

Toolkits



Global Dialog in Simulated World

- Existence/visibility/enabling of interactors
 - e.g., menu items are disabled when they can't be legally invoked
 - "special" dialog components (e.g. save dialog box) are hidden or do not exist until needed
- Input distribution
 - e.g., modal dialog box that gets all input

Where is Global Dialog Handled?

- Typically ends up as part of "application"
- But, it *is* an interface issue
 - lack of explicit support tends to mix application and UI
 - leads do ad hoc techniques in application
 - hard to implement, harder to change/maintain

Need better support for Global Dialog

- Want something as powerful as FSM/ATN/RTN/... is for conversational
- How?
 - Go back and adapt parsing based techniques?
- Need "piece-wise" approach
 - One possibility: Rule based systems

Rule Based Systems



- In general:
 - working store (e.g., blackboard)
 - rules: "if pattern/predicate then action"
 - "fires" when match in working store match
- Can add new ones easily
 - Can't with parsing based methods
 - New/old rules can interact in strange ways

How are action part of rules used?

- To change interactor enable/disable status

- e.g., change enable/disable of "save button"

- | if `unsaved_chg` -> `save_button.enabled = true;`
- | if `!unsaved_chg` -> `save_button.enabled = false;`
- | if `makes_change(current_command)` ->
`unsaved_changes = true;`
- | if `is_save(current_command)` ->
`unsaved_changes = false;`

How are action part of rules used?

- To create/destroy interactor (sub-)trees
 - e.g., modal dialog box
- To invoke application components based on global situation
 - e.g., password dialog
(login only when specific conditions met)


Example: pre- and post-conditions in UIDE

- Blackboard is simple set of conditions
- Pre-conditions express predicate over those conditions
- Post-conditions express effect of action
- Rules of form:
if <pre-condition> then <post-condition> +
<side-effect-action>

Why have post-conditions?

- Why not just fold into the action?
- Analysis via back-chaining inference
 - Ask “why is this menu item disabled”
 - | chain backwards, seq. of rules to enable item
 - | translate into an explanation
 - Can support things like animated help: “how do I do X?”
 - | chain backwards, seq. of rules that will do X
 - | work forward animating UI actions needed to cause those firings

Application Interface



- Recall overall flow/outline:
 - Input
 - Application Interface
 - Output

- App -> Toolkit: API
 - no "separation of concerns"

Application Interface: Toolkit -> App: Callbacks

- Some API Goals:
 - Reflect app semantics in UI
 - No new mechanism for app code
- Callbacks fail at 1st (no semantic meaning)
- Callbacks are messy
 - Little procedures
 - Not modular

Rules and the API



- UI -> app: action part of rules
- App -> UI: modify blackboard

- Better separation of concerns
 - More abstract than tree manipulation API
- Requires app to adopt UI mechanisms
- Another mech. next week: constraints

Output



- Basic update cycle, again:
 - Access and dispatch input
 - Act on input (modify interactor trees)
 - Redraw from tree

- How do we do the last step?
 - Geometry management
 - Redraw

Why view output as two parts?

- Why not compute size & position immediately prior to redraw?
 - dependencies may not follow redraw order
 - | i.e. may need to know size/pos of something we haven't drawn yet
 - keeping size & position correct as modified
 - | lots of wasted work

Output: Geometry Management

- Distribute responsibility among objects
 - e.g., in Swing containers have layout mgrs
- Toolkit may provide more support via geometry management policy
 - Artkit interactors use very simple policy
 - size is bottom up (parent size based children)
 - position is top down (parent determines child pos)
 - this policy has some problems
 - i.e., what happens when a window changes size?

Example: X_t

- Position is top down (or explicit to kids)
- "Negotiation" for size
 - node receives "suggestion" from parent
 - allocates and suggest size to kids
 - receives requests back from kids
 - node makes final decision, sets kids sizes
 - passes size request to parent
- In practice, tends to be top-down policy

Xt example, cont'd

- Node uses policy to allocate kids space
 - e.g., give equal amounts to each child
 - give fixed amount one child, rest to other
- There are more sophisticated policies
 - Interviews, some Swing layout mgrs
 - boxes and glue model from TeX
 - Some use more general mechanisms
 - constraints

Output: Redraw

- Distribute work among the objects
 - e.g., Swing, define *paintComponent(Graphics)*
 - | automatic if model changes
 - tell container, which works down till leaves are done
 - | *repaint()*, *revalidate()* if necessary
- Problem/opportunity:
 - usually the whole screen isn't changing
 - only need to update parts
=> incremental redraw

Incremental Redraw Example: Arkit

- Interactors inform system when appearance (or size or position) changes
- System tracks "damaged" areas, only updates interactors that need it
 - single rectangle that encloses all damage
 - at redraw time, use it as clipping region
 - Can do "trivial" accept/reject tests
- Better incremental update: Asente