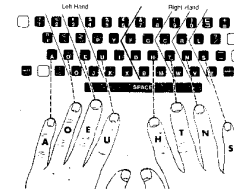




# GUI's and Keyboards

Larry Rudolph  
March 15, 2006



MIT  
Massachusetts  
Institute of  
Technology

# User Interface Goal

- **Convey and gather information from user(s)**
  - when user is not sitting in front of a PC
- **Support a set of standard actions and outputs**
- **Graphical User Interfaces (GUI) have been well studied**
  - Must understand them before generalizing
    - *Pervasive computing uses many types of user interfaces*
- **A web page is an example of a GUI**
  - Why is there a need for anything else?
  - Because of historical and efficiency reasons
  - Want more direct and richer variety of interface



What do User Interfaces have to do with pervasive, mobile computing? Nearly everything. The stuff underneath the covers is fairly standard computing, from networking, coding, databases, etc. But how the user or users interact with the system is what is different.

The user is not standing in front of a computer screen with a mouse and keyboard. The user may be walking, standing, driving, sitting in class, or even sleeping. But where do we start?

The best way is to go back to basics and review what we know and generalize from there. We know about Graphical User Interfaces. Well, we know something about them after decades of use. They are still evolving, especially as computer hardware resources continue to improve. Some of the GUI lessons are being applied to handheld devices, but they mostly are concerned with moving previous solutions to these new platforms.

# User Interface Goal II

**"As interfaces become easier to use, they become harder to create" [Meyers 1994]**

- **Do you agree?**
  - KISS: Keep It Simple
  - It takes a lot of hard work to make things look simple
- **What user interfaces do you like?**
  - iPod, Tivo, ...
- **Microsoft Windows 2000 have dynamic pulldown menus**
  - Does anyone like them? why not?
- **The new Microsoft office menu's -- do they help?**



# KISS: Keep it simple

- **Goal is for user to not think about how to do something; it should be automatic:**
  - If there is a choice, then one must think
    - *movie theaters offer very limited choice of candy*
    - *supermarkets offer huge variety of soft drinks -- what is difference?*
  - Save file via menu, keyboard, icon, rightclick, ...
  - One mouse button simpler than three
  - Pay a lot of money for large screen, why waste it on rarely used menu bar



# KISS: Keep it simple

- **People do not think about repetitive actions**
  - “Do you really want to delete file?”
    - *after third time, people usually click OK without thinking*
- **Lots of research on design principles**
  - and it is often ignored :(



# Manipulation

## ● Indirect Manipulation

- E.g. program places graphic objects on screen
- Nothing is “clickable”
- Today it feels awkward to use keys on mobile phone to manipulate objects on screen.
  - *Everything used to be that way.*

## ● Direct Manipulation of Objects

- User directly manipulates Graphical Objects with mouse or keyboard
- “Tangible User Interface” of the future, users will manipulate physical objects with their hands





# Widgets

- For lack of natural term, GUI objects are **Widgets**
- Everything in a GUI is a widget
- Widget is picture displayed on screen that is under control of the GUI.
- Widgets are hierarchical: they contain other widgets.  
Examples are:
  - Window frame, radio button, scroll bar, menu, ..
- GUI's support certain types of pictures
  - others types must be converted







# Big Idea: Abstraction



(c) 2003 Incognito Games Ltd.

- “Any problem in computer programming can be solved by adding a level of abstraction”
- “Performance can be improved in programs by removing a level of abstraction”
- Indirection used to support abstraction



Abstraction is a big idea in computer science. It is used everywhere. It is even more important in pervasive computing where one wants to replace traditional interfaces with ones more suitable for mobile and other nonstandard interaction situations.

# Non-widgets

- **A GUI is a level of abstraction between user and program, hence it affects performance**
- **Graphical Computer Games demand high performance for realistic animations**
  - They directly manipulate the screen, mouse, keyboard
- **Audio not part of GUI**
  - Duh. OK, but not part of window system either
  - Could do it by assigning a “channel” to each application and have user select the channel as in a radio



# Mice

- **Mouse is clicked inside of window or widget**
  - e.g.: controls standard widgets: drag a slider, twist a dial, scroll up or down -- all ways to enter a number
- **Major innovation: mouse cursor changes sprite as moves between widgets**
  - Real world analogy: frog looks different on a lily pad and in the pond, car looks same in drive- & park-way
- **Mouse is part of GUI, but not multiple mice**
  - New need arising from wireless mouse and from pda or cell phone as "mouse"
  - Perhaps we need different sprites per user?



# Mice II

- **Mouse actions:**

- Click is two actions: “button down” and “button up”
- Movement is relative: “delta x, delta y” events
- Drag: move while button is down
- Wheel: “Button D” or “Button U” events

- **One button mouse easier to handle**

- **Multiple button mouse requires training**

- people have been successfully trained already

- **Mouse acceleration big success.**

- Are there other applications of acceleration?



# Touch Screens

- **Not exactly the same as a mouse**
- **Click: no button down or button up**
  - dwell: leave finger in same location
  - double click hard to hit same pixel twice in a row
- **Movement: absolute, no consistent origin**
  - top left, or bottom right can be (0,0)
- **Drag: very difficult, need "modal" command**
  - modes are considered harmful, e.g. shift lock is bad
- **Wheel: perhaps use some gesture?**
- **Not easy to simply replace mouse with touch**



# Keyboards

- It seems simple, precise, and nothing new
- One hand typing --
  - my idea: double keypress -- means mirror key
    - “aa” means “;” “ss” means “ll” & “gg” means “hh”
    - I needed it when I sat with a baby in my lap
  - there exist other one hand keyboards:
    - twiddler (chording keyboard)
    - half keyboard
  - Telephone pad keyboard: 1-2, 2-3, 1-4, 1-4-2-5, 2-5
- Keyboard entry not exact
  - on-screen keyboard
    - a “G” could be an “f”, “t”, “h”, or “b”
  - Cellphone keyboard
    - a “G” could be a “4” or “H” or “l”

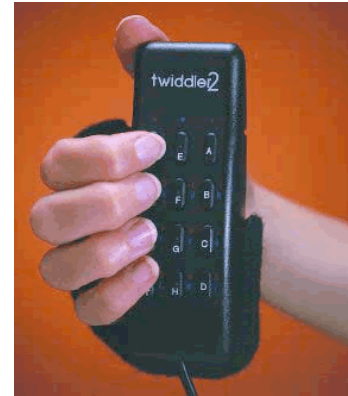
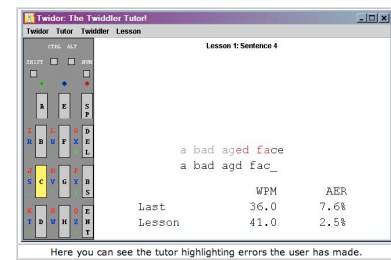


# Virtual Laser Keyboard



Long list of critiques as to where and when this works and does not work.

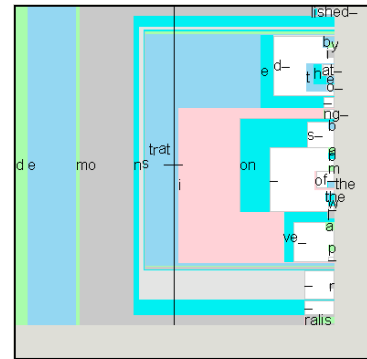
# Twiddler Keyboard



Now we see what keyboards have to do with mobile computing. It is closely related to what the wearable computing folks study.



# Dasher



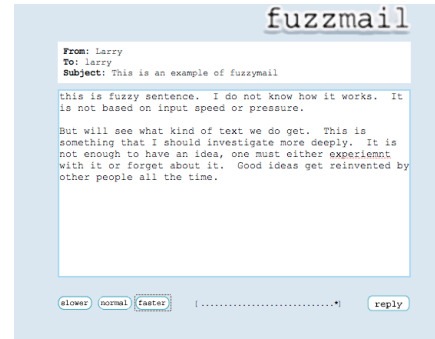
- go to [www.inference.phy.cam.ac.uk/dasher](http://www.inference.phy.cam.ac.uk/dasher)



Dasher is a system that allows a user to input text as if playing a video game. It has lots of applications for disabled people, but maybe also for people on the go.

# Expressive Typing

- **Writing with a pen on paper does more than express words:**
  - it is a picture & conveys non-verbal expression
- **Use the intercharacter typing speed to vary either the font or the inter-character spacing or vertical alignment**
- **fuzzymail.org is like this to show speed directly**



Pervasive Computing MIT 6.883 Spring 2007 Larry Rudolph

This is just a pet idea of mine that I never have time to work on but it is getting easier to implement each year. Popular tagging technology, where the size of the tag is proportional to its importance is close to my idea. The difference is that feedback is important. If one can see how their typing affects how the words look, then ones typing habits can adapt. This bio-feedback (within a 1/10 of a sec) is probably a fundamental law that has yet to be fully exploited by UI folks. The video game people understand this very well which is why manuals are not needed.

# GUI Programming

- Embed in code
  - Java AWT, Python TCL,TK
  - Very hard to code, debug, maintain and modify
- Use GUI builder, e.g. Visual Basic
  - rapid prototype
  - reliable
  - consistency across applications
  - easier to implement “help” and “undo”
  - easier to port

# Glade (for linux, x-windows)

- Three layers:
  - user application, user GUI, window system
  - we care about first two, glade deals with GUI
- Application separate from GUI
  - Interface is via “callbacks”
  - Each widget has a set of standard interfaces
- see <http://glade.gnome.org/index.html>

# 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



We have seen some examples as to how users might interface with handheld devices. A handheld device is always with you and so can be used to detect, record, and communicate your location (well, actually its location). A handheld device can interact with other devices in the surrounding environment -- via bluetooth for example. Having a smart devices sense it's environment, it can improve the performance. As a side note, it is worth pointing out that it can also easily lie about the results.

## 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



Handheld devices have all the functionality of computers, but with more constraints. On screen keyboards and number pads are error prone and slow to use.

There are alternatives. Dasher is an example. The size of letters scrolling across the screen are proportional to their likelihood of being selected. The better the model, the faster and more accurate the performance.

Note that it is not the letters but the box around the letters. The implementation is critically important. How it is presented to the user can make or break a good idea.

# Bi-directional Abstraction Barriers

- Expectations: (probability distribution)
  - engine --> interface
- Disambiguation: (choices and their likelihood -- n-best)
  - interface --> engine



# 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



Pervasive Computing MIT 6.883 SMA 5508 Spring 2006 Larry Rudolph

there are different ways to make stuff happen: direction manipulation and indirect can drag and drop, or click. Or can issue indirect command. Turn on computer causes all sorts of things to happen. But there is also the issue of declarative versus procedural. Latex vs word.

for gui vs menu --

Views, localization... start with strings: 1. do indirection and keep file with strings in the appropriate language. Note this works for “scalable” interfaces -- short vs long strings.

but that is not good enough: what about languages that write in different directions. So want whole view.

Who should have access to these views.

my universal



# 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.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.ao_sleep(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?

