



Architecture and Organization of UI Software

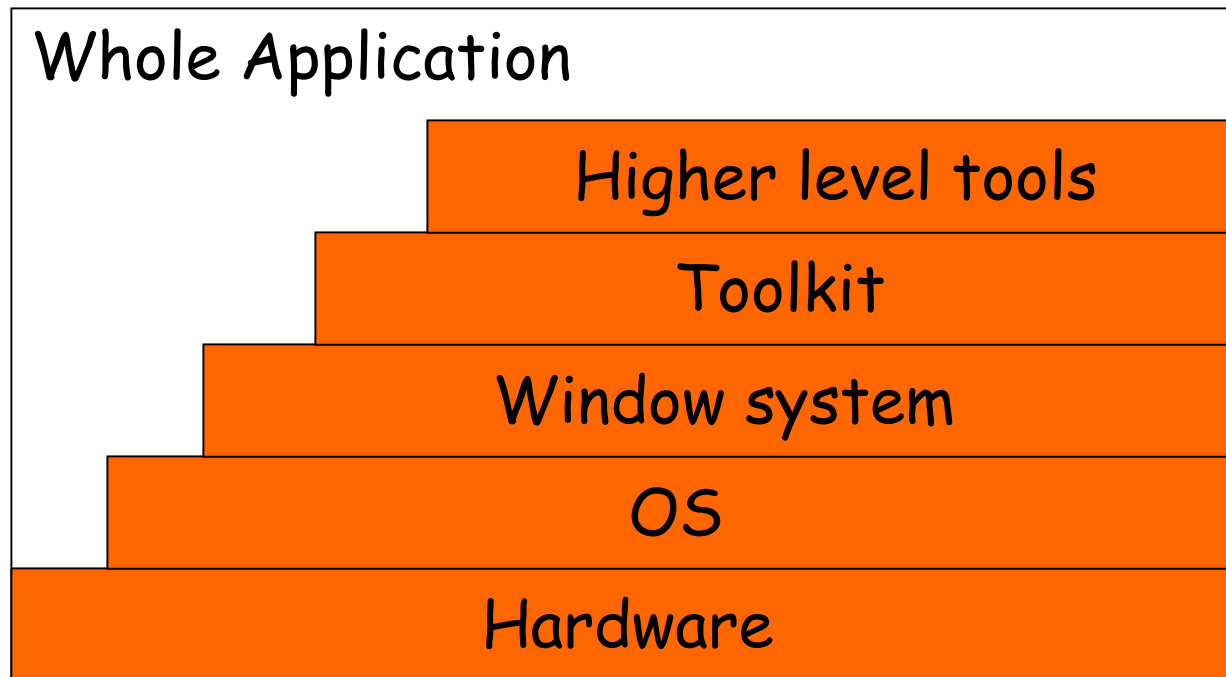


3 Ways to look at it



- Layers
- Detailed tasks
- What the code looks like

View #1: Break UI tasks into layers



Hardware



- Output: raster display device
- Input: keyboard & mouse
 - others
 - | pointing devices (trackball, etc)
 - | knobs and sliders, banks of buttons
 - | touch screens, tablets (e.g. pen input)
 - | 3D locators
 - | Voice input, DataGlove, Eye tracking

OS, Window system, Toolkits



- OS
 - Not really concerned about it here
- Window system
 - provides a virtual device interface
- Toolkits
 - Library of reusable interface components
 - Central level we program in

Higher level tools



- Automate aspects of producing UIs
- Different names
 - User Interface Management Systems (UIMS)
 - User interface builders
 - User interface design/development environments

Whole Application



- Often distinguish between
 - UI parts
 - "core functionality" (application)
 - Carry out actions on behalf of user
- UI has two interfaces
 - one w/ user, one w/ application
 - Challenge: balancing demands of each

Strong or weak separation?

