Dialog Design: Introduction and Command Languages

This material has been developed by Georgia Tech HCI faculty, and continues to evolve. Contributors include Gregory Abowd, Jim Foley, Elizabeth Mynatt, Jeff Pierce, Colin Potts, Chris Shaw, John Stasko, and Bruce Walker. Last updated by Valerie Summet, Emory University. Permission is granted to use with acknowledgement for non-profit purposes. Last revision: Feb. 2011.
Agenda

• Dialogue design styles and issues
• Command languages
  ▪ Advantages, disadvantages
  ▪ Design guidelines
Dialog Styles

1. Direct Manipulation
2. Command languages
3. Speech/Natural language
4. WIMP - Window, Icon, Menu, Pointer
5. Gesture, pen
Dialog Design - Humor

- How does a user interact with the interface?
General Issues in Choosing Dialogue Style

- Who is in control - user or computer
- Initial training required
- Learning time to become proficient
- Speed of use
- Generality/flexibility/power
- Special skills - typing
- Gulf of evaluation / gulf of execution
- Screen space required
- Computational resources required
Command Languages

- Earliest interaction style
  - If you ignore hardware interactions like switches, punched paper tape and cards, and plug boards
- Examples
  - MS-DOS shell
  - UNIX shell
  - dBase
Shell
CL Characteristics

• Little or nothing is visible so…
  ▪ Work primarily by recall, not recognition
  ▪ Heavy memory load
• Poor choice for novices but can be very good for experts
**CL Advantages for Expert Users**

- **Speed, conciseness**
  - Is (hard to beat for speed)
- **Can express actions beyond a limited set**
  - Flags, piping one command to another
- **Repetition, extensibility**
  - Scripting, macros
- **Power**
  - Abstraction, wild cards
- **Little run-time cost**
  - Used to be a huge consideration
  - Irrelevant except in special situation
Some CL Disadvantages

• Error-prone
• Harder to learn
• Requires typing
• With added power, comes added responsibility and danger
  ▪ UNIX
    - `rm -r f*`
    - `rm -rf *`
    - Deletes every file that you have, provides no feedback, and you can’t get them back
Unix Shell CL Disadvantages

- Learning takes a long time
- Hard to remember command names
- Some command names don’t make sense, so have to memorize
- No in-progress feedback - how much longer?
- System state is invisible, and have to know which commands to use to get which information
- Hard to make sense of outputs, such as with `ls` - no headings, no code interpretations
- No “look”
- No warning if bad things are going to happen
- No universal Undo; to reverse a command, have to know the inverse command (create directory, delete directory)
- Have to use `man` command to find help for the desired command
- Because commands are short, typos can lead to incorrect command
- Inconsistent flag meanings
- Inconsistent parameter orders
- Have to type a lot - touch typing needed
CL Reflection

• Command languages are often maligned (for good reason)
• But increased functionality can win out over bad UI (e.g., UNIX)
  ▪ Try to get both
  ▪ Avoid excess functionality (comes at cost)
CL Design Goals/Guidelines

- Consistency
- Good naming and abbreviations
- Doing your homework in design can help alleviate some of the negatives
CL: Consistency

- Provide a consistent syntax
  - In general: Have options and arguments expressed the same way everywhere
  - UNIX fails!
    - Commands were developed by lots of different people at different organizations
    - No guidelines / style guide
- If commands are long, have simple and consistent abbreviations
CL: Consistency (Syntax)

- Simple command list
  - e.g, vi, minimize keystrokes
- Commands plus arguments
  - realistic, can provide keyword parameters
  - `cp from=foo to=bar`
- Commands plus options plus arguments
  - what you usually see today
  - `cp -r foo bar`
CL: Order

- English: S-V-O (Subject-Verb-Object)
  - “you” is assumed as the subject - imperative!

- CL: S assumed (you)
  - Is V-O or O-V better?
  - % delete file or % file delete

- V-dO-iO vs. V-iO-dO
  - (dO = direct Object, iO = indirect Object)
  - % print file calvin
  - % lpr -Pcalvin file
  - Which is better?
CL: Ordering

- Keep ordering consistent
  - VO (Verb Object) seems to be the most natural
  - Typically need to pick where options go
- Example of inconsistent ordering
  - `ln -s target linkname`
  - `cp file1 file2`
CL: Terminology

• Keep terminology consistent
• Same concept expressed with same options
• Useful to provide symmetric (congruent) pairings
  ▪ forward/backward
  ▪ next/prev
  ▪ control/meta
CL: Example of Congruent Pairs

- vi text editor
  - w - forward word
  - b - backward word
- Wouldn’t ‘f’ be better for forward?
  - ‘f’ already used
- How about ‘fw’ and ‘bw’?
  - Extra keystrokes
CL: Names and Abbreviations

• Specificity versus Generality
  ▪ General words
    – More familiar, easier to accept
  ▪ Specific (typically better)
    – More descriptive, meaningful, distinctive

▪ (Nonsense does surprisingly well for small set of commands)
CL: Abbreviations

• Abbreviations allow for faster actions
  ▪ Expert performance begins to be dominated by motor times such as # of keystrokes
  ▪ Not good idea for novices
  ▪ (Allow but don’t require)
CL: Abbreviation Strategies

• Simple truncation (works best, but conflicts)
• Vowel drop plus truncation (avoid conflicts)
• First and last letters
• First letters of words in a phrase
• Standard abbrev from other contexts
  ▪ qty, rm, bldg
• Phonics
  ▪ xqt
CL: Abbreviation Guidelines

• Use single primary rule (with single fallback for conflicts)
• Use fallback as little as possible
• Mark use of fallback in documentation
• Truncation is good but generates conflicts
• Fixed length is better than variable length
• Don’t use abbrevs. in system output
• Can use auto-completion as alternative to abbreviations
CL: Design Affects Performance

Terse CL
Find:/tooth/; -1
List:/ko;*
Rs:/ko/,.ok/

Wordy CL
backward to “tooth”
list all lines with “ko”
change “ko” to “ok”

<table>
<thead>
<tr>
<th>User type</th>
<th>% task completed</th>
<th>% wrong commands</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Terse</td>
<td>Wordy</td>
</tr>
<tr>
<td>Inexperienced</td>
<td>28</td>
<td>42</td>
</tr>
<tr>
<td>Familiar</td>
<td>43</td>
<td>63</td>
</tr>
<tr>
<td>Experienced</td>
<td>74</td>
<td>84</td>
</tr>
</tbody>
</table>

All command words can be abbreviated with a single letter.

Ledgard et al, “The Natural Language of Interactive Systems,” CACM, October 1980 556-563; see also subsequent letters to the editor.
Key Ideas

• Multiple Dialogue Styles
  ▪ Each has pros and cons

• Command Language Pros and Cons

• Command Language Design Guidelines