Implementing a clipboard interface under X11 - RCOS Presentation

Avi Weinstock

July 21, 2015

Interface provided by rust-clipboard

- An intuitive mental model of clipboards is that there's a global, OS-managed blob of data, with `get` and `set` operations.
- This is not the case under X11 (and doesn't seem to be the case under Windows either, but seems like it might be the case under OSX).

Interface provided by rust-clipboard (continued)

- The goal of `rust-clipboard` is to provide the simple/naïve get/set interface to the clipboard across all major OS's.

-
  ```
  struct ClipboardContext; // innards are OS-dependent

  impl ClipboardContext {
      pub fn new() -> Result<ClipboardContext, &str> { /* ... */ }
      pub fn get_contents(&self) ->
          Result<String, &str> { /* ... */ }
      pub fn set_contents(&self, data: String) ->
          Result<(), &str> { /* ... */ }
  }
  ```

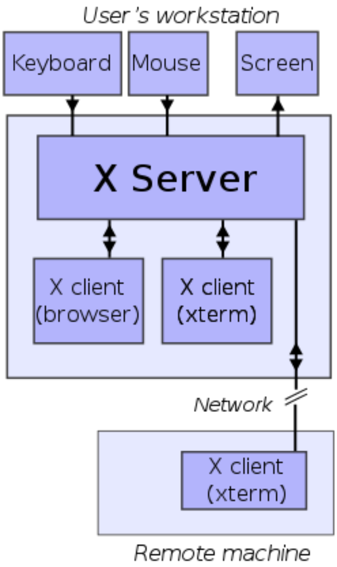
Example program using rust-clipboard

```
extern crate clipboard;
use clipboard::ClipboardContext;

fn main() {
    match ClipboardContext::new() {
        Ok(ctx) => {
            let data = ctx.get_contents().unwrap_or("");
            println!("Current clipboard contents: \"{}\"", data);
        },
        Err(msg) => {
            println!("Error initializing clipboard: {}", msg);
        }
    }
}
```

X Window System background

- The X Window System was developed at MIT in 1984 as a successor to the W Window System.
- Clients communicate with the user (and each other) by sending events through the server.

