Python on Symbian
Part II

MIT 6.883
Spring 2007
Larry Rudolph

Massachusetts Institute of Technology

Goals

- Mobile Communication: Finding your mate
- Symbian (Python) GUI structure
- Next P-Set

Finding a mate in a static world

- Given two people (or agents, processes)
  - Alice wants to send message to Bob
- What if Alice does not know Bob's address?
  - Bob queries Alice
  - Bob publishes his address at known location
  - Alice posts her desire at known location
  - Alice goes through intermediary (mutual friend)

Alice and Bob have addresses. If Alice knows Bob's address, it is easy for the system to route a message to him or to setup a circuit (physical, logical, or virtual). If Alice does not know Bob's address, then there are several choices. (1) Bob can periodically query Alice if she wants to chat. (2) Bob can publish his address in a place that Alice can find (Alice needs to know some facts about Bob) (3) Alice can go through an intermediary (mutual friend). (4) Alice can post her desire to talk to Bob at a well know location and hope that Bob looks there.

There are examples of each one of these possibilities with computers today. Known IP address, known DNS name, Chat rooms, and so on.

Finding a Mate in Dynamic World

- Fixed address, infrastructure maintains dynamic route -- so how does it do it?
  - Similar, recursive problem
    - phone registers with tower and that info is recorded in central database. Updated each time switch to new tower
- Static Locations -- via web
  - Polling is the key issue
- Proximity

Lots of problems and issues. The phone network is always on, but not the internet (GPRS or Wifi) because it is too expensive in terms of power.

Our choices

- Telephone call -- not really
- SMS message -- when is this good?
- Internet (GPRS or Wifi)
- Bluetooth

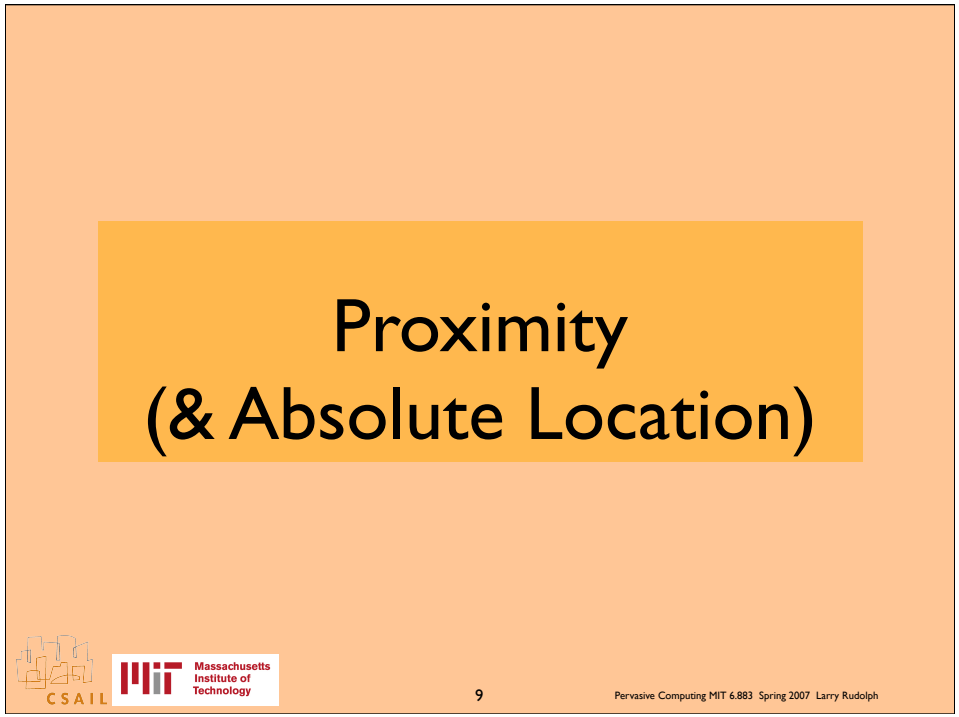
Server Code

```
import sys, socket

if len(sys.argv) < 2:
   print "usage: socketserver <port>"
   sys.exit(2)

# create the server socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

port = int(sys.argv[1])

# allow the socket to be re-used immediately after a close
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

s.bind( ("0.0.0.0", port) )

s.listen(5)             # start the server socket

(client, address) = s.accept()
print "accepted connection from %s:%d" % (address[0], address[1])

while True:
   data = client.recv(1024)
   if len(data) == 0:
      print "connection with %s closed." % address[0]
      break
   sys.stdout.write(data)
client.close()
```

Client Code

```
import sys
import socket

if len(sys.argv) < 3:
   print "usage: socketclient <address> <port>"
   sys.exit(2)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.connect( (sys.argv[1], int(sys.argv[2]) ) )

print "connected.  type stuff."

while True:
   data = sys.stdin.readline()
   if len(data) == 0:
      print "closing connection with server"
      break

   s.send(data)
```

A major theme of mobile computing.



Another major theme of mobile computing.

