

HUMAN-COMPUTER INTERACTION THIRD EDITION
 DIX FINLAY ABOWD BEALE

Chapter 8

Implementation support

1

HUMAN-COMPUTER INTERACTION

Implementation support

- Programming tools
 - levels of services for programmers
- Windowing systems
 - core support for separate and simultaneous user-system activity
- Programming the application and control of dialogue
- Interaction toolkits
 - bring programming closer to level of user perception
- User interface management systems
 - controls relationship between presentation and functionality

2

HUMAN-COMPUTER INTERACTION

Introduction

How does HCI affect the programmer?

Advances in coding have elevated programming hardware specific
 → interaction-technique specific

Layers of development tools

- windowing systems
- interaction toolkits
- user interface management systems

3

HUMAN-COMPUTER INTERACTION

Elements of windowing systems

Device independence
 programming the abstract terminal device drivers, using image models for output and (partially) input

- pixels (series of columns/rows of points)
- PostScript (MacOS X, NextStep)
- Graphical Kernel System (GKS)
- Programmers' Hierarchical Interface to Graphics (PHIGS)

Resource sharing
 achieving simultaneity of user tasks by sharing the resources of a single hardware configuration with several copies of an abstract terminal

window system supports independent processes from the specifics of hardware devices
 isolation of individual applications from resource sharing

4

HUMAN-COMPUTER INTERACTION

Roles of a windowing system

Independence
 from the specifics of programming separate hardware devices

Management
 of multiple, independent but simultaneously active applications

5

HUMAN-COMPUTER INTERACTION

Roles of a windowing system (ctd)

```

    graph TD
        subgraph Applications
            A1((application program))
            A2((application program))
            A3((application program))
        end
        subgraph Control
            MAC[multiple application control]
        end
        subgraph Windowing_System [Windowing System]
            W1[Window 1]
            Wn[Window n]
        end
        subgraph Hardware [Hardware]
            M[mouse]
            K[keyboard]
        end
        A1 --> MAC
        A2 --> MAC
        A3 --> MAC
        MAC --> DI[device independence]
        DI --> W1
        DI --> Wn
        M --> W1
        M --> Wn
        K --> W1
        K --> Wn
    
```

6

Architectures of windowing systems

Three 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

7

The client-server architecture

8

X Windows architecture

9

X Windows architecture (ctd)

- Pixel imaging model assuming the existence of some pointing mechanism
- X network protocol clearly defines server-client communication, can be implemented on different computers and operating systems, making X more device independent
- Client and server need not even be on the same system, in order to communicate to the server
- Each client is associated to an abstract terminal or main window

10

X Windows architecture (ctd)

- The X server:
 - allows or denies access to the display from multiple client applications
 - interprets requests from clients to perform screen operations or provide other information
 - demultiplexes the stream of physical input events from the user, passing them on to the client
 - minimizes traffic along the network
- Separate window manager client enforces policies for input/output:
 - how to change input focus between applications
 - tiled vs. overlapping windows
 - inter-client data transfer

11

Programming the application 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
    
```

Application has complete control over the processing of events it receives. But this control must be executed for every possible event that the client may receive

12

Programming the application (ctd) 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
}
    
```

13

Going with the grain

- System style affects the interfaces
 - modal dialogue box (forcing user response)
 - easy with event-loop (just have extra read-event loop)
 - hard with notification (need lots of mode flags)
 - non-modal dialogue box (not forcing user response)
 - hard with event-loop (very complicated main loop)
 - easy with notification (just add extra handler)

Beware!
 If you don't explicitly design, it will just happen
 – implementation should not drive design

14

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

15

Object orientation

- Two features of interaction objects and toolkits make them amenable to an object-oriented approach
- Instantiation
 - define a class of interaction objects, which can be invoked many times within one application with minor modifications to each instance
- Inheritance
 - building complex interaction objects using simple interaction objects
- Use of classes as templates for interaction objects
 - a class template provides default values for attributes

16

Single inheritance class hierarchy for the Xview toolkit

```

graph TD
    Window --> Frame
    Window --> Tty
    Window --> Openwin["(Openwin)"]
    Window --> Scrollbar
    Window --> Icon
    Openwin --> Text
    Openwin --> Canvas
    Canvas --> Panel
    
```

17

Interfaces in Java

- Java toolkit – AWT (abstract windowing toolkit)
- Java classes for buttons, menus, etc.
- Notification based;
 - AWT 1.0 – need to subclass basic widgets
 - AWT 1.1 and beyond -- callback objects
- Swing toolkit
 - built on top of AWT – higher level features
 - uses MVC architecture (see later)

18

User Interface Management Systems (UIMS)

- UIMS add another level above toolkits
 - toolkits too difficult for non-programmers
- Concerns of UIMS
 - conceptual architecture for the structure of an interactive system, concentrating on separating application semantics and presentation
 - implementation techniques for the application and its presentation, preserving the intended connection between them
 - support techniques for managing, implementing and evaluating a run-time interactive environment
- Non-UIMS terms:
 - UI development system (UIDS)
 - UI development environment (UIDE)
 - e.g. Visual Basic

19

UIMS as conceptual architecture

- Separation between application semantics and presentation
- Improves:
 - portability - application runs on different systems, as device-dependent interface is separate
 - reusability - components reused cutting costs
 - multiple interfaces - accessing same functionality with different interfaces, enhancing interactive flexibility
 - customizability - by designer and user, thus increasing its effectiveness without altering the underlying application

20

How does application communicate with presentation

- This role of communication is referred to as *dialogue control*
- Now, conceptually we have the three main components of an interactive system:
 - the application
 - the presentation
 - the dialogue control

21

UIMS tradition - interface layers / logical components

- Linguistic: lexical/syntactic/semantic
- Seeheim:
- Arch/Slinky

The diagram shows a layered architecture. At the top, 'presentation' and 'application' are connected to a central 'dialogue' box. Below this, a semi-circular arch is divided into segments labeled 'func. core', 'lexical', and 'physical'. A 'functional core' is also indicated at the bottom left.