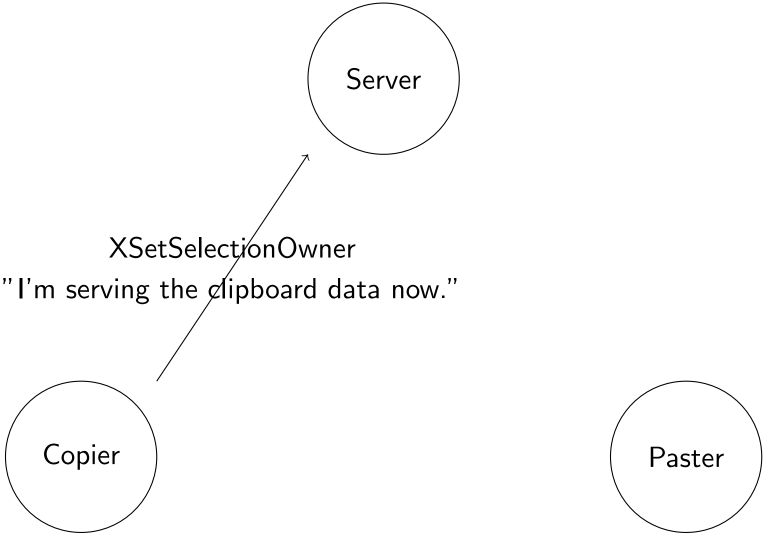


[1]https://en.wikipedia.org/wiki/File:X_client_server_example.svg

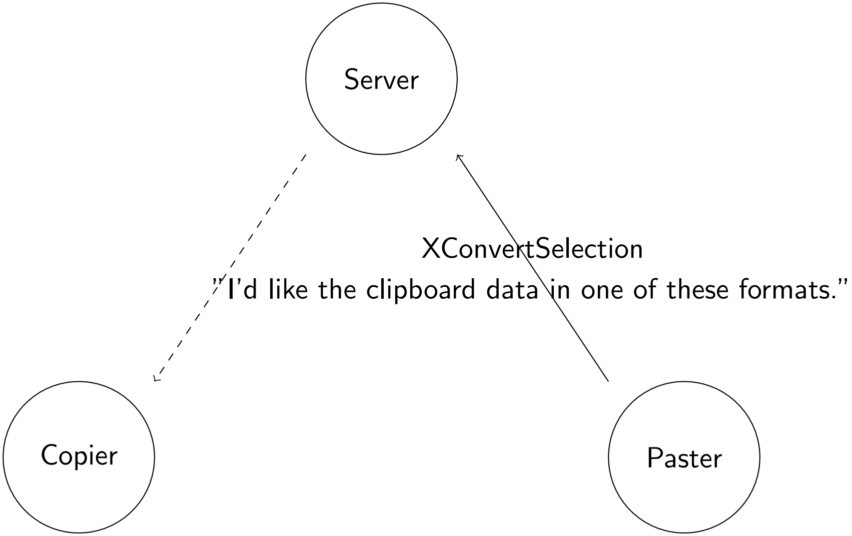
X11 Clipboard protocol

- There is a simple mechanism for copying/pasting under X (called "cut buffers", accessed through `XStoreBuffer` / `XFetchBuffer`, but it's deprecated (due to performance reasons?).
- When a program "copies data to the clipboard" under X11, it actually starts a "server" process (an X client) that is responsible for {streaming, chunking, format-negotiating} the data with other X clients.
- "Retrieving data from the server" likewise involves sending messages to the current clipboard owner (the aforementioned "server").

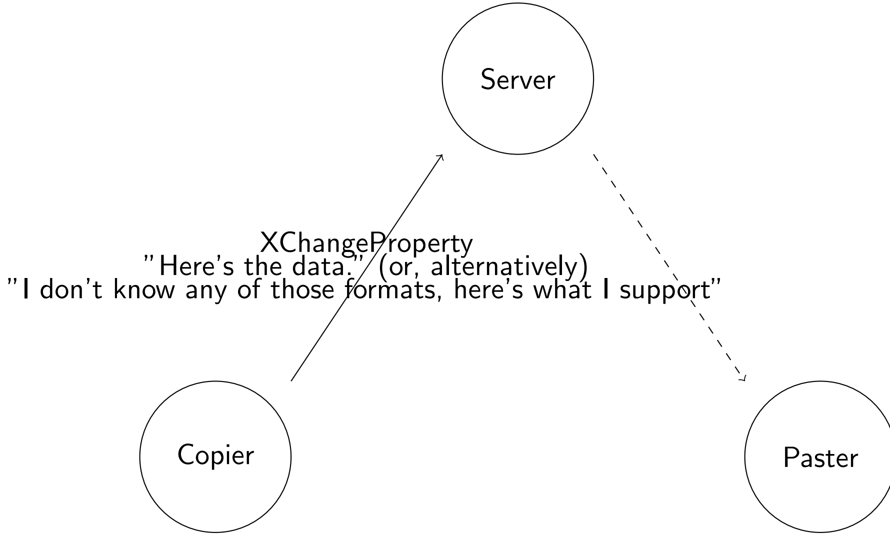
X11 Clipboard protocol



X11 Clipboard protocol



X11 Clipboard protocol


Server
XChangeProperty
"Here's the data." (or, alternatively)
"I don't know any of those formats, here's what I support"
Copier
Paster

The bug that took weeks to resolve

- My initial implementation of copy-to-clipboard seemed to cause paste actions to hang/timeout.
- According to print statements and `ltrace`, all the values were the same in my program and `xclip` (the C program I was using as a reference for the protocol) until the call to `XNextEvent`, which "returned bogus data" (the event's "requestor" field showed up as my library, which then started responding to itself).

Sum types and tagged unions

- Some languages have a feature called ADTs (Algebraic Data Types).
- Haskell:

```
data Result t e = Ok t | Err e
```

- Rust:

```
enum Result<T, E> {
    Ok(T), Err(E)
}
```

