User Interface Design
& Development

Lecture 5
GUIs Technical Background

João Pedro Sousa
SWE 632
George Mason University

GUIs family tree

- 70's: first generation
  displayed text on the screen
- second generation
  displayed text plus vector graphics
  - based on geometrical primitives
    points, lines, curves, and polygons
  - heavy use of math: scientific applications
  - still in use today in high-end graphics, games, virtual reality…
- 90's: dominated by raster graphics aka bitmap
  - direct mapping of bits in memory to pixels on the screen
  - much simpler to program for

raster graphics
evolution

- the X windows library late '80s
  - public domain source
    a precursor to open source
  - very low level: a lot of work to develop an app
- widget libraries
  higher level functions built on top of low-level libraries
  - Motif (X), DEC Windows (X), HyperCard (Macs), MS Windows (PCs)
- last three decades were dominated by
  the workstation model and WIMP UIs
  Windows, Icons, Menus, Pointer
  - WIMP interfaces go back to Xerox Alto & Star: 1970s

today
overview of WIMP Elements

- widgets
  - menus
  - toolbars
  - dialog boxes
  - controls

Acknowledgment
some of the material presented throughout this course is adapted
from previous offerings of the same by Jeff Offutt

widgets are visual elements that facilitate communication with users

- four types of widgets
  - imperative : used to initiate a feature
  - selection : used to select options or data
  - entry : used to enter data
  - display : used to visually show and manipulate data
- a typical app uses only a small set of widgets
  - Windows and Mac use only a fraction of the invented widgets
  - HTML implements very few GUI widgets

additional reading on widgets:
Ray Eberts: *User Interface Design*, Prentice-Hall

GUI app structure on top of widget library

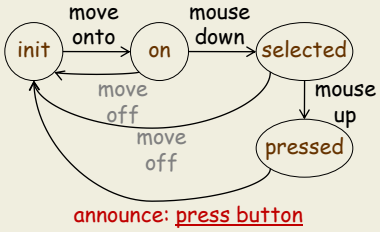
- initialize
  - include libraries
  - create widgets
    - register callback handlers for widget events
- layout widgets on screen
  account for
  - screen size & widget proportions
  - alignment
  - resizing properties
- start continuous loop
  - wait & process widget events
  - detect *exit* event

widget libraries generate events

- libraries run their own set of processing threads which constantly monitor for input events
  - mouse move
  - moving onto a widget
  - moving off of a widget
  - mouse button down
  - mouse button up
  - mouse click
  - double-click
  - press a button
- events at different levels of abstraction
  - up to the app writer to decide which events to register for

sequences of events have meaning

- app monitors sequences of these higher level events
- transition diagrams help understanding which are valid sequences to look for
  - events are edges
  - valid states are nodes
  - annotate states/edges with output/processing

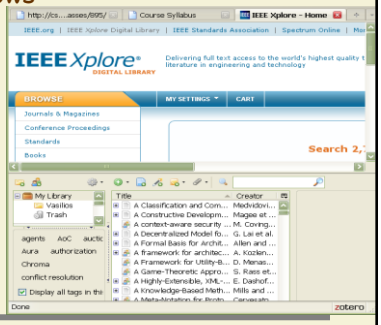


- the *button* widget monitors:
  move onto widget -> mouse button down -> mouse button up
  = press button

containers
aggregate other widgets

- add other widgets into a container aka parent widget
- removing the container removes all children

- form widget
  - layout helper to be included in windows
  - generates no events
- tabbed pane
- panned window
  - areas separated with a sash
    often draggable
  - use areas for diff purposes
    e.g., browse vs. bibliography



outline

- widgets
  - menus
  - toolbars
  - dialog boxes
  - controls

menu
guidelines aka conventional wisdom

- small number of menus 7±2
- list of options on each menu
  - tradeoff with number of menus
  - menu organization depends on use
    - more frequent items: easier to access
    - less frequent items: consider alternatives to menus
- group related options
  - in different pull down menus
  - using separators within same menu
  - using hierarchical menus
- menus are for actions
  not for selecting items such as *courses* or *US states*

menus
proven design decisions

- disable menu items that are not currently relevant
  - keep them visible but grayed out
- text vs. icons for menu items
  - icons are faster for experienced user
  - text is easier to recognize by novices
  - some menus show both, especially if also available in a toolbar
- support accelerators for frequent actions
  - Ctrl-X/C/V for cut/copy/paste
  - and mnemonics for easy keyboard access Alt-File-Save As...

