CS 66-460
Introduction to Graphical Human Machine Interfaces

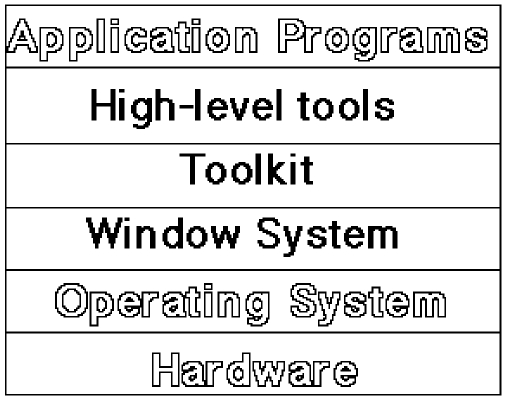
An Overview of User Interface Software (Part 2)

WEDNESDAY, FEBRUARY 5, 1997

Instructor: G. Bowden Wise

Rensselaer Polytechnic Institute
Spring 1997

Recall The Component Layers of UI Software



Next we look at **toolkits** ...

Toolkits

- A **widget** is a way of using a physical input device to input a certain type of value.
- A **toolkit** is a library of interaction techniques called by applications.
- Toolkits
  - contain procedures to create and manipulate various widgets:
    * menus, buttons, scroll bars, dialog boxes, windows, icons, text boxes
  - provide a procedural interface
  - are used only by programmers
- Example toolkits:
  - Macintosh Toolbox
  - Microsoft Foundation Classes
  - Xtk for X (Motif and OpenLook both use Xtk)
  - Interviews for X (in C++)
  - NeXTStep for NeXT
  - tk part of tcl/tk
  - Amulet

Toolkits: Advantages and Disadvantages

Advantages:

- gives applications a consistent look and feel
- allows for re-use of code
- programmers do not have to write code for standard functions, such as menu selection or drawing a button

Disadvantages:

- styles of interaction are limited to those provided by the toolkit
  how would you create a "slider" with two indicators, one for the upper value, and another for the lower value? toolkit does not provide everything
- expensive to create
  the primitives themselves do not seem all that complex, but applications written with toolkits become very intricate and complicated
- difficult to use
  very large libraries
  lengthy manual sets
  not clear when and how to call what

Toolkits: Layers

Because designers of X could not agree on a single look-and-feel, they created an **intrinsics layer** on which to build different **widget sets**.

- **intrinsics layer**:
  - the procedural interface for implementing widgets
  - provides common services, such as techniques for object-oriented programming, and layout control
- **widget set**:
  - provides a collection of widgets which are built using the intrinsics
  - provides a particular look and feel

Toolkits: Intrinsic Layer

- The intrinsics are look-and-feel independent
- Widgets built on top can have any desired appearance and behavior
- Come in two styles of programming:
  - procedural
  - object-oriented

Procedural

- libraries of procedures
- Examples: Macintosh Toolbox, SunTools for SunView
- Advantage: simple to implement

Toolkits: Intrinsic Layer

Object-Oriented

- library defines standard classes
- programmers can specialize by sub-classing
- requires the use of an object-oriented langauge (e.g., C++)
- Examples: Smalltalk, Xt, Interviews, Garnet
- Advantage: Natural way to think about widgets since widgets; menus and buttons (the widgets) seem like objects
- Advantage: Easier to customize (sub-classsing)
- Disadvantage: requires a special (single?) programming language

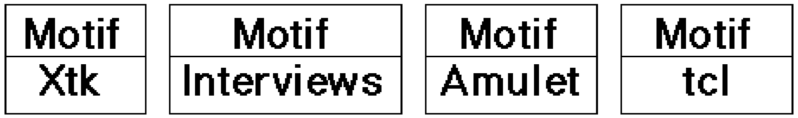
Object-Oriented toolkits must implement the objects either by inventing its own object system or using an existing object system:

- Xt, Garnet, Amulet use their own object system
- Interviews uses C++
- NeXTStep uses Objective-C
- AWT uses Java

Toolkits: Widget Sets

- Widget sets are implemented using intrinsics.
- Widget sets must have a particular look and feel.
- Different widget sets (with different looks and feels) can be implemented on top of the same intrinsics:

- Or, the same look and feel can be implemented on top of different intrinsics:



  - Although they look and operate the same, they are implemted quite differently, and have completely different procedural interfaces for the programmer.

