What's to Come

- Wrap-up some loose ends:
  - User Interface Software: Higher Order Tools
  - Usability Testing and Evaluation
- Advanced Topics
  - We will look at some readings
  - In-class discussions; you will each need to bring 3 discussion questions to these sessions
  - Your paper and presentation

Possible Advanced Topics

- Hypertext and the WW; Web Design/Usability
- Alternative Modalities:
  non-speech audio, speech, haptic, pen
- Computer Supported Co-operative Work (CSCW)
- 3D Interaction
- Virtual Reality and Telepresence
- Designing for Diversity: disabled, young, and old
- Information and Data Visualization
- New Computing Paradigms:
  Non-WIMP interfaces, wearable computers

If you would like a specific topic covered; let me know and I will try to cover it.

Programming Projects

Only one more **moderate project** (due at end of term):

– Java and JavaAWT

– we will talk about Java in class

– we will do parts of the on-line Java tutorial

Probably 1 or 2 short **mini-projects**:

– Maybe go through a tutorial

– Create a simple interface to get an idea how the tool works

– Ideas:
  - SUIT
  - tcl/tk
  - SILK (if I can get it)
  - VisualAge (C++ OO framework on OS/2)

Also ....

We may have a few more ...

- videos

- In-class exercises and write-ups (probably 2)

Weekly electronic discussions on continues:

- on `rpi.courses.spring97.GHMI`

CS 66-460
Introduction to Graphical Human Machine Interfaces

An Overview of User Interface Software (Part 3)

MONDAY, MARCH 17, 1997

Instructor: G. Bowden Wise

Rensselaer Polytechnic Institute
Spring 1997

Review

- Interfaces are hard to design and create
- There is a need for UI tools

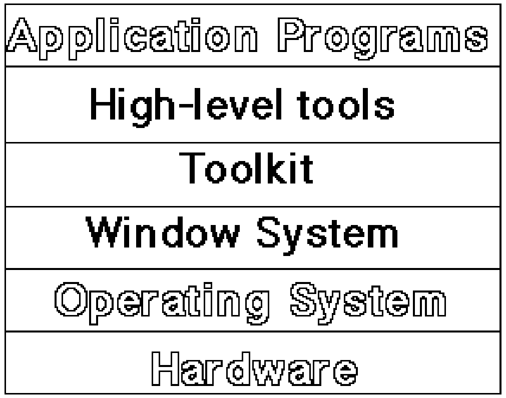
- Component layers of UI software
- Event driven programming

- Architecture of GUIs
  - window systems
  - window managers

- Implementation
  - X Windows toolkit programming

- Example systems
  - Microsoft Windows
  - X Windows
  - object-oriented frameworks
  - prototyping tools: SILK

The Component Layers of UI Software (Review)



Categories UI Software Tools

- **Toolkits:** widget sets, intrinsics
- **Virtual Toolkits**
- **Higher level tools**
  - prototyping tools
  - interface builders
  - application frameworks
  - user interface development environments

Toolkits (Review)

- 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
  - interface to applications via "callback" procedures

- Give applications a consistent look-and-feel
- Provdes re-use of code since widgets share code

- Very large libraries
- Complicated calling conventions and protocol
- Hard to use

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
- Example: **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
produces 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-Implements the widgets

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

DISADVANTAGE
large run-time library

Virtual Toolkits (continued)

- Most toolkits that work on multiple platforms can be considered virtual toolkits of the second type (re-implements widgets).

- However ...

- **SUIT, Garnet**:
  - work on X, Macintosh, and Microsoft Windows
  - provide same look-and-feel across all platforms
  - do not look like other applications that run on that platform
  - therefore not considered virtual toolkits

- The **JavaAWT** toolkit can be considered a virtual toolkit
  - programmers write the code once
  - and it runs on any platform with a Java virtual machine

Higher Level Tools

- Toolkits are hard to use; need higher-level support

- **Prototyping Tools**
  - Quickly see how UI is going to look and act
- **Interface Builders**
  - Layout widgets
  - Create dialog boxes, menus, and other windows
- **Application Frameworks**
  - Object-oriented libraries for UI construction
- **User Interface Development Environments**
  - Provide comprehensive support for UI development

- Tradeoffs among tools
  - Ease of use vs. power
  - Range of interfaces vs. amount of help (if narrow, can provide more support)

- Research shows that using interactive tools is 10 to 50 times faster than strict toolkit programming.

