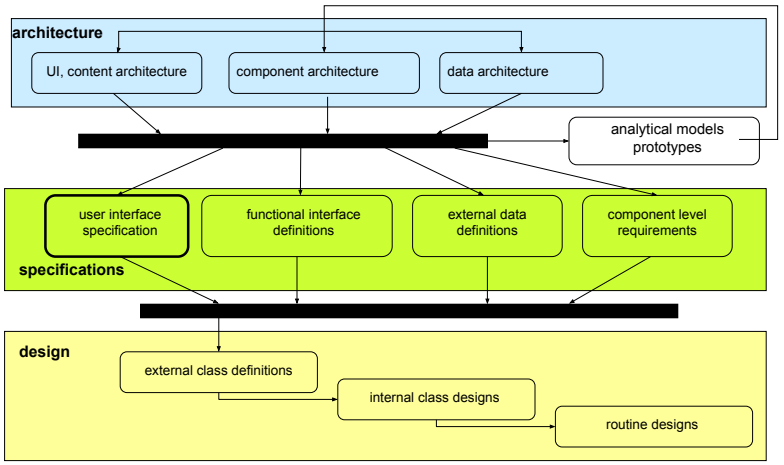
User Interface Design

- problems of User Interface design
- elements of good User Interface design
- designing U/Is and content structure
- U/I design exercise
- supplementary subjects
  - web vs. phone U/Is
  - choice of presentation/navigation metaphor
  - reviewing and testing U/Is
  - CLIs

Model Hierarchy/Succession



Why User Interfaces are critical

- sit between the user and the software
  - good UI enables users to exploit the s/w
  - bad UI prevents users from using s/w
- most users do not receive formal training
  - UI must be obvious and/or self-teaching
- technical support is difficult to get
  - UI must prevent/diagnose/fix most problems
- UI can make or break a product
  - largest single element of the user experience

User Interfaces aren't easy

- our s/w is growing ever more complex
  - encompassing ever more tasks & options
  - working in ever more complex environments
  - integrating with ever more applications
- the users are not homogeneous
  - they have different needs and goals
  - they have different technical depth
  - they have different backgrounds
- the designers may not be like the users
  - have different skills and experiences
  - may have different goals for program function

It takes a village to build a UI

- Technical Knowledge/Skills
  - familiarity with the product design
  - familiarity with the chosen UI toolkit
- Domain Knowledge/Skills
  - familiarity with the various classes of users
  - familiarity with the tasks being automated
- Human Factors skills
  - techniques of complexity assessment
  - techniques of information organization
- plus artistic and writing skills

Elements of good U/I design

- Familiar and Consistent
  - contexts, objects, actions
  - icons, positions, style
  - metaphors, nouns, navigation, "grammar"
- Intuitive and Understandable
  - current context (objects & options) is clear
  - offers context/history-appropriate options
  - clear how to perform all common operations
  - meaning of presented information is obvious

Understandable Displays

- organization
  - group related functions (content, navigation)
  - consistent positioning of all elements
- presentation
  - most of the space reserved for content
  - use white space to separate display elements
  - avoid putting to much info in one display
  - use multiple windows where this makes sense
- usability
  - consider smaller displays, slower links
  - don't over-use scroll-bars

Elements of good U/I design

- Simple and Convenient
  - doesn't expect user to remember much
  - don't overwhelm with information or options
  - anticipates needs (likely actions, default values)
  - but … does not force user down a path
- Communicative and Responsive
  - current context, state and options are clear
  - status of in-progress operations is clear
  - completion and status of recent operations

Elements of good U/I design

- Helpful and Robust
  - defaults and option menus for user input
  - thorough input and request validation
  - error response is meaningful and helpful
- Adaptable and Configurable
  - offers different user roles different views
  - offers multiple (e.g. novice/expert) modes
  - configurable default context, options, views
  - language, locale, and accessibility options

Levels of User Interface design

- interaction and content models describe
  - sequence of screens for each task
  - general contents of each screen
- navigation design describes
  - how user will move through defined screens
    - general structure, and then widget types
- detailed widget behavior
  - is usually very well defined by UI toolkits
  - defining new UI widget behavior is dangerous