A Brief Look at Widgets From Different Toolkits

- Microsoft Windows
- X Windows: Athena Toolkit
- X Windows: Motif Toolkit
- X Windows: OpenLook Toolkit

Virtual Toolkits

- The widgets found in various toolkits are pretty much the same, just subtle cosmetic differences.
- Suppose you want to make your application available on several platforms:
  - X Windows, Macintosh, Microsoft Windows
- One way to do this might be to port the application to three different toolkits:
  - X Windows: using Motif
  - Macintosh: using Macintosh Toolbox
  - Microsoft Windows: using the Microsoft toolkit
- But ... it is a lot of work to take a Motif application and convert it to the Macintosh ... and then to Microsoft Windows
- Instead use a **virtual toolkit**
  - allows the programmer to write the code *once* using **virtual widgets**
  - the code runs without change on the various platforms supported
- virtual toolkits are also called **cross-platform development systems**

Virtual Toolkits (continued)

There are two styles of virtual toolkits:

Actual widgets

- the virtual toolkit links to the actual toolkit on the host machine
- **XVT**
  - provides a C or C++ interface that links to the actual widgets of Motif, OpenLook, Macintosh, Microsoft Windows, and OS/2 Presentation Manager

  ADVANTAGE
  produecs UIs that are look-and-feel conformant
  DISADVANTAGE
  must provide an interface to graphical drawing primitives on each platform
  DISADVANTAGE
  tend to only provide widgets/functions common to all toolkits; toolkit specific style features are lost

Virtual Toolkits (continued)

Re-implemented widgets

- the virtual toolkit re-implements the widgets for eah supported platform
- Examples: **Galaxy** and **OpenInterface**:
  - each re-implements all of the widgets on the various platforms

  DISADVANTAGE
  large run-time library

Application Frameworks

Toolkits are difficult for programmers to use:

- how and when are the various toolkit functions called
- are the correct user interface guielines being followed for the platform

Application frameworks are very popular now:

- usually object-oriented
- classes are provided for all of the various widgets
  - programmers specialze the classes (by using inheritance) to implement behavior unique to their application
- classes are also provided for other important features of the platform and hide much of the complexity of the underlying API

  application frameworks **encapsulate the API**

- often contain interface builders

Application Frameworks: Examples

- Microsoft Windows: Microsoft Foundation Classes (MFC) and Borland's Object Windows Library (OWL)
- Macintosh: CodeWarrior PowerPlant

Other frameworks are for specific types of applications:

- graphical editors/applications: Unidraw, Amulet
- graph programs: Edge, TGE
- network programming: ACE

Event Driven Programming

- Older character-based or form-based user interfaces created a **sequential, linear** interaction mechanism
  - when presented with a selection menu, users had no choice but to make a selection
  - it was not possible to manipulate some other aspect of the application
  - the **application was in control**
- A sequential, linear interaction mechanism does not work well with the **direct manipulation** style of interaction found on most GUIs
  - users see visual representations of objects
  - users manipulate the objects directly to perform commands/actions at any time the user desires
  - the **user is in control**
- Applications cannot pause execution and just wait for the user to make a selection
- Instead, an **event-driven** model of interaction is used

Event Driven Model of Interaction

- Applications (usually several simulatenousluy) run, awaiting for **events**:
  - Mouse button events: LeftButtonDown
  - Mouse movement events
  - Keyboard events: KeyUp, KeyDown
  - Windowing events: resize, close, destroy, move, minimize
- Applications adopt the motto "don't call us (the API), we'll call you"
- When the user presses on the left mouse button, for example, a LeftButtonDown event is generated, this results in the API calling application code in response to the button press
- Several ways to implement **event handling**:
  - switch/case statements (Macintosh)
  - callbacks (X Windows)
  - window procedures (Microsoft Windows)
  - object-oriented techniques

Event Handling: switch/case statements

- Switch or case statements are common on the Macintosh
- The Macintosh Toolbox consists of several modules, one is the **EventManager** which is responsible for inserting events into the event queue of each application
- Queues behave in a "first-in-first-out" (FIFO) fashion, however, some events have higher priority and are to be processed sooner than their FIFO ordering
- Once an application has initialized itself, it enters a **main event loop**
  - the application repeatedly polls its event queue for new events
  - the type of the event is examined to determine what actions the program is to take
  - there is usually a large block of switch/case statements which calls the correct routines based on the event type

A Typical Event Loop on the Macintosh

```
 // initialize application
 ...

// main event loop
 do
 {
    // get an event from the queue
    getnextevent (eventmask, &event);

    // handle the event
    switch (event.type)
    {
    case mousedown:
       DoMouseDown(event);
       break;

    //  lots and lots of case statements !

    // quit
    case quitmessage:
       done = true;
       break;
    }

 } while (!done);
```