- SE view: a strong separation is good
- UI view: may not be
 - user doesn't know/care
 - UI's take large % of code
 - hard to tell bits apart!
- "Separation of concerns"

View #2: Detailed tasks



- Task oriented vs. systems oriented
 - What does a UI need to do?
 - How do we organize SW to do it?
- Partially depends on UI style/metaphor
- Two dominant metaphors for UI
 - *conversational or language* metaphor
 - *the simulated world* metaphor

The conversational metaphor



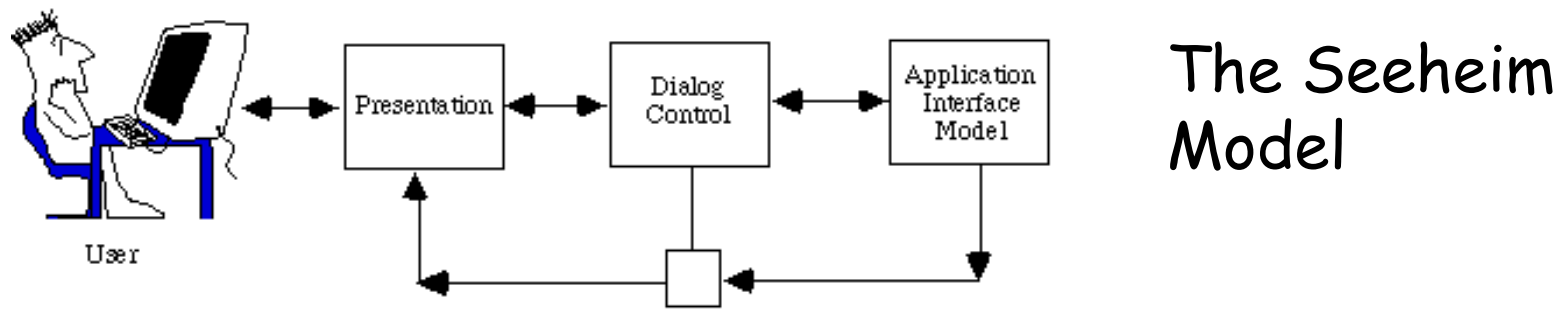
- Aim: analogous to human conversation
 - User "talking" to computer in a "language"
 - Computer responding in some "language"
 - i.e. command and menu style interfaces
- Dominant approach till early 90's

Interface as a Language Interpreter

- Lexical: accept low level inputs
- Syntactic: parse to assign structure
- Semantic: assign meaning, act on them


- Produce results in "output language"
- Input/output media are typically related
 - "Inter-referential I/O"

An architectural model based on the language metaphor



- Presentation: lexical level I/O
- Dialog: syntactic level (dialog mgmt)
 - Input in context: where, why, next?
- Application interface: input translation, output abstraction

Other major metaphor: Simulated Worlds



- UI as collection of (simulated) objects
 - Direct manip. of objects vs. conversation
 - Avoid conversational intermediary
- Conversation hasn't really gone away
 - illusion of directness
 - same kind of tasks, on a per object basis

Seeheim architectural model revisited



- Same parts, but dialog less imp.
 - per object basis vs. whole system
 - syntax localized hence simple
- Separation of Concerns, again
 - SE: we want to separate concerns
 - DM metaphor: want to mix into each object
 - return to this again in toolkits (OO methods)

Underlying Commonality



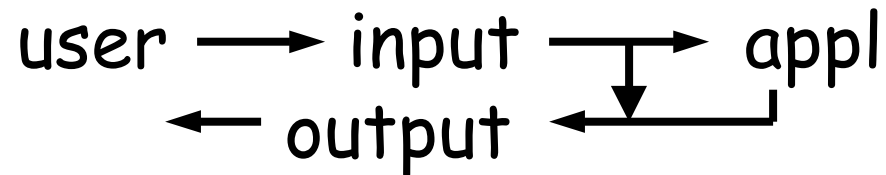
- The same central task
 - UI is a mechanism that translates manipulation of input devices into actions carried out for the user

View #3: What does the code look like?

■ Application Level

User ↔ UI ↔ Appl

■ Information flow



User Side



- Input: events
- Output: photons

Application Side



- Input and Output
 - procedure or method calls
- Input side: "callbacks"
- Recall conversational metaphor
 - input => commands/actions
- UI code is similar
 - input => procedure calls (or messages)

Callbacks imply a major restructuring of app code

- "Normal" code provides all control flow
 - UI code is acting on behalf of the user
 - => main control flow in user hands
- Code is reactive rather than pro-active
 - "event driven programming"
- Usually harder
 - Programmer has little control
 - harder to modularize/structure

So, what does this look like in the code?



- Initialize;
- Repeat
 - Wait for next user action (expressed as "event record");
 - Decide what it means;
 - Act on it;
 - Update the screen to reflect the changes
- Until done;

In OO, distribute interp. and handling across objects

- Initialize;
- Repeat
 - Wait for next user action (event);
 - Decide which object(s) this affects;
 - Pass ("dispatch") event to object(s) as message(s);
 - Redraw screen
- Until done;