menus
other design decisions

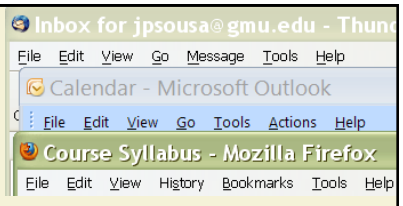
- allow options as menu items
  see *checkbox* and *radio box* later
  - checkmark/icon indicates selection state of option
- placement of *options*
  - under Tools? under Tools-Customize?
    some applications use both, e.g. PowerPoint 2003
- expanding menus e.g. MS Office 2003
  - show most/recently used items, then expand if user waits
  - may slow users down – certainly for infrequently used items
  - allow turning it off
- cascading/hierarchical menus
  - often designed inside-out: look at all the cool features
  - may defeat the purpose of menus: easy to find frequent actions

menus
stretching design decisions

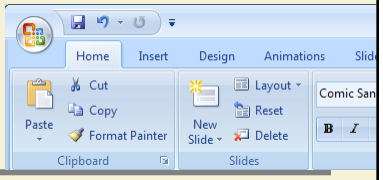
- bang menus
  annotate the menu label with a !, e.g. Print!
  - promote a command normally found inside a menu
    to a menu header
  - break the expectation of showing a passive list of options
    that the user inspects and may or may not select

using menus is supposed to be easy
designing them poorly is even easier

many current menus are legacy

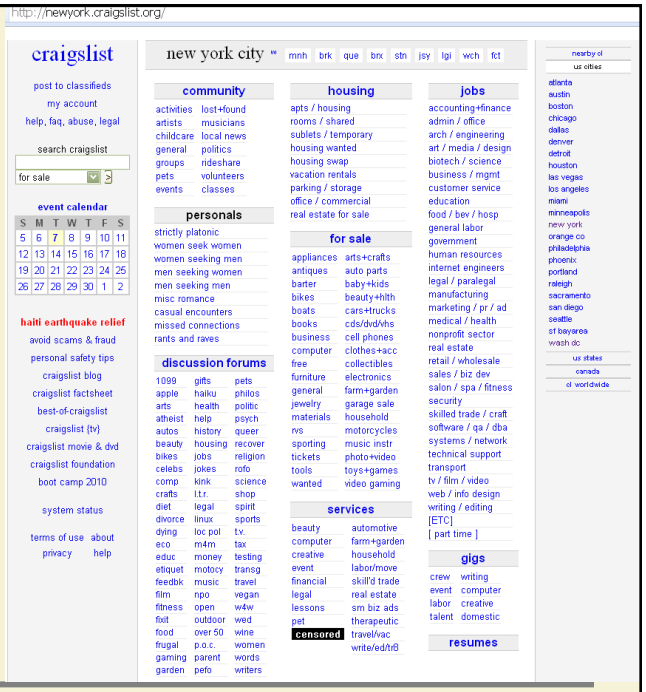


- having fairly standard organization promotes usability
  - users learn one organization, instead of one for each app
- have been around since the early 80's
  make *interesting* design choices:
  - File – Save/Print... is object-verb
  - Insert – Symbol/Picture... is verb-object
  - Edit – Cut/Select All/Object... anything goes
- challenge: how to improve design without
  - breaking off from standardization
  - confusing users



example

- defies conventional wisdom
- or does it?



example

- combine graphical representation
  www.alamo.com
- use visual highlight
  www.flickr.com

