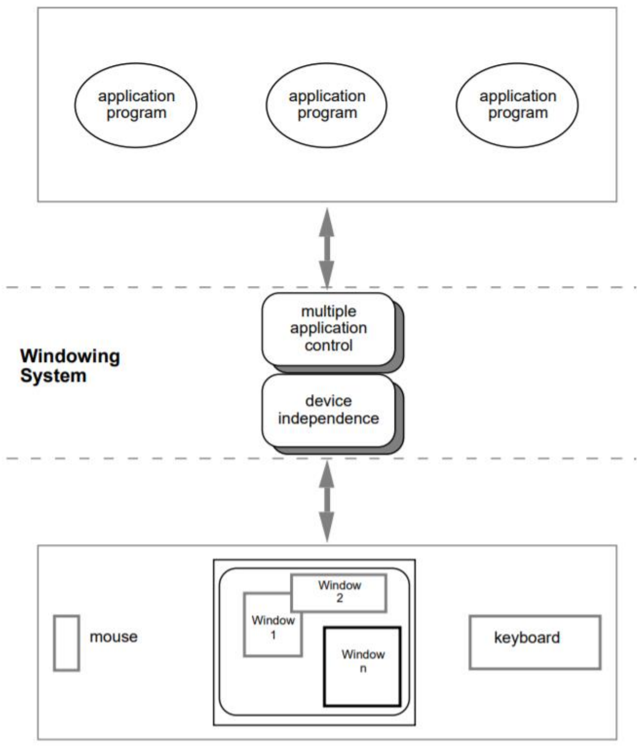
Device independence

- programming the abstract terminal device drivers image models for output and (partially) input
  - pixels
  - Graphical Kernel System (GKS)
  - Programmers' Hierarchical Interface to Graphics (PHIGS)
  - PostScript

Resource sharing

- achieving simultaneity of user tasks
- window system supports independent processes
- isolation of individual applications

The roles of a windowing system



Architectures of windowing systems

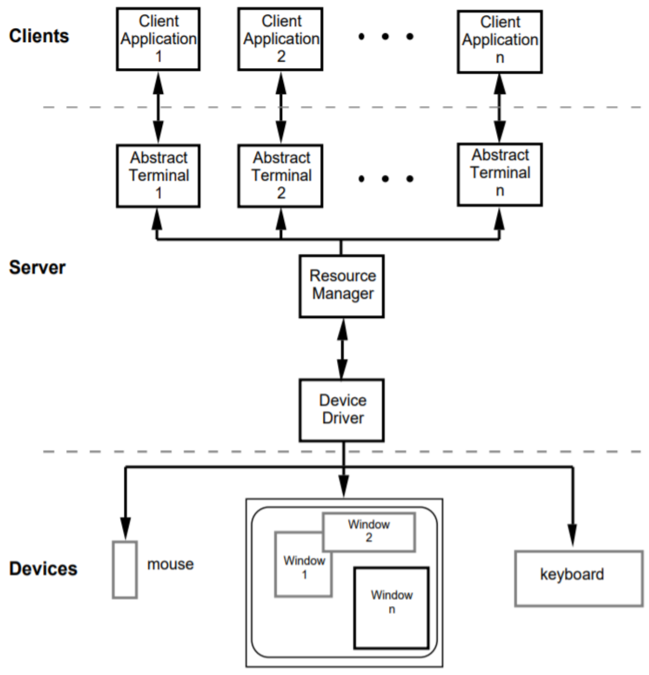
Three (3) possible software architectures

- all assume device driver is separate

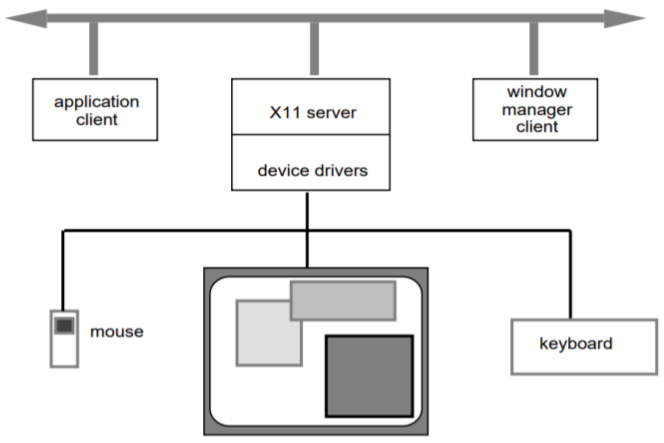
- differ in how multiple application management is implemented