- the event `mask` is used to filter events out events which are not of interest to the application

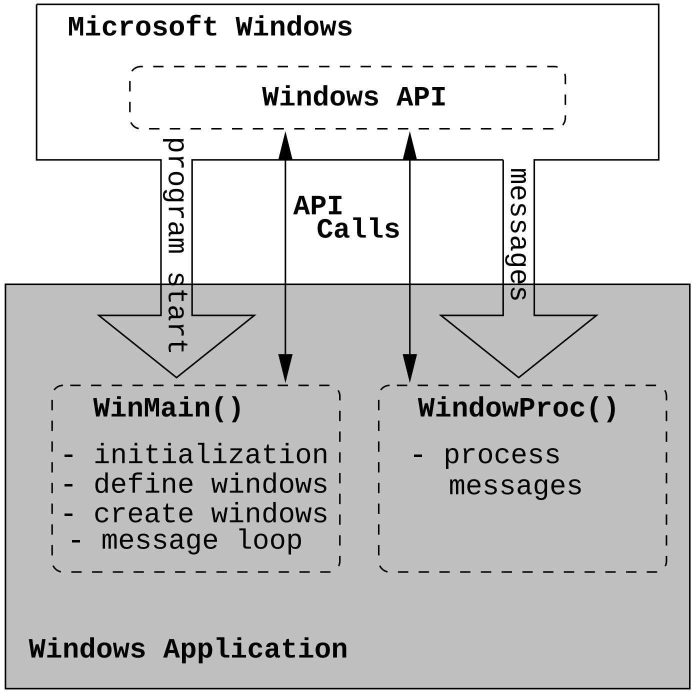
Microsoft Windows

- Events in window are called **window messages**
- Each application has its own **message queue**
- Each widget (button, window, etc) in Micrsoft Windows has its own procedure for processing messages sent to it called the `WindowProc`
- The Windows operating system places some messages into an application's queue, while other messages, are sent directly to the appropriate `WindowProc()`
- Each application must have an event loop that polls the queue for new messages:

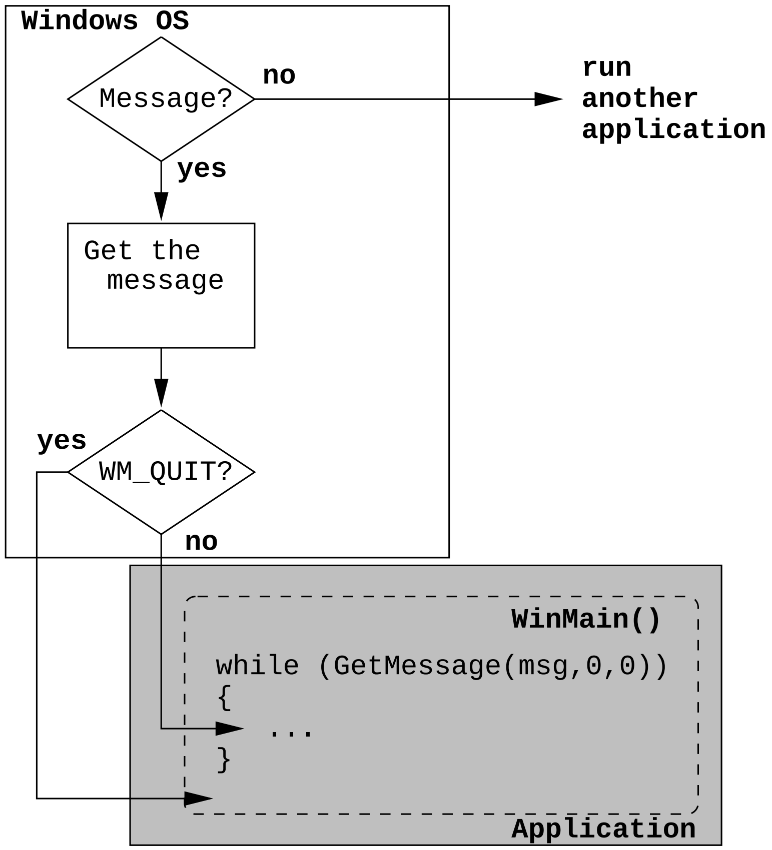
```
MSG msg;
while (GetMessage (&msg, 0, 0, 0) == TRUE)
{
   TranslateMessage(msg);
   DispatchMessage (msg);
}
```