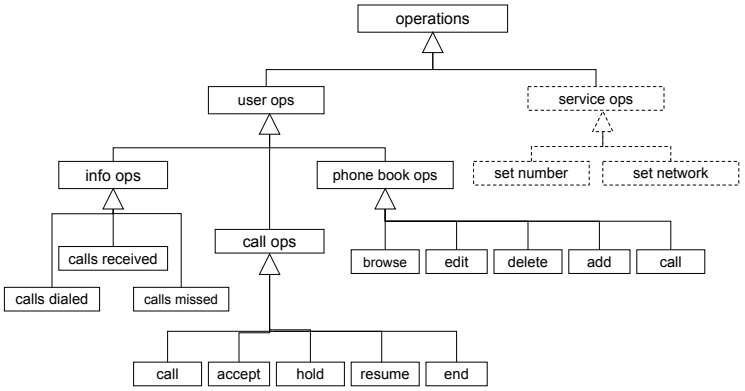
Developing a User Interface

1. Identify the scope of the user interface
   - user types, use cases, domain objects
   - develop detailed task descriptions
2. Structure the user interface
   - identify the user visible states and transitions
   - resolve the task steps into screens
3. Structure the content
   - types of content to be presented
   - how to rationally structure and present them
4. Detailed design of each screen
   - specify contents, control, navigation options

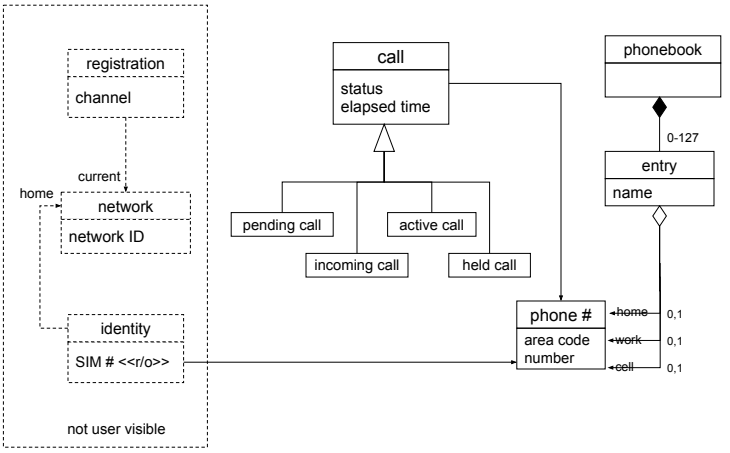
1. Identify scope of User Interface

- identify the types of users for the S/W
  - characterize each by needs and experience
  - refine a set of distinct user-roles
- identify tasks performed by each role
  - use case scenarios for each task
- develop detailed task descriptions
  - class diagrams to describe domain objects
  - activity/state diagrams to describe task steps
  - list information users will provide and want
- elaborate and validate these descriptions

Cell Phone Operations



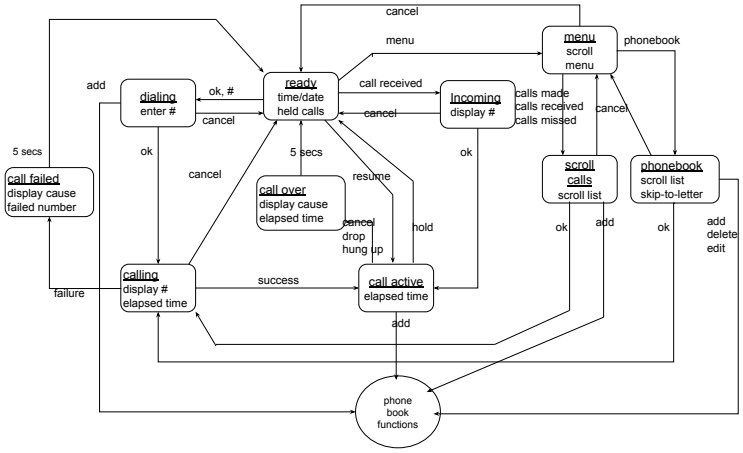
Cell Phone Object Classes



2. Structuring the User Interface

- break tasks down into screens
  - where information is presented/gathered
  - changes result from user or external events
- map all the screens with a state diagram
  - name each screen
  - summarize information to be presented
  - summarize information to be entered
  - show all possible transitions to other screens
- refine, review, and validate this model