1. each application manages all processes
   - everyone worries about synchronization
   - reduces portability of applications
2. management role within kernel of operating system
   - applications tied to operating system
3. management role as separate application
   - maximum portability the client-server architecture

The client-server architecture



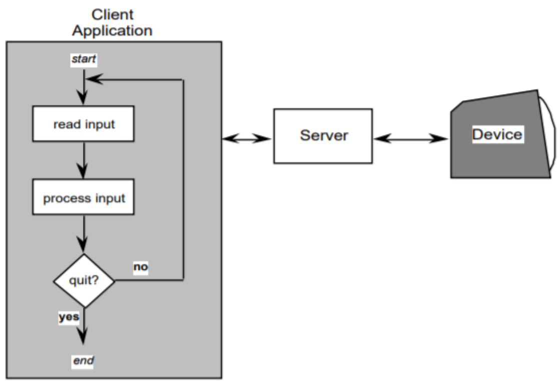
The X Window System architecture



- pixel imaging model with some pointing mechanism
- X protocol defines server-client communication
- separate window manager client enforces policies for input/output:
  - how to change input focus
  - tiled vs. overlapping windows
  - inter-client data transfer

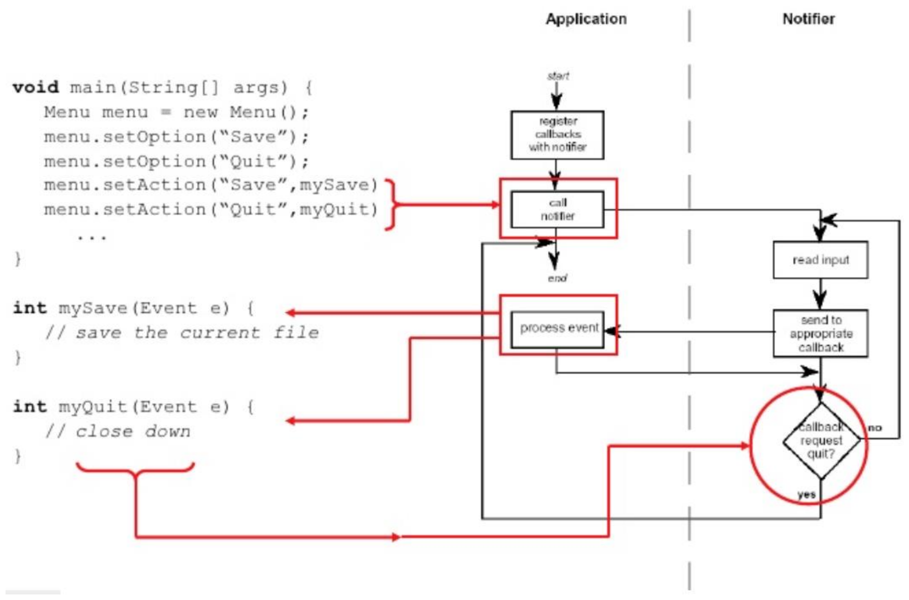
Programming the application

Two (2) programming paradigms:
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
```

2. notification-based



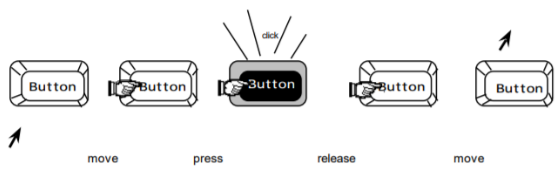
```
void main(String[] args) {
   Menu menu = new Menu();
   menu.setOption("Save");
   menu.setOption("Quit");
   menu.setAction("Save",mySave)
   menu.setAction("Quit",myQuit)
      ...
}

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

int myQuit(Event e) {
   // close down
}
```

Using toolkits

Interaction objects
- input and output intrinsically linked



Toolkits provide this level of abstraction
- programming with interaction objects (or techniques, widgets, gadgets)
- promote consistency and generalizability through similar look and feel
- amenable to object-oriented programming

User Interface Management Systems

UIMS add another level above toolkits
- toolkits too difficult for non-programmers alternatively:

    - UI development system (UIDS)
    - UI development environment (UIDE)

As a conceptual architecture

- provides separation between application semantics and presentation, improving:
    - portability, reusability, multiple interfaces, and customizability
- identifies roles (e.g., Seeheim)
    - presentation component
    - dialogue control
    - application interface model

Implementation of UIMS

Techniques for dialogue controller

- menu networks
- grammar notations
- state transition diagrams
- event languages
- declarative languages
- constraints
- graphical specification

The drift of dialogue control

- internal control (e.g., read-evaluation loop)
- external control (independent of application semantics or presentation)
- presentation control (e.g., graphical specification)