Prototyping Tools

- Goal: create a mock-up of the interface and see it very quickly
- Allows one to try out different screen designs and see what they look like
- Usually do not create actual interfaces or use real widgets
- Usually does not require programming; can be used by graphic designers
- Often no possibility of migrating to real application
- Examples:
  - Bricklin's demo (PC)
  - MacroMind Director (Macintosh)
  - Hypercard (Macintosh)
  - SILK (bridges to implementation)

Interface Builders

- Allow the designer to layout widgets to make windows
- Provide a pallete of widgets
- Select a widget, put it in a window
- Set widget properties
- Use actual widgets from a toolkit; so can be used to build parts of a real application
- Most generate code (C or C++); some generate a description of the interface in a language that can be read at run-time.
- Easy-to-use but limited
  - no support for drawing graphics inside window
  - no support for dynamically created menus/dialogs
- Examples:
  - Menulay (1983; research system)
  - NeXT interface builder
  - UIM/X (X)
  - BlueSky's WindowsMaker (Microsoft Windows)
  - VisualBasic
  - Most OO frameworks have interface builders

Application Frameworks

Toolkits are difficult for programmers to use:

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

Application frameworks are very popular now:

- usually object-oriented
- classes are provided for all of the various widgets
  - programmers specialize 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

User Interface Development Systems

- Provides comprehensive support for developing UIs
- May support the "insides of windows" for a particular domain (e.g., CAD, GIS)
- Often include application frameworks
- Often include interface builders
- Examples:
  - MacApp (Macintosh)
  - Amulet (CMU; C++)
  - Garnet (CMU; Lisp)
  - IDE / Rapid / USE (Suntools; X Windows)
  - PC frameworks: BorlandC++; VisualC++

Research Issues

- Need for new programming languages
- Increased depth
- Increased breadth
- End-user programming and customization
- Application and UI separation
- Tools for tools

Research Issues
Need for new programming languages

- most tools use libraries and interactive programs which are **separate** from the programming language
- techniques like OOP, multiple-processing, and constraints are best provided as **part** of the library
- even Java uses separate libraries, which makes it harder to create interfaces
- better idea: an integrated environment
  graphical parts specified graphically, and the rest textually
- research area: how to improve programming languages to better support user interface software

Research Issues
Increased Depth

- Most tools today help with the generation of the code of the interface
- Assume the fundamental design of the UI is complete
- Tools are needed to support the entire cycle:
  - task analysis, design, implementation, evaluation

Research Issues
Increased Breadth

- User interfaces of tomorrow will be different from today's WIMP GUIs
- Already we are seeing:
  - 3D visualizations and animations
  - gesture and handwriting (pen) techniques in PDAs
  - virtual reality systems
  - new media: sound and video
  - groupware
- New input devices will soon replace the conventional mouse and menu styles
- Research in new tools to support these new interaction technologies are needed

Research Issues
End User Programming Customization

- The spreadsheet is one of the most successful pieces of software
- Why? It allows end-user programming (e.g., macros, formulas)
- End-user programming is rare in other applications
- If it is there, it usually requires a programmer:
  - emacs: uses `e-lisp`
  - AutoCAD: uses `Lisp`
- More effective mechanisms, usable by end-users, are needed
- Should not be built into individual applications
- Instead be a part of the system level, so the user can just learn one language.
- Should not be a programming language like `C` but at a higher level.

Research Issues
Application UI and Separation

- Fundamental goal for UI tools: to allow better modularization and separation of user interface code from application code
- Conventional toolkits actually make this separation more difficult
  - due to large number of callbacks
- Further research needed to find ways to better modularize code, and how tools can support this

Research Issues
Tools for Tools

- Creating user interface tools
  - is very difficult
  - takes a lot of effort
- Research is needed to make the tools themselves easier to create
- Some promising work has begun:
  - Garnet supports the creation of highly graphical applications
  - Unidraw supports the creation of interface builders
- More work is needed ...