Cell Call State Model



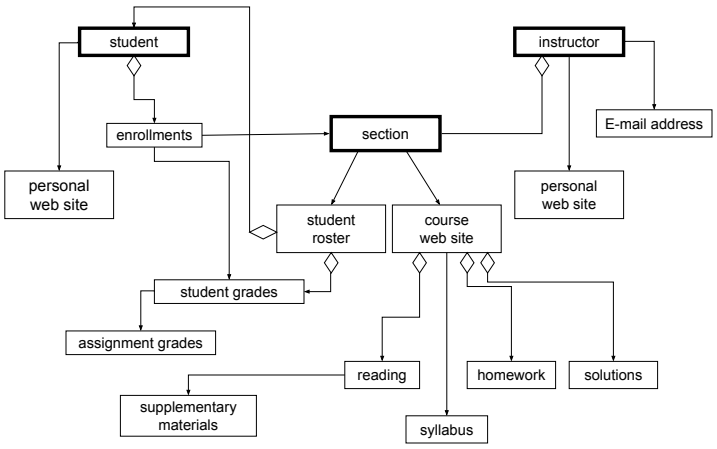
3a. Content Model - Scope

- enumerate types/sources of all content
- information directly associated with tasks
  - information displayed in course of tasks
  - information entered in course of tasks
- information browsable through application
  - task domain object attributes and history
  - attributes and history of related objects
  - help information
  - related information links

3b. Structuring the Data

- identify entry points into data hierarchy
  - students, sections, professors
- identify all containment associations
  - course description contains a roster
- identify <u>all</u> relevant associations
  - section directly refers to an instructor
  - reading assgts include supplementary URLs
  - student can get from section to grades

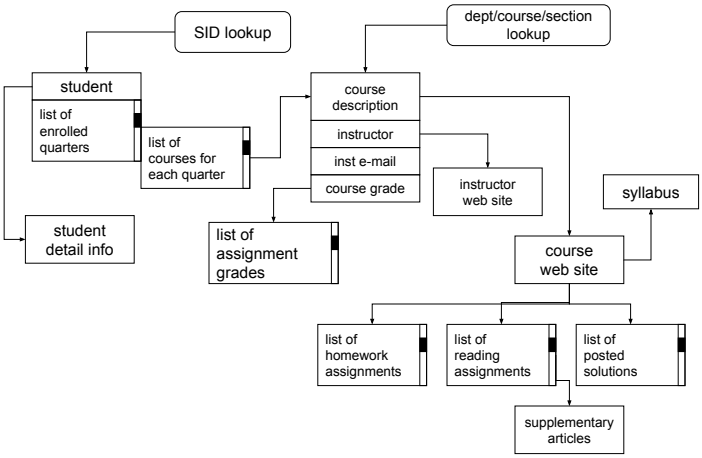
map natural structure of data



3c. Define User Visible Structure

- **UI views need not be the domain classes**
- one UI view can include many classes
  - bring referenced objects into the container
  - hiding irrelevant classes and associations
- one class can break into multiple UI-views
  - create new (useful) relationships
    - summary and detail views
    - search and browse views
  - view data in context of different relationships

UI views



3d. Specify Content Navigation

- define means to follow each UI view arrow
- many choices may be obvious
  - from a name/icon to the corresponding object
  - from a summary field to the supporting detail
  - object selection from a list (incl. scrolling)
  - previous/next object in current succession
- some navigation may not be obvious
  - different view of the same object
  - new high level object search

Team U/I design exercise

- Consider: your application
  - key user tasks/goals and content elements
  - natural task-based navigation paths
  - natural content structure/relationships
  - relevant information on each path
- pick an area where some of these may be vague/complicated
  - find ways to make this clearer/simpler
- be prepared to discuss your results

Mid-Term Exam (Tu 10/12)

- closed book (no notes), 70 minutes
  - I will be available to explain questions
- ten multi-part questions
  - all based on key learning objectives
    - most focusing on concepts and issues
  - all call for brief answers
  - most parts more difficult than quiz questions
    - but have been answered in reading/lectures
- one extra credit question
  - practical application, not discussed in class

For Next Lecture

- McConnell: 6 Classes
- McConnell: 34.4 Language Limitations
- Object Mentor: Packaging Principles
- Spolsky: What are Specs?
- UML: Class, Package, Object Diagrams