Symbian (Python) GUI Structure

- What are the choices?
  - Event loop: wait for event; if not relevant then pass it on; otherwise, call routine to handle the event
  - Very error prone (code bug freezes device)
  - Too much code to write
    - want help for typical cases

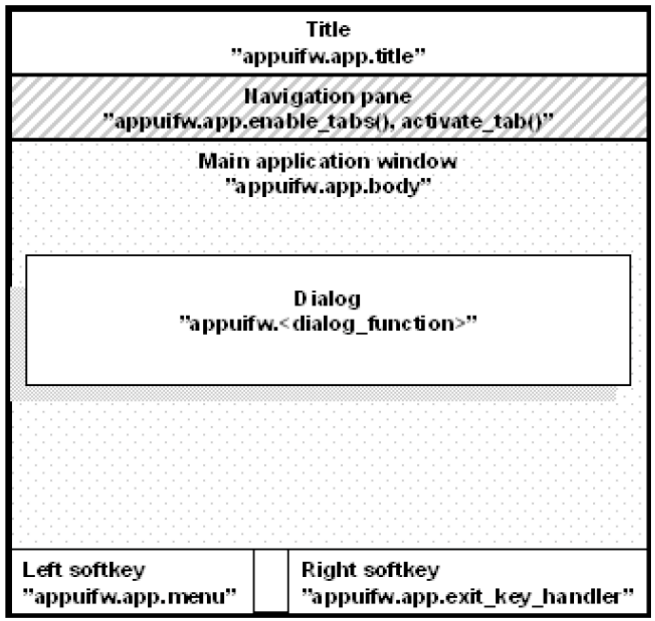
Event-Driven

- Register “callbacks” or ”handlers” for events that can happen.
- Callbacks run for short times.
  - If need longer, then do it in another thread
- What is an “event”?
  - extensible?
- How to register for event?
  - synchronization issues

It is like regular GUI systems, but with more restrictions.

Relevant Events?

- Menu Selection (pull-down & popup)
- Listbox Selection
- Any key presses (down & up)
  - done via image manipulation
- Time outs, Message Arrival, Phone Calls
- Notification from other applications


Title
"appuifw.app.title"
Navigation pane
"appuifw.app.enable_tabs(), activate_tab()"
Main application window
"appuifw.app.body"
Dialog
"appuifw.<dialog_function>"
Left softkey
"appuifw.app.menu"
Right softkey
"appuifw.app.exit_key_handler"

SPy60 Approach

- provide option for more usable screen area
- great for prototyping.
  - Use default font & size; minor graphics

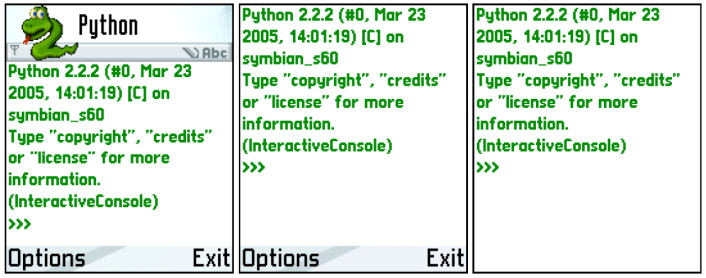
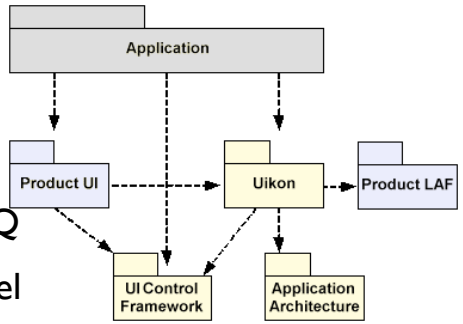


Figure 5.3: UI layouts. left: 'normal', middle: 'large', right: 'full'

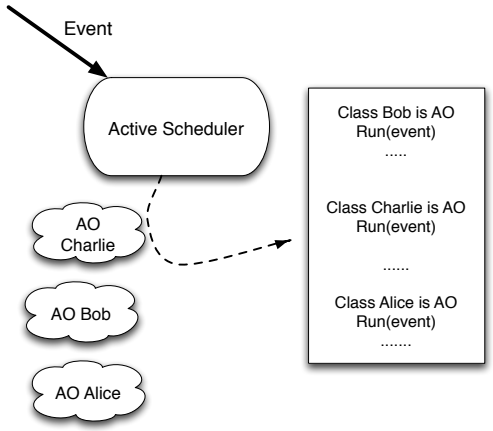
Symbian UI

- Uikon is generic user interface components
- Product UI is S60 or UIQ
- Product LAF: Look & Feel
  - None of this is relevant to us now



Active Objects

- If Symbian written today, AO's would be called "listeners"
- Get called by scheduler (have a little bit of state)
- Run to completion then return to scheduler



Python's AO

- Previous discussion was for Symbian in general
- Python hides the scheduler
  - but after setting up callbacks, just do a return
- Can control AO by allocating an e32.Ao_lock() and then doing wait() and signal() with this lock

Libraries for Nokia

- appuifw: nokia ui interface
  - (UIQ is another ui but for Sony-Ericsson)
- e32: symbian specific library
- special purpose libraries:
  - graphics, e32db, audio, sysinfo, telephone, contacts, location, camera, messaging, calendar
- print (could be redirected to file, via e32._stdo) with putools: comes to console

There are two main libraries. The first is the application user interface frame work (appuifw).
Obviously, it contains all the user interface routines that are part of Nokia's user interface.
There are other user interface frameworks, UIQ is the other famous one.

