More User Interface

Larry Rudolph
Review

- Examples of handheld interface
  - device is with you & knows location ==>
    it knows your location
  - can “improve” location information
- Device can interface with other devices
  - e.g. those nearby via bluetooth & those in
    the world via internet & that it knows
    and trusts
Review II

• Handhelds have functionality of computers
  • but with more constraints
  • exploit computation to compensate for shortcomings -- main message of course

• Small screens & keyboards
  • Probably also small microphones, speakers, and other I/O -- discuss implications for UI
Review II.a

• Keyboard solution
  • external ones
  • T-9 & Predictive text
  • Dasher
    • predictive text (model of user input)
    • graphics actually matter a lot; box not letter that change size; (other graphics?)
Manipulation

- How does user interact to make things happen?
- Direct vs Indirect -- what does this mean?
  - Drag & Drop, Click, Keyboard command
- Rule: Do something when an event happens
- Pervasive theme: direct vs indirect
  - Examples: turn on computer; word vs latex
- Many other examples
Specifying UI View

- Direct: in code
  - within program execution
  - separate resource & specialized language
    - what are the pro’s and con’s
- Indirect: implicitly specify via something
Symbian Views

• (Not a python thing)

• Each application has 5 components, one is view
  
  • Do not want text strings with code if app is for international audience. So use ptr?
  
  • Use whole view. Different view package for different locations / languages / screen
  
  • Make view available to other applications
Implicate Method

- Structured domain, automatically generate view
- If all menu handlers are in same object (class), can use introspection to generate menu items. Done at run-time.
def refresh(self):
    app.title = u"Larry's First App"
    app.menu = [
        (u"add to set", self.add_handler),
        (u"new object", self.new_handler),
        (u"change mode", self.mode_handler),
        (u"Cut", self.cut_handler),
        (u"Paste", self.paste_handler),
    ]

    def mode_handler(self):
        self.currentMode = ModeList[ popup_menu(List) ]
        self.display()

    def cut_handler(self):
        index = self.lb.current()
        cf = self.rawentries[index]
        self.clipboard = cf
        self.clipboard = cf
        self.display()
from install_menu import *

class test_menu:
    def __init__(self):
        self.a = initial values
    def menu_zero(self):
        print "got a callback to zero"
    def menu_one(self):
        print "got a callback to one"
    def menu_two(self):
        print "got a callback to two"

m = install_menu( test_menu() )
def install_menu(c):
    """given a pointer to a class, add all class methods
    whose names begin with 'menu_' to the menu."""

    menu_items = []
    for n in dir(c):
        if n.find('menu_') == 0:
            s = eval('c.'+n)
            menu_items.append( ( unicode(tag[5:]) , s ) )
        if n.find('exit_') == 0:  exit_key = eval('c.'+n)

    appuifw.app.menu = menu_items
    appuifw.app.exit_key_handler = exit_key
Handlers everywhere

- We have constraints on where are handlers
- In same class, easy
- In other classes, how to reference them?
  - make class instances global?
  - pass methods into menu builder?
  - what do you think?
Indirection to the rescue

class EventPublisher:
    
    utility class to provide basic Publish/Subscribe functionality.

    def __init__(self):
        self.__published = {}
        self.notify = e32.ao_callgate(self.__notify)

    def subscribe(self, event, callback):
        
        Subscribes a callback function to specified event. There are no
        timing restrictions on callback. Event must have been published.
        
        if event not in self.__published:
            raise ValueError("no such event %s" % str(event))
        if not callable(callback):
            raise ValueError("callback must be callable")
        self.__published[ event ].append(callback)
def publish(self, event):
    
    publishes an event, so that subscribers can subscribe.
    
    if event in self.__published:
        raise ValueError("already publishing %s" % event)
    self.__published[ event ] = []
def __notify(self, event, *args):
    # notifies the event subscribers that an event has occurred. Schedules each subscribed callback function
to be invoked with the specified args. Does not actually invoke them, to ensure that a call to this
method returns promptly and without blocking. All callbacks will be invoked in the context of the thread that created
this object. 

    funclist = self.__published[event]
    if len(funclist) > 0:
        dbg("util", "%s - callbacks to notify: %d" % (event, len(funclist)))
        for cb in funclist:
            def callback( cb=cb, args=args):
                try:
                    cb( *args )
                except Exception, e:
                    dbg("util", "uncaught exception in callback!")
                    dbg_exc("util")
                    e32.aosleep(0, callback)
• Different screen resolutions
  • e.g. 176x208; 240x320 (quarter vga); 352x416
• Different screen orientations
Navigation

• On each screen, user should be able to answer questions:
  • Where am I?
  • Where can I go from here?
User Interface

- Application needs external interaction.
  - Disconnect application from user interface
    - UI could be: character display & keyboard
    - Or: file, tape, CD, ...

Graphical user interface

- Must make many assumptions about h/w
  - Code usually written to lowest common denominator
  - Why write different code for each fancy feature
- Code gets tied to GUI
- Abstraction advantage is minimized
  - hard to adapt to different interfaces
  - e.g. try to access web page via dumb terminal (lynx)
Signals and Actions

User Interface

- output: data & signals (event == signal)
  - data is usually text to display to user
  - cause something to happen e.g. popup window
- input: data & signal
  - data is usually text that the user types
  - event is other user action, e.g. enter key, mouse click, mouse movement

Two types of signals (events)

- window manager, e.g. kill window, resize
- application UI, e.g. menu selection

Signals invoke application code

- “call back” or “handler”
Evolution of GUI

Evolution

- application code does everything
- libraries provide uniformity
  - apps examine all events & pass on irrelevant ones
- multi-tasking pushes routing down to OS level
- window manager (pull routing out of OS)
- multi-threading: data & signals passed between app and gui via window manager
- full separation: xml, soap, ...
Graphical Toolkit

Need something to draw graphics
Need something to manage windows, menus, buttons, icons, etc.
Need a “Toolkit”

- There are lots of them; some language specific
- We use GTK 2 (pygtk2 python interface)

http://www.moeraki.com/pygtkreference/pygtk2reference/
Graphical User Interface

We will use Glade 2: can see what gui will look like (but maybe not on ipaq small screen)

Glade is a program designed to enable the quick building of graphical user interfaces for GTK+ and Gnome applications.

http://glade.gnome.org/
Glade Screen Shot
GUI $\leftrightarrow$ App

- Application registers call-backs with GUI
- Signals or events invoke the call-backs
- Who ensures that the right connection?
  - It is all in the programmer’s head
    - (generally not a good idea)
- How much does the GUI know about the semantics of the input?
  - Not very much
- Want abstraction and autogen or checker
  - still open problem
Better Specification of UI

- Want universal user interface description
- Application can interact with several UI’s
  - keyboard, mouse, touch screen
  - speech, gesture
  - agent (a real PDA)
  - brainwaves, ESP
- Answer: still open research question
What can be done?

- Add semantic knowledge
  - Part of UI specification
  - Limit acceptable input

- User interface can pass on several alternatives to application
  - Top “N” choices with associated probability
  - Application can handle probabilistic choice of input