All time most popular tags

animals architecture art asia australia baby band barcelona beach berlin bike bird birds birthday black blackandwhite blue bw california canada canon car cat chicago china christmas church city clouds club color concert cute dance day de dog england europe fall family fashion festival film florida flower flowers food football france friends fun garden geotagged germany girl girls graffiti green halloween hawaii holiday house india iphone ireland island italia italy japan july kids la lake landscape light live london love macro me mexico model mountain mountains museum music nature new newyork newyorkcity night nikon nyc ocean old paris park party people photo photography photos portrait raw red river rock san sanfrancisco scotland sea seattle show sky snow spain spring street summer sun sunset taiwan texas thailand tokyo toronto tour travel tree trees trip uk urban usa vacation washington water wedding white winter yellow york zoo

menus for small devices

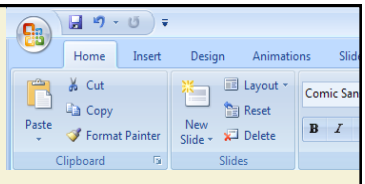
- often take full screen
- limited input device
  - touch
    - not always available/practical
    - coarser grain than mouse
  - no mouse
    - creative use of 2D pointer

Design is not just what it looks like and feels like.
Design is how it works
Steve Jobs

outline

- widgets
  - menus
  - toolbars
  - dialog boxes
  - controls

toolbars
are visual, always-on menus



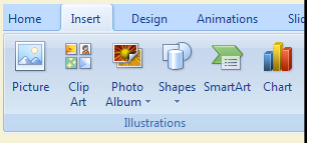
- fast access to frequent functions
  similar to shortcuts
- visual
  - icons are easier to remember than to learn
  - makes it harder for new users
  - icons may be complemented with a label and/or tooltips
  - …exploratory role of undo: more in later classes
- consequence
  - once users learn the toolbars
    they may resist accessing features on conventional menus
    why MS Office 2007 practically got rid of conventional menu layout

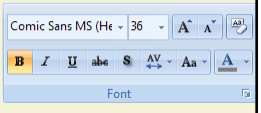
note
tooltips

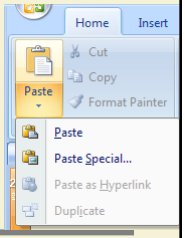
- an idea in the research literature in the '70s
- Apple tried balloon help
  - too big and intrusive for frequent users
- Windows improved by making them
  - smaller
    less intrusive
  - delayed activation
    monitor the event *hover* instead of *mouseover* aka *move onto*

toolbars
types of buttons on tool bars

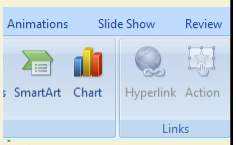
- action aka momentary
  - push it to start a feature
- option aka latching
  - selects an option *and* indicates the selection
  - better fit for options than conv. menus
    - why?
      always visible
- pop-up
  - similar to hierarchical menus
    see *combutcons* later







toolbars
guidelines



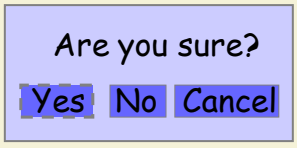
- inactive buttons
  - should not disappear
  - should not depress: confusion with option selection
  - should be grayed out
- toolbars should be customizable
  - move, dock & resize
  - add/remove buttons
  - change size of buttons: visual acuity of users
  - maybe even detach some buttons: magnetic buttons
  - MS Office *ribbon* does not allow any of that – why?

outline

- widgets
  - menus
  - toolbars
  - dialog boxes
  - controls

dialogue boxes are also legacy

- library designers wanted to facilitate app programming by offering pre-packaged dialogues for frequent cases
- built-in dialog boxes are tempting but
  - maybe not exactly what is needed
  - maybe confusing to users
    what is the difference between no and cancel?



dialogue boxes come in two flavors

- modal
  - no other interaction with the app until the dialog is closed
  - easy to program
  - easy for users to understand but intrusive
    - e.g. error messages should disappear with next action on app
- modeless
  - interactions with the app continue independently e.g. find
    - some interactions may be restricted
      e.g. modeless dialogue in MS Word disables text drag-n-drop
  - less intrusive than modal, but may be confusing
    - when do they disappear? why can't I do X?

