CS 66-460
Introduction to Graphical Human Machine Interfaces

An Overview of User Interface Software

MONDAY, FEBRUARY 3, 1997

Instructor: G. Bowden Wise

Rensselaer Polytechnic Institute
Spring 1997

Review ....

So far we have been looking at **design** aspects

- everyday objects
- models/metaphors
- usability

The claim is that to design **good interfaces**, we must

involve the user as much as possible

Remember the iterative nature of the HCI lifecylce:

- design
- build
- evaluate
- re-design ...

Today we begin looking at the development (implementation) side of user interfaces.

Today's user interfaces (GUIs) are particularly complex

Using and understanding **user interface software tools** will help us implement user interfaces.

Why Are UIs So Hard to Build/Implement?

- UI Software is difficult and expensive to create.
- UI Software accounts for more than 50 percent of:
  - design effort
  - implementation effort
  - maintenance effort
  - and code size
- They are hard to design and require iterative implementations
  - Create prototype
  - Evaluate with real users
  - Re-design and re-implement based on feedback
- They are "reactive" and programmed from the "inside-out"
  - Must respond to events
  - More difficult to modularize
    callbacks; changes to UI impact application

Why Are UIs So Hard to Build/Implement? (continued)

- They require multi-processing
  - Must deal with user typing
  - Must respond to mouse movements
  - Must deal with aborts / undo
  - Must redraw (refresh) windows asynchronously
  - Must deal with multiple input devices
  - UI must accept events at anytime
  - UI encounters same problems with multi-processing (synchronization, maintaining consistency among multiple threads, deadlocks, etc)
- They have real-time requirements
  - Output 60 times per second
  - Keep up with mouse tracking
  - Multimedia: video, animations, sound, speech
- Need for robustness
  - No crashing on any input
  - Helpful error messages, graceful recovery
  - Allow abort and undo

Why Are UIs So Hard to Build/Implement? (continued)

- Lower testability (difficult to test direct manipulation)
- Little language support
  - Primitives found in languages are bad for UIs (printf/scanf)
  - Special, complex, and enormous libraries required
  - Features like OOP, constraints, and collaborative systems
- Complexity of tools
  - 6 books for Macintosh Toolbox
  - 6+ for X Windows (?)

Some Definitions

- **User Interface**

  ... of a computer program is the part of the program that handles the output to the display and the input from the user
- The rest is called the **application** or the **application semantics**
- Four different classes of users involved with UI software:
  - The **user** or **end user** is the person using the finished program
  - The **user interface designer** creates the user interface
  - The **application programmer** creates the application code
  - UI tools are developed by **tool creators**.
- The designer may utilize UI tools to help create user interfaces.

Importance of UI Tools: Better Quality Interface

- Designs can be rapidly prototyped and implemented, possibly even before application code is written
- Easier to incorporate changes during testing
- It is justifiable to spend lots of time using the tool, since many applications will be developed using it
- Different applications are likely to have consistent user interfaces since they are developed from the same tool
- A UI tool makes it easier for a variety of specialists to be involved in the design and implementation of the UI
- Features like UNDO and HELP are more likely to be available since they might be provided by the tools.

Importance of UI Tools: Easier To Create And Maintain

- Interface specifications can be represented, validated, and evaluated more easily
- Less code to write since much will be provided by the tool
- There will be better modularization due to the separation of the UI component from the application
  - should be possible to change UI or APPLICATION without impacting the other
- The level of expertise of the interface designers and the implementors may be lower because the tools may hide some of the complexity of the underlying software
- The reliability of the UI will be higher since the code for the UI is created automatically from a higher level specification
- It will be easier to port an application to different software/hardware platforms since they device dependencies are isolated into the UI tool

What Should Tools Do?

Based on the goals on the previous pages, we can come up with a number of features that tools might provide

These features can be used to **evaluate** different tools

Note: no tool will have everything; and different designers/developers have different needs.

In general, the tools might:

- help the design of the interface given a specification of the end user's task
- help implement the interface given a specification of the design
- help evaluate the interface after it is designed and propose improvements
- create easy-to-use interfaces
- allow the designer to rapidly investigate different designs
- allow non-programmers to design and implement UIs
- allow end-users to customize the interface to their liking
- be easy to use themselves

How Might Tools Do That?

