Copy & Paste, Drag & Drop

Interaction Techniques and Technologies (ITT), SS 2017
Session 21 (20.07.2016), Raphael Wimmer

Overview

These are slides/notes for the lecture, automatically generated from the slide set. Please extend this outline with your own notes.

Goals for this and next Week

Overall: Learn more about generic interaction techniques and tracking

Know

- History and implementation of Undo, Copy&Paste

Learn

- using algebra for tracking / transformations

Practice

- put all knowledge from this course to work

Common Interaction Techniques and Implementations

Shneiderman's Eight Golden Rules of Interface Design

1. Strive for consistency.
2. Enable frequent users to use shortcuts.
3. Offer informative feedback.
4. Design dialog to yield closure.
5. Offer simple error handling.
6. Permit easy reversal of actions.
7. Support internal locus of control.
8. Reduce short-term memory load.

<small>Shneiderman, B. and Plaisant, C., Designing the User Interface: Strategies for Effective

Human-Computer Interaction: Fifth Edition, Addison-Wesley Publ. Co., Reading, MA (2010), 606 pages. online </small>

Overview

- Clipboard, Drag and Drop
  - User Interfaces
  - Interactive exploration
  - Implementations in different operating systems

Clipboard

General Concept

Questions:

- What is a *clipboard*?
- How does the user interface for clipboards look like?
- How can clipboard functionality be implemented?

Buzz Group

How does a clipboard *behave*? What do you expect from a typical clipboard?

Qt Clipboard inspector

```
~~~~ show_clipboard.py
#!/usr/bin/python3
from PyQt5.QtWidgets import QApplication

app = QApplication(["",""])
clipboard = app.clipboard()
mimeData = clipboard.mimeData()
print("Current clipboard offers formats: " + str(mimeData.formats()))
for f in mimeData.formats():
    print("---- %s ----" % f)
    data = str(mimeData.data(f))
    if len(data) > 100:
        print(data[:100] + " [...]")
    else:
        print(data)
    print("")
```

~~~~

Microsoft Windows

- Clipboard formats.aspx) describe the type of data in the clipboard
- Clipboard may contain the same data in different formats (e.g., as plain text, HTML, RTF)
- Standard formats.aspx) (CF*BITMAP, CF*UNICODETEXT, …)
- Registered formats (custom, for data exchange between applications, e.g., CF_HTML.aspx)
- Private formats (for internal use)
- Access via SetClipBoardData(format, data), IsClipBoardFormatAvailable(format).aspx), EnumClipboardFormats.aspx), GetClipBoardData().aspx)
- Clipboard needs to be explicitly cleared before copying data to it.
- Further Reading: „How the clipboard works“ (Part 1, Part 2)

Mac OS X

- „Pasteboard“ for historical reasons
- manager process `pbs` accessed via APIs
- Standard classes for clipboard content (`NSString, NSAttributedString, NSImage, NSURL, NSColor, NSSound`), general class `NSPasteboardItem` or custom classes adopting `NSPasteBoardReading` protocol
- additional *find buffer* for text searches
- see also: pasteboard programming guide

iOS

- *General Pasteboard* and *Find Pasteboard* similar to OS X
- apps can create additional pasteboards
- `Uniform Type Identifier` (UTI) describes type of data (e.g., `public.jpeg` or `com.myCompany.myApp.myType`)
- `Pasteboard` class provides convenience methods for copying/pasting images, URLs, …
- (documentation)

X11 (Linux/BSD)

- Main difference to OS X and Windows: no data is actually stored in the clipboard
- Instead:
  - On „copying“ data to the clipboard, the application tells the X Server that it now owns the `CLIPBOARD` selection.
  - When pasting data from the clipboard, the application directly requests the data from the application that owns the `CLIPBOARD` selection.
  - Content negotiation: pasting applications asks for data types the application can provide and can also try to request data in arbitrary format.
- furthermore: `PRIMARY` buffer (holds selected text, pasted via middle click) and `SECONDARY` buffer (mostly unused), similar mode of operation as clipboard.
~~~~

- data types identified via MIME types

Further reading: explanation by Jamie Zawinski, Freedesktop.org clipboard "specification", ICCCM: Peer-to-Peer Communication by Means of Selections

Android

- Clipboard holds one `ClipData` object at a time, consisting of:
  - `ClipDescription` object containing list of MIME types
  - one or more `ClipData.Items`, all having the same type: text, URI, or Intent
- *Content providers* allow for retrieving data with a specific MIME type via a URI
- Usage: `[ClipboardManager](http://developer.android.com/reference/android/content/ClipboardManager.html).setPrimaryClip(ClipData)`
- facilities for copying data structures and streams

Qt Implementation

- Wrappers for every platform (e.g., X11, Windows, OS X)
- QClipboard
- data type indicated via MIME

Further Reading

- "Copy-paste conclusions: put the metadata on the clipboard"
  - „Step by step we've been teasing apart the clipboard to see how metadata could survive a copy-paste between applications. If the metadata survives, then the destination can be used to automatically credit the original source and the creator, without the user having to do it manually." "On Clipboard Formats – 2006-09-15"
  - „The Carbon version of Gecko doesn't interoperate with anything but other Carbon Gecko processes. I figured I should try to do better with the Cocoa nsClipboard.

  This stuff is so underdocumented that it isn't even funny. This document is written so that others might find something when they search the Web."*

Drag and Drop

- X11: DND extension
- Windows: Object Linking and Embedding (OLE)
- OS X: Cocoa Drag Manager
- generally similar to clipboard operation

DnD Qt Implementation

- QDrag, QDrag{Enter,Leave,Move}Event, QDropEvent
- again: wrappers with OS-specific code
- MIME-encoded content (like X11), gets converted for Windows and OS X.
- Code example: tutorial at zetcode.com

Recap

- Different clipboard implementations across operating systems
- Data often available in different formats
- X11 has asynchronous clipboard
- format negotiation is quite a mess

ENDE