Top-Down Design and UI Design

Continuing design
Issues to consider for implementation

Option: Two examples from last week again
Our Specifications

- Chatterbox
- CIME
- File Explorer
- Net Monopoly
- SimCasino
- Virtual Market
Top-Down Design

Group functionality into small number of major subsystems

Goal: Minimal but sufficient interface between components
- Sufficient: Must get job done
- Minimal: More inter-component communication leads to harder debugging
- Be lazy too... Don’t repeat work done in other components

Remember package diagrams?
- Groups of related classes
- Top level of your design wants a small number of package diagrams

This applies to UI design too
The Key: Consistency

Macintosh widely known for usability

Mac apps are usable because all Mac apps work the same

If app can do $x$, user knows where $x$ is

Don’t be inventive when deciding where things go in the UI

Do be inventive in deciding *what* it can do, not *where* the control is

A principle that applies to any platform
If Consistency is Key, Copying is the Keychain

Make your UI like other good UIs
- Not icons or splash screens
- Copyrighted artwork, duh — make your own!

Organization
- What menu items are where?
- What controls are grouped in which panel?

Rethink your app from user’s POV
- N.b., usually your users are not programmers

Start from the top and work your way down
- Good software engineering
- Think about entire application and group functionality by user task
Example: Top-Down: Menu First

Typical Mac menu bar is highly organized

“Size” of actions decreases from left to right

- Left: Apple menu — entire system
- Next: Application — entire application
- Next: File — entire document
- Next: Edit — section of document
- Right: Window — nothing to do with doc
- Far right: Help — No effect on app (but easy to find)
A Concrete User Interface

One document, one window

Controls affecting that document are part of that window
  • Psychological notion of affordances
  • Like an appliance: it has buttons that control it
  • Doesn’t mean your app looks like an appliance

Separate windows don’t affect each other

Has implications for how controls are grouped....
Modal Dialogs or Sheets

Document-modal dialogs
- Must complete this action before doing anything else with this document
- Other documents are unaffected

On Mac, visually attached to specific document
- Enforces one window—one document

Save/Save as file names and file formats
Toolbars

On Mac, part of the window (not floating)
User-customizable (using a sheet)
Any scope of action, up to and including entire document
  • Since user can customize, anything goes
  • But you should make a good default set (restoring default set is part of the customization sheet)
  • Not the place for application-wide controls
  • Not the place for window controls like close and maximize (those are already on the title bar)
Drawers

Attached to a window; Non-modal

Can serve as a hide-able toolbar
  • More room than a toolbar
  • But doesn’t need to be there

Toolbar button to show/hide

The right place for tool options
  • Instead of common floating panel
  • Can have more than one per window (though usually only one is shown at once)
  • Two windows open means two drawers means see options on two tools simultaneously
Panels

Non-modal
Not attached to a document
Good for application preferences

Complex selections for a single control
  • Color
  • Font
  • Arguably, this breaks the visual language, though (which window “owns” the color panel?)
A Structured Document Editor

Create and edit structured documents in a “WYSIWYG” fashion

Use standard file formats: XML et al.

Make it as easy to use as a word processor

While producing clean, flexible mark-up

Does our package breakdown still work when considering UI?
Editing: Use Cases

Process: Acquisitions editor structures document, writer creates it, editor edits it, production staff prints/publishes

Writer
- Chooses template
- Writes prose

Acquisitions editor
- Creates templates
- Identifies production formats (web, print, PDA, etc.)
- Identifies components (TOC, index, etc.)

Production staff
- Designs style sheets for production formats
Edited: Packages

- Obvious breakdown: XML acronyms

Templates: Document Type Descriptions (DTDs)
Writing: XML document with DTD specified
Style sheets: CSS
Automated content: XSLT

UI: Little in common across acronyms
- Beyond usual file, edit
- Suggests package breakdown works for UI, too
But Where’s MVC? Or UI/App Split?

UI toolkit part of platform; don’t build your own

- Even undo/redo, scripting, cut-copy-paste

Major functionality is all interactive

Split off interaction and you cut across entire application: Not a narrow API

Examine API between packages described

- DTD to XML editing: Permitted elements
- XML to CSS: Element and context (path in tree)
- XML to XSLT: Pipeline from one XML doc to another
- Some don’t need to talk much (a good sign): CSS to XSLT
Another: Media Asset Management

Content production group making use of stock or existing media
Find artwork; Add new artwork to collection
Track royalties, credits, rights, versions
Support full MIME range of media
High quality, possibly off-line

UI-intensive application
MAM Use Cases

Archivist maintains media quality
Producer obtains and uses content
Artist creates new content and archives it
Distributor delivers content to audience
Rights holder gets paid

Each role has substantially different tasks
  • Implies very different user interface
MAM Packages

Indexing, searching, and summary
  • Needs asset description for attributes
  • Needs location for content-aware searches
  • May need rights if rights are criteria in search

Asset location tracking
  • Online, offline?
  • Multiple versions? Levels of quality?

Asset description
  • Usual list of attributes: name, date, serial number
  • Computed data: index or cross-link info

Rights and usage control
  • A whole can of worms all on its own
  • Permissions, payments, contracts, etc.
MAM Package Comments

Single role uses one or two packages at a time
- Producer will use index to find assets
- Artist will mostly use description
- Administrators use rights management to make payments
- Key events may invoke three at once: Publishing new asset specifies location, description, and rights

Little inter-package communication
- Obvious (and rational) exception: Indexing
- Description doesn’t need location or rights mgmt
- Location doesn’t need (much) description or rights mgmt
- Rights mgmt almost completely isolated

Package breakdown working for UI, too
The Next Step Down: What’s In a Package?

Class hierarchies
- Generally isolated inside a package
- Exception: Interface or abstract base class exposed as API

Data flow, State transition diagrams
- When getting something from outside package, what do you do with it?
- Trading things back and forth across package boundaries not likely; In general, send and receive only

UI designs
- Correspond strongly (exactly?) with class diagrams
- Methods for UI buttons, actions, etc.
For Next Week

And every week for the rest of the semester

Milestone report
• In writing
• As long as you think it needs to be
• For someone else to read it
• Containing whatever you think is important about what you accomplished in the past week

In-class presentation
• 5 minutes
• Open to questions from class
• Recommend PDF or HTML pages online
• Feedback to improve your project