dialogue boxes
the programmers' point of view

- dialog boxes serve a number of useful purposes
  - properties/options
    e.g. formatting & printing
  - function
    e.g. spell checking, finding
  - progress
    e.g. download, installation
  - feedback
    e.g. error messages, confirmation/hesitation

dialogue boxes
the users' point of view

- anything that appears on the middle of the screen
  and requires interaction, even if only for closing
  - is an interruption
  - unjustified interruptions are intrusive
- example of justified interruptions
  - the application cannot proceed without resolving the question
  - unrecoverable…
  - the operation is physically dangerous
- what seems justified to a programmer
  may be unimportant to a user
  - *know thy users*

dialogue boxes
the UI designer's point of view

- most of that a dialog box can do,
  can be done in another way
  - MS office apps use a dialog box for *find*
  - web browsers and Acrobat Reader include *find* in a toolbar

when you do decide to use one

- make modeless boxes visually different from modal ones
  - be consistent about modeless box termination
  - minimize impact of available functionality on main window

dialogue boxes
guidelines

- make dialogues movable
  - remember the position the next time it opens
- reduce intrusion
  - if possible include checkbox: *always do this from now on*
- make it clear for the user
  what will happen when a choice is made
  - yes/no may be not as clear as yes/cancel
  - what does closing the dialog do? consider disabling it
  - include a *help* button pref. not adjacent to *cancel*
- formulate questions positively
  - negative example: Do you want to *abort* the installation?
    - *cancel* becomes a double negative

take 5

outline

- widgets
  - menus
  - toolbars
  - dialog boxes
  - controls

control widgets
come in many flavors

Imperative
1. Push Button
2. Butcons / toolbar

Selection
3. Checkbox
4. Latching buttons
5. Radio box
6. Combutcons
7. List
8. Combo box
9. Tree

Entry
10. Bound Value
11. Spinner
12. Text

Display
13. Label
14. Scroll bar
15. Sliders
16. Thumbwheels
17. Splitters
18. Drawers

imperative controls
1. push button

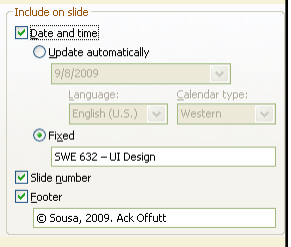
- used to activate a particular action
- often offers no feedback
- the action may be undoable
  - do not use in dangerous situations without a confirmation aka hesitation

properties:
label, icon
parent
callback

imperative controls
2. butcons *button + icon* aka toolbar button

- toolbars used to group butcons
  - can be thought of as a "menu of buttons"
- butcons frequently
  - are square
  - show an icon but no text
  - include a tooltip to help explain the purpose

selection controls
3. checkbox

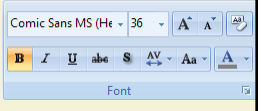


- support binary choices yes/no
  - may also indicate a state

4. latching butcons

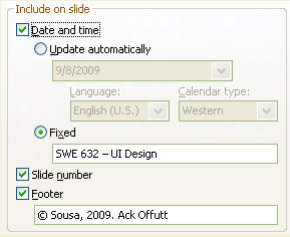
combine butcons with checkboxes

- e.g. selecting **bold**, *italics*, underline each a binary choice in MS Office products



selection controls
5. radio box

- offers a set of mutually exclusive options
  - collection of checkboxes with single selection
  - guideline: less than 8 options
- uses :
  - set some state in system
  - set options for customization
- originally diamonds, MS changed to circles



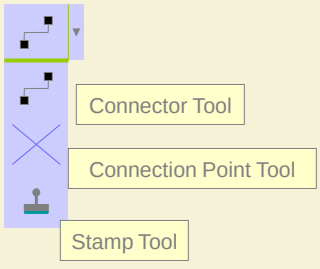
- very fast, low errors, but uses a lot of screen space
  - variant: radio butcon

