Debugging Applications in Pervasive Computing

Larry Rudolph

May 1, 2006

SMA 5508; MIT 6.883

Outline

Video of Speech Controlled Animation

Survey of approaches to debugging

Turning bugs into features

- Speech recognition is not 100%
  - Who likes it when its wrong?
  - Children
    - Example: story telling (easy reading)
    - Computer recognizes the words that the child is reading and animates it

Voice controlled animation

- A very fertile domain: room for improvement
  - mouse is very limited
    - hard to specify parameters
      - choose from list -- awkward when long
      - one action and one parameter
  - speech allows multiple parameters (and sub-parameters)
    - objects are parameters; adjectives are params of params
- Unfortunately, no good models of children's voices
  - so we have to act like children :)

Testbed for other ideas

Naming

- give basic object a name
- give composite object a name (macro)
- many parameters come from context (environment)
- differentiate between base object and instantiated object

Controlling Errors

Two types of consequences to errors:

- something useful (or interesting)
- something destructive (or boring)

Who gets to decide?

- tolerating some errors --> flexibility
- avoiding all errors --> too rigid

Semantics

- Where does the semantics get checked?
  - no consensus (speech, vision, sketch)

Our approach

Command: action and parameters

error: incompatible action and param

dogs: sit, run, lick, beg, bark

cats: sit, run, lick, sleep, purr

Consider the error: “dog purr”

if cat is on stage, it purrs

if dog is on stage, do random action

random actor does random action

Considerations

Really depends on the cost of error

can action be “undone” easily?

is the user getting frustrated?

Rather than selecting at random

choose the most likely action

Informing the user

- System consisted of lots of components on lots of machines
  - flash (XP), galaxy (Linux), audio (iPaq)
  - how to find out about serious errors?
    - cannot inform user; no output dev
    - not clear if other apps will forward

Some Challenges of "traditional" debugging approaches

Stop/Inspect/Go

- Stepping through the code (e.g. gdb)
  - stop and inspect memory & data structures
  - hard to get program to stop or break at correct point
- Run backwards
  - problem usually occurs just before death, so backup and check data-structures
  - Many ops are reversible (x = x + 1 x = x - 1)
  - push on stack control flow and non-reverse ops

Stop/Inspect/Go

- Logs
  - Log all interesting events (I/O ?)
  - Need way to organize independent logs
  - Need way to see paths in the forest
    - visualization tools are helpful
    - extensive log event tags
- Log control-flow history
  - off-line playback or re-execution

Risk of Masking Bugs

Shared Memory (lots of experience)

Many things look like share memory

automatic synchronization; caching; distributed FS

Low-level bugs due to strange timing bugs

set flag; check flag; do operation

Programmers think everything executes at same rate

weird bugs when on process executes a little, pauses, executes a little more, pauses, etc.

Concurrency

- Debuggers don’t deal well with threads.
- Conditional Breakpoints:
  - Break when phone locks DB & camera locks mic
- Need deterministic replay
- Need to understand all possible parallel executions
  - race-condition detector
- Software Transactions (memory & data-base)
  - hand time-outs
  - heart-beat messages

Distributed Communication

Central way to control system-wide parameters

duplicate message detection; non-idempotent operations

unified interface to debuggers on different systems & OS's

- start up; switch between debuggers

Distributed LEDs (one per process)

Virtual Computer

Start with a set of

- Emulators & Virtual Computers

Add

- Scheduler (various orderings)
- Fault-Injection
- Instrumentation

Debug under idealized world

- then move to real world

Yet another approach

Change-point detections

What do you do when things stop working?

Seek out a friend. Their first question: "What did you change?"

Your first response: "Nothing"

Your second response: "Oh yea, thanks"

Too hard with pervasive computing env.

How to support this?

Too hard at the moment to automatically fix all problems.

Worthwhile to point out potential sources

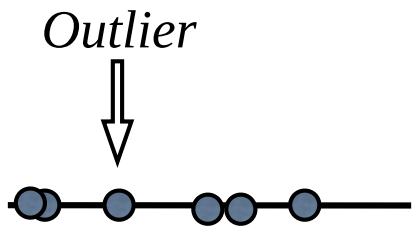
Monitor everything, learn what's typical

- report what is atypical
- monitoring must be on-line and cheap

Use human-level timing

- sec, min, hour, day, week, month, year

Isn’t this like data-mining?



- Data mining for failure indicators?
  - No long log files; no labeled data
  - On-line and easier
- Finding outliers is expensive
- Finding what recently changed is cheap

Use out-of-band communication

If main application has problems

- error messages may not get forwarded
- normal channels of communication might be the source of difficulties
- want separate communication channel

Use IM & SMS for query

- ubiquitous, natural, usually works

Wrapping up

My conclusion is that

- physical world poses new challenges
- user's must help in fixing problems
- system must help the user in this task
- we've only just begun ...