Supplementary Slides

Graphic User Interfaces

- There are competing GUI toolkits
  - Windows, Motif, MAC, Java UI classes, etc.
  - they offer different widgets
    - with different appearances and behavior
  - they offer different programming models
  - these are confusing for users and developers
- When building a native GUI
  - you must choose a toolkit to use
  - may affect choice of navigation metaphors
  - will definitely affect design of components

Web Application UIs

- HTML browsers are more standardized
  - links, forms, frames, pop-ups, scrollable text boxes, style-sheets, multi-media content, etc.
  - powerful extensions (DHTML, Java, J-script)
  - provide interfaces for local and remote users
- some GUIs are becoming WEB front-ends
  - providing improved standardization
- but this is not yet a panacea
  - Java applets can still create own widgets
  - high performance apps still use direct screen

Standard GUI Metaphors

- Information
  - forms w/ input fields
  - tables of information
  - successive pages
  - scrollable displays
  - wizard dialogs
  - pop-up windows
  - (cursor) tool tips
- Controls
  - menu/action bars
  - control button icons
  - object icons
  - drag-n-drop
  - right-click
- Navigation Aids
  - explicit links
  - tabs
  - site maps
  - search windows

choosing application metaphors

- many well known navigation techniques
  - each has advantages and adherents
  - none is intrinsically superior to another
- the best choice is the most familiar
  - other applications used by same customers
  - other applications in same product family
- this decision may be forced
  - by user interface toolkit
  - by corporate style guidelines

Reviewing U/I Functionality

- Conformance with Requirements
  - supports all specified tasks and options
- Ease of Use
  - how many screens for common scenarios
  - how many cursor motions and key strokes
  - how much information remembered/supplied
  - how it responds to the most likely errors
- Consistency
  - of navigation, metaphor, operation grammar

Reviewing U/I Appearance

- General
  - Consistent positions and representations?
  - All content displays are self-identifying?
  - Distinct elements are readily distinguishable?
  - Primary navigation options are obvious?
- Large report displays
  - Easily viewable in subsets?
  - Ease of navigation to desired subset?
  - How does it scale to smaller windows?

UI Evaluation - Usability Testing

- informal usability testing
  - a few customers play with a prototype
  - they write a report on how they liked it
  - developers may be present during testing
- formal usability testing
  - performed in a controlled usability testing lab
  - users are given a scenario to perform alone
  - developers are not present during testing
  - the session is video recorded
  - usability analysts produce formal report

Command Line Interfaces

- two kinds of users
  - power users who know just what they want
  - automated scripts (shell, Perl, CGI, make, …)
- both want to avoid program interaction
  - all parameters are specified up-front
  - input data from specified files or stdin
  - results to specified files or stdout
  - diagnostics go to stderr
- this completely changes design goals
  - but many of the principles remain the same

Command Line Interface goals

- Power
  - all options can be set from command line
- Brevity
  - short specification strings (reduce typing)
    - e.g. “-l” vs “-view=long”
  - good defaults (eliminate need for args)
    - reasonable built-in defaults
    - system and per-user configuration files
    - key parameters set by environment variables
      - e.g. search paths, location of configuration file

Command Line Interface goals

- Cohesion - program does only one thing
  - a few basic get/set/transform functions
    - may even be separate commands to get and set
  - all related to one set of objects or functions
- Familiarity and understandability
  - standard/mnemonic arguments
    - e.g. **-l** for long, **-v** for verbose
  - consistent argument syntax rules
    - e.g. optarg conventions
  - unrecognized options give usage message

Typical CLI conventions

- single letter options
  - turn specific options on or off
  - can be specified separately or in groups
    e.g. "**-l -v -r**" or "**-lvr**"
- options with arguments
  - specify value for an input parameter
    e.g. **-I /usr/include/midnight**
- environment variables set context
    e.g. **LIBPATH="/usr/lib:/usr/ucb/lib:/usr/local/lib"**
    LOCALE=ENG_US

Pipe-line-ability

- The toolbox concept
  - build transformations out of standard tools
  - one program's output is another's input
- many handy stream processing tools
  - grep, sort, cut, tr, awk, etc.
- functions that work with lists of files
  - ls, find, test, tar, mail, print, more, etc.
- make your output suitable as input
  - one line per record, tab separated fields, etc.