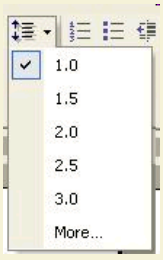
quiz
pull-down menus vs. radio box

- are they semantically the same?
  - menu options perform actions
  - radio buttons set options/formatting and indicate state
- use of space?
  - menus show on demand / may fluctuate
  - radio boxes are fixed within a window/dialog
- other differences?
  - menu options may be disabled based on app state
  - if the radio box is shown, all buttons are available

selection controls
6. combutcons

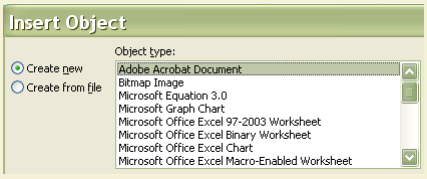
- a button that opens a menu of latching butcons
- a variant on drop-down menus
- examples:





selection controls
7. list aka list box, picklist

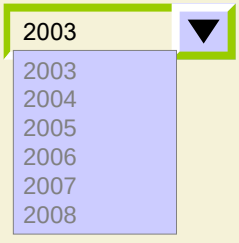
- use to
  - choose items
  - reorder items
  - drag & drop to/from lists



- may show text, icons or both
- may support multiple selection aka earmarking
  - by adding checkboxes
  - by shift/ctrl click
- guideline
  - limit number on entries -> vertical scrolling
  - avoid horizontal scrolling

selection controls
8. combobox

- combines a text edit field, a button and a list
- users can
  - press the button and select from the list
  - type, in two flavors
    - only choices from the list
    - freely
- good usability
  - fast
  - flexibility
  - low errors
  - easy to learn/remember

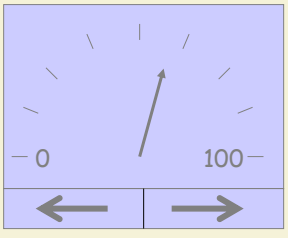


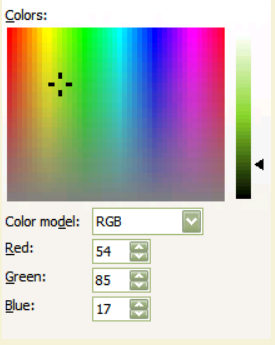
selection controls
9. tree

- shows hierarchical data
- usually shown sideways and often with icons
- example: mail folders

entry controls
10. bound value – scale

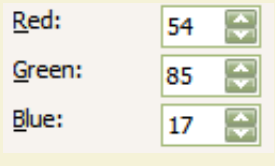
- use to select a value from a large range
  - allows only valid inputs
- may combined with text fields for flexibility
  - apply data immunity





entry controls
11. spinner

- use to select a precise value from a large range
  e.g.
  - age
  - day of month
  - RGB value
- can be combined with text fields for flexibility
  - text field may or may not allow values out of range



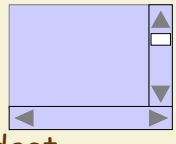
entry controls
12. text box

- use to allow flexible data entry
  - must be validated for errors
    - active: ignore invalid keystrokes
    - passive: check at the end, e.g. when user presses OK
  - if contents are interpreted
    may present security vulnerabilities
- slower to use than select from list, combobox...
- often supports operations: *select, copy, cut, paste*

display controls
13. text label

- use for simple disambiguation
- usually combined with other widgets
- no events

14. scroll bar



- use when contents are too long to fit in widget
- hard to use
  requires coordinating
  - fine motor control: holding a button on a small icon
  - large motor control: moving your arm
- consider expanding the container widget

choosing widgets for a GUI

- don't just use the usual suspects
  - text field
  - combobox
  - menu
- choose the best widgets for each job

think outside-in

designing UIs keep it simple

- anyone can make something confusing
- it takes hard work, knowledge, and skills to make things simple

"It takes three weeks to prepare a good ad lib speech."
Mark Twain