- User interface programmers must write code to perform application specific processing for messages of interest and place the code in the `WindowProc` for each window
-
- Un-interesting messages can be passed back to the Windows operating system for default processing by calling:
  `DefWindowProc ()`

Microsoft Windows: Event Handling

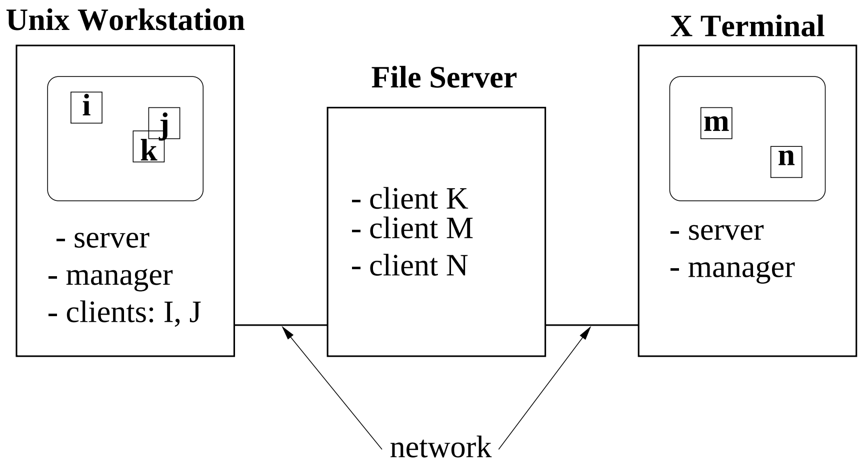


Microsoft Windows: Message Loop



- Periodic calls to `GetMessage()` are necessary to prevent an application from taking control of the machine.

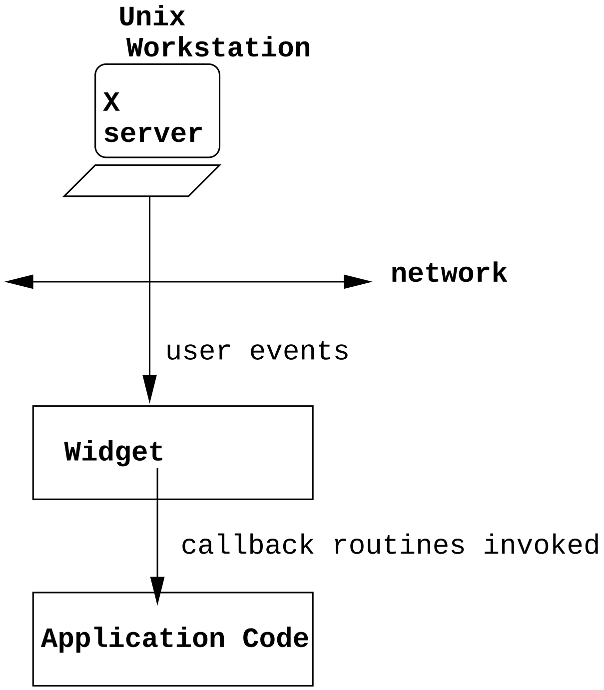
X Window System Protocol



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

Event Handling in X Windows



- Events are sent from the server to the client over the network.
- The event reaches the widget code which then calls a special procedure in the application code called the **callback**
- Callbacks must be registered with X during program initialization

Event Handling in Object-Oriented Frameworks

- Recall that **application frameworks** consist of object-oriented class libraries that
  - encapsulate the widgets
  - encapsualte the complexity of the API
- Programmers use sub-classes (inheritance) to implement their programs with application-specific behavior.
- Beneath the toolkit, the event mechanism of the underlying window system is still the same.
- However, instead of callback procedures or code inside of `WindowProc`s, the framework calls special class member funtions, called **handlers**, when a message is received.
- Programmers must code these handler routines in order for the widgets in their application to respond to events.

Event Handling in Object-Oriented Frameworks

- For example, if we were using Microsoft Foundation Classes, and we wanted to add code so that when the "submit" button is pressed, we would add a handler for the `BN_CLICKED` message so that it would call the `OnClickSubmit()` method when the message is received:

```
void CMyDialog::OnClickSubmit()
{
// ...
}
```

- Some classes already provide a **virtual** function for certain messages. For example, all windows have a routine called `OnPaint()` that is called whenever a `WM_PAINT` message is received. This method is part of the class tt CWnd. So if you want to specialize the redrawing behavior of your window, you need only provide a method called `OnPaint()` in your derived class. You do not necessarily need to add the handler since one is already provided by the base class.