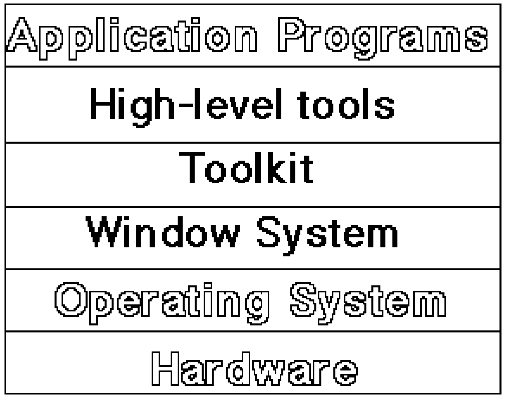
Perhaps by having the tools:

- automatically choose which user interface styles, input devices, widgets, etc to use.
- help with screen layout and graphic design
- validate user inputs
- handle user errors
- handle aborts and undoing of operations
- provide appropriate feedback to show that inputs have been received
- provide help and prompts
- notify the application when the user modifies graphical objects
- handle field scrolling and editing
- help with the sequencing of operations
- insulate the application from all device dependencies and the underlying software and hardware platforms
- provide customization features for end users
- evaluate the graphic design, layout, usability, and learnability of the interface

Components of UI Software

Layers:

- Application
- Higher Level Tool
- Toolkit
- Windowing System
- Operating System
- Hardware



Windowing System

- supports separation of the screen into different (rectangular) regions called windows
- Can be:
  part of a program (Smalltalk)
  part of the OS (SunTools)
  a separate program (X)
- Often divided into two layers:
  - **window system**
  - **window manager**

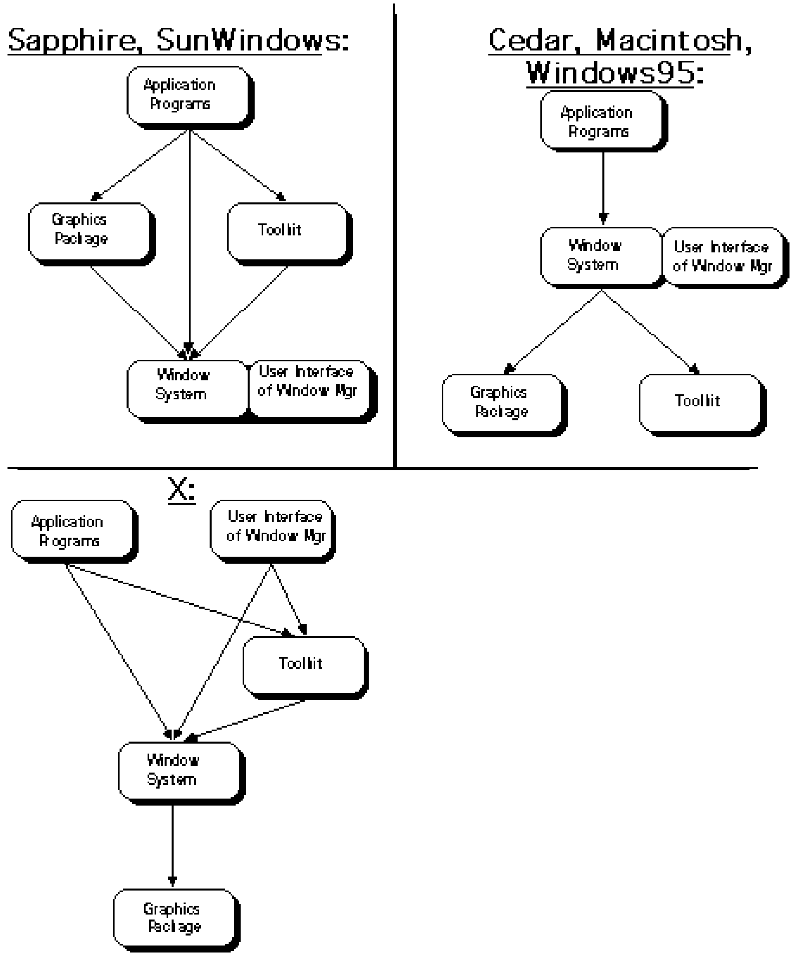
Windowing System Layers

- The **Window System** provides a programmer interface (API) that allows
  - output of graphics to draw on the screen to a clipped region (window); this is the output model
  - input channels from mouse/keyboard to appropriate window; this is the input model
- The **Window Manager**
  - allows end users to move windows, resize them, iconify them, select them, etc
  - provides decorations on windows, such as, titles, borders, colors, icons, etc
- often the Window Manager (WM) and Window System (WS) are blurred into one, since many systems (SunTools, Macintosh, Windows, NeXT) do combine them as one
- Others allow different WM on same WS (X, NeWS)

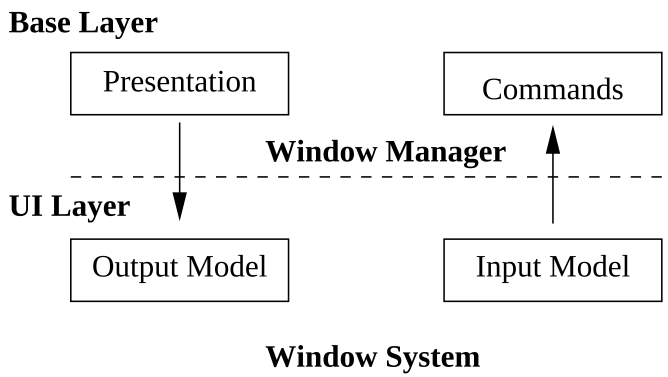
WM and WS Evolution

- Early systems
  - tightly coupled WM and WS
  - assumed sophisticated graphics and toolkits would be built on top
  - integral part of OS
  - Sapphire, SunWindows
- Next generation
  - incorporate into the window system the graphics and toolkits
  - this gave the window manager the same sophistication and look and feel as applications, applications would look more consistent
  - Cedar, Macintosh, NeXT, Windows95
- Current
  - allows different WM and different toolkits
  - still embeds sophisticated graphics packages
  - allows different windowing system on same OS
  - uses IPC and network for connection to applications
  - NeWS, X

Organizations of Windowing Systems



Structure of Windowing Systems



Structure of Windowing Systems (continued)

The **base layer**

- implements the basic functionality of the windowing system
  - provides primitives for generating pictures, text on the output display: this is the output model
  - provides mechanisms for handling user input (keyboard, mouse): this is the input model
  - primary interface to the base layer is through a procedural interface, i.e., an application programmer interface (API)

The **UI layer**

- includes all aspects that are **visible** or apparent to the user. Including,
  - presentation: what the user sees, windows, pictures, text, etc.
  - commands: the operations the user can use to manipulate windows and their contents

Window System: Output Model

- Graphics primitives (procedures) that an application can call to draw pictures, text, etc.
- All usually directed to a particular *window* so the drawing is "clipped" to the bounds of the window
- Usually can only draw objects of the types provided by the window system
- Older systems (SunTools)
  - simple primitives
  - draw rectangles, text, lines
  - operations on pictures: RasterOp or BitBlt
  - advantage: easy to implement
- Newer systems (Macintosh, X)
  - still have rectangles, lines, and text, but also
  - filled polygons, splines, color, clipping
  - all 2D objects, but extensions often available for 3D objects
  - advantage: prettier images are easier for application to create

Window System: Output Model (continued)

There are various graphics standards or styles:

- Postscript
  - language invented by Adobe for sending pages to printers
  - a complete textual programming language
  - provides:
    - arbitrary rotation and scaling
    - complete hardware independence (coordinates are floats)
  - used as an output model for some windowing systems:
    - NeWS, DisplayPostscript, NeXT, DEC
- OpenGL
  - 3D model from Silicon Graphics
  - powerful rendering capabilities
  - provides independence for 3D objects
  - standard part of: Windows, X

Window System: Output Model (continued)

- Other standards
  - CORE (ran on top of SunView)
  - PHIGS
  - PEX = PHIGS + 3D for
- Alll new systems implement a graphics style **as part** of the windowing system (Macintosh, X, NeWS, Microsoft Windows).

Window Systems: Input Model

- Earlier systems (e.g., CORE/PHIGS) provided an input model that did not work well with the direct manipulation style of interaction
  - For example, programs called a routine to request a value using a "virtual device": such as a "locator" for a pointing device position; a "string" for edited text; a "choice" for a menu selection; or a "pick" for selecting an object
  - The problem is the model forced the system to wait until the user provided input.