22

Seeheim model

The flowchart shows a linear flow from USER to Presentation (lexical), then to Dialogue Control (syntactic), then to Functionality (semantic), and finally to APPLICATION. A 'switch' box is positioned below Dialogue Control and Functionality, with bidirectional arrows connecting it to both.

23

Conceptual vs. implementation

Seeheim

- arose out of implementation experience
- but principal contribution is conceptual
- concepts part of 'normal' UI language

... because of Seeheim ...
... we think differently!

e.g. the lower box, the switch

- needed for implementation
- but not conceptual

The diagram shows 'presentation', 'dialogue', and 'application' boxes connected in a sequence. Below them is a smaller box representing the 'switch' component.

24

Semantic feedback

- Different kinds of feedback:
 - lexical - movement of mouse
 - syntactic - menu highlights
 - semantic - sum of numbers changes
- Semantic feedback often slower
 - use rapid lexical/syntactic feedback
- But may need rapid semantic feedback
 - freehand drawing
 - highlight trash can or folder when file dragged

25

What's this?

The diagram shows a flow from USER to Presentation (Lexical), then to Dialogue Control (Syntactic), then to Application Interface Model (Semantic), and finally to APPLICATION. A red circle highlights the Dialogue Control layer, indicating its central role in the interaction process.

26

The bypass/switch

This diagram illustrates a bypass mechanism. It shows the standard flow from Presentation to Dialogue Control to Application Interface Model. A red circle highlights a bypass path that allows for direct communication between the Application Interface Model and the Presentation layer, bypassing the Dialogue Control layer. This is labeled as 'rapid semantic feedback' and is noted as being 'direct communication between application and presentation but regulated by dialogue control'.

27

Arch/Slinky

- More layers! - distinguishes lexical/physical
- Like a 'slinky' spring different layers may be thicker (more important) in different systems
- Or in different components

The diagram shows a semi-circular arch structure representing layers of abstraction. From top to bottom, the layers are: 'dialogue', 'lexical', and 'physical'. On the left side, there are 'func. core adaptor' and 'functional core' components.

28

Monolithic vs. components

- Seeheim has big components
- Often easier to use smaller ones
 - esp. if using object-oriented toolkits
- Smalltalk used MVC - model-view-controller
 - model - internal logical state of component
 - view - how it is rendered on screen
 - controller - processes user input

29

MVC model - view - controller

The diagram shows the MVC pattern. A central 'User' interacts with a 'View' (via a 'Display') and a 'Controller' (via a 'Mouse' and 'Keyboard'). The 'View' and 'Controller' are interconnected, and the 'Model' is connected to both. The 'User' is also connected to the 'View' and 'Controller'.

30

MVC issues

- MVC is largely pipeline model:
input → control → model → view → output
- But in graphical interface
 - input only has meaning in relation to output
e.g. mouse click
 - need to know *what* was clicked
 - controller has to decide what to do with click
- but view knows what is shown where!
- In practice controller 'talks' to view
 - separation not complete

31

PAC model

- PAC model closer to Seeheim
 - abstraction - logical state of component
 - presentation - manages input and output
 - control - mediates between them
- Manages hierarchy and multiple views
 - control part of PAC objects communicate
- PAC cleaner in many ways ...
but MVC used more in practice
(e.g. Java Swing)

32

PAC presentation - abstraction - control

33

Implementation of UIMS

- Techniques for dialogue controller
 - menu networks
 - grammar notations
 - declarative languages
 - graphical specification
 - state transition diagrams
 - event languages
 - constraints
- for most of these see chapter 16
- N.B. constraints
 - instead of what *happens* say what should be *true*
 - used in groupware as well as single user interfaces
(ALV - abstraction-link-view)

see chapter 16 for more details on several of these

34

Graphical specification

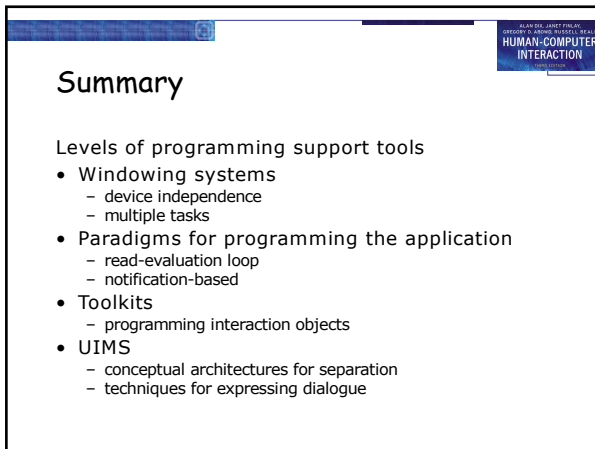
- What it is
 - draw components on screen
 - set actions with script or links to program
- In use
 - with raw programming most popular technique
 - e.g. Visual Basic, Dreamweaver, Flash
- Local vs. global
 - hard to 'see' the paths through system
 - focus on what can be seen on one screen

35

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)

36



Summary

Levels of programming support tools

- Windowing systems
 - device independence
 - multiple tasks
- Paradigms for programming the application
 - read-evaluation loop
 - notification-based
- Toolkits
 - programming interaction objects
- UIMS
 - conceptual architectures for separation
 - techniques for expressing dialogue

37