Using the screen

- Appuifw contains an instance of the class application, called app

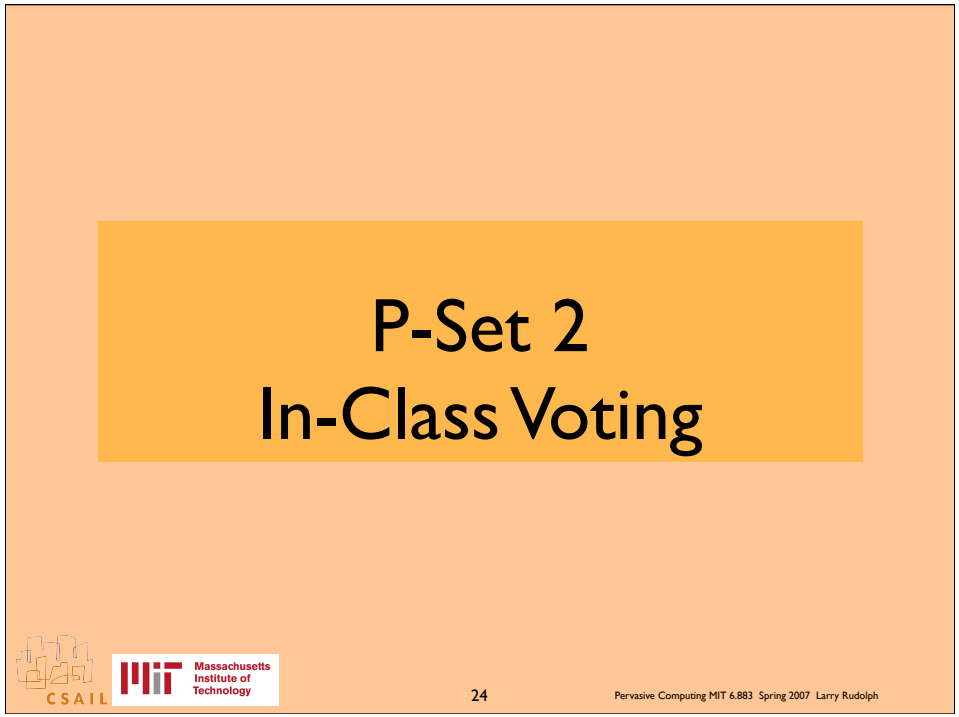
```
appuifw.app.title = u'title of screen'
appuifw.app.screen = 'normal' # size
from appuifw import *
app.body = Text | Listbox | Canvas
app.menu = list of (title, callback)
app.set_tabs( list of tab names, callback)
```

```
import e32
import appuifw

class MyApp:
    def __init__(self):
        self.lock = e32.Ao_lock()

        self.old_title = appuifw.app.title
        appuifw.app.title = u"My Application"
        self.exit_flag = False
        appuifw.app.exit_key_handler = self.abort

        appuifw.app.body = appuifw.Listbox([u"Loading..."], self.handle_modify)
        appuifw.app.menu = [  (u"Add", self.handle_add), (u"Delete", self.handle_delete)]
```

```
    def loop(self):
        while not self.exit_flag:
            self.refresh()   #  do any updates
            self.lock.wait() #    Let the active object scheduler do its thing
        self.close()

    def close(self):
        appuifw.app.menu = []
        appuifw.app.body = None
        appuifw.app.exit_key_handler = None
        appuifw.app.title = self.old_title

    def abort(self):              # Exit-key handler.
        self.exit_flag = True
        self.lock.signal()        # loop code will now continue

#  start here
ap = MyApp()
ap.loop()
# all done
```

```
# accepts an sms message, then forwards the contents to a web server,
#  and returns an sms message with the reply
import e32, appuifw, inbox, messaging
import sendToServer

dns = "www.upcdatabase.com"  # or rudolph.csail.mit.edu
handler = "bookland.asp"          # or  'SeenServer.py'
id = []

def callback(id_cb):
    global id
    id.append(id_cb)
```

```
inb = inbox.Inbox()
inb.bind(callback)
while True:
    while len(id) == 0:   e32.ao_sleep(10)
    id_cb = id[0]
    id.remove(id_cb)

    address = inb.address(id_cb)
    content = inb.content(id_cb)
    fields = [ ('address',address), ('content',content)]
    reply = sendToServer.post_multipart(dns,handler,fields)
    messaging.sms_send(address,reply[:159])
```



Another major theme of mobile computing.

- Motivation: Lecturer would like to know if students are following. Students can continually rate current exposition via their phones. Results appear on lecturers laptop
- Part A:
  - install Python and tools
  - Handle text and keypress input
- Part B:
  - bluetooth communication
  - tree-like voting among phones