Window Systems: Input Model (continued)

- Modern systems employ a "mode-free" style of interaction whereby the user can decide whether to make a menu choice, select an object, press a button, when desired.
- To handle the asynchronous, mode-free style of interaction, modern systems employ an **event-driven** input model
  - a stream of events are sent to the currently active window (the one designated to receive events)
  - an event record includes
    * event type (what kind of event): WindowClose, WindowDestroy, LeftButtonDown, KeyUp
    * mouse coordinates (x,y)
    * timestamp
    * key modifiers (e.g., is SHIFT or ALT pressed)
  - the windowing system queues keyboard events, mouse button events, mouse movement events, and window operation events, and other special events to windows, and their programs must decode the events and process them
  - events are asynchronous

Window Systems: Input Model (continued)

Problems with event-driven model:

- Application must always be able to accept events
- Race conditions may occur since events are asynchronous
- Not device independent. Suppose you wanted to add a 5-button mouse to your system. Because the event records have fixed fields, there is no way for the standard event mechanism to determine the state of the extra buttons.
- no provision for special "stop output" (control-S) or "abort" (control-C)

Window Systems: Communication

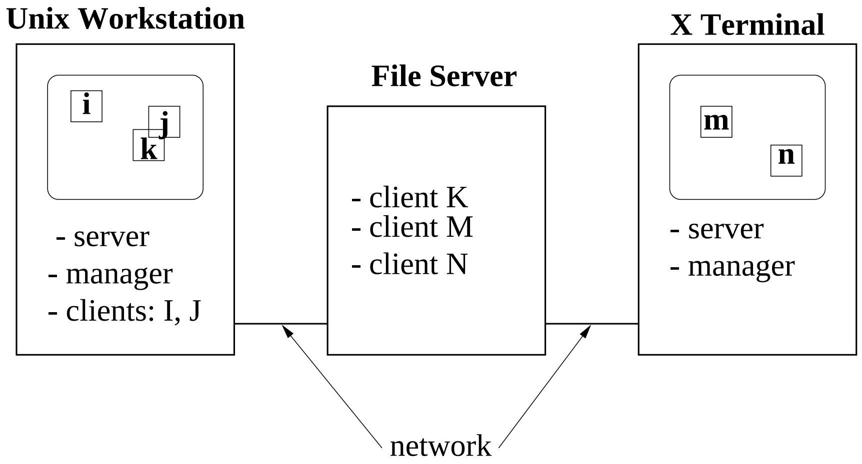
- Window systems often are protected processes so bad applications will not crash entire machine.
  Not true for: Windows 3.11, Macintosh
  True for: X, WindowsNT
- How do applications communicate with the window system?
  - OS kernel – Sapphire, SunView
  - Network protocols (e.g., TCP/IP, sockets) – SunTools, Macintosh
  - Send messages to processes via network – X, NeWS

Window Systems: Communication in X Windows

In X, messages are sent to processes via the network.

- The X mechanism has many advantages:
  - Application programs (clients) can be on a different machine than it's windows and window manager (server)
  - Makes it easier for a person to utilize several machines, but have all of the windows appear on a single machine
  - Can utilize different languages to implement applications:
    C with XLib
    Lisp with CLX
- The disadvantage of the X mechanism is efficiency:
  - Each window event is encoded, passed to the network, and then decoded by the client, even when the server and client are on same machine.

A Simple Example of the X Protocol



Notes:

- Each display has an associated server
- Clients may be run on same machine as server but do not have to
- The window manager is usually run on the same machine as the X server, but this is not required

Window Manager: Window Decorations

- how the windows are arranged and decorated
- windows can be tiled or overlapping
  - specifies whether windows can be on top of one another
  - don't see tiled much anymore: HTML frames are sort of like tiled, old Microsoft Windows, and Cedar had tiled windows
  - overlapped windows actually appeared first (Smalltalk, 1976)
  - a study showed tiled windows to be more efficient than overlapping
  - overlapping is the dominant style in most window managers today, perhaps due to its similarity to the desktop metaphor (i.e., piles of papers)
- Decorations:
  - titles, colors, window backgrounds, virtual workspaces, icons
- In X, you may choose your window manager. There are many to choose from: `olw, vwm, twm, ctwm, fvwm, mwm`

Window Manager: Commands

- how the user controls the windows
- mouse and keyboard commands
- interaction with menus, buttons, etc
- one window must be the **listener** or have the **focus**: the chosen window receives keyboard and mouse events
- how is the focus determined?
  - **click-to-type**: user must click mouse in window to select
    Macintosh, Microsoft Windows
  - **move-to-type**: user must place mouse over desired window, no need to click:
    X
  - In X, some window managers let you choose which style you prefer
- window commands
  - move to front, send to back
  - resize, move, iconify/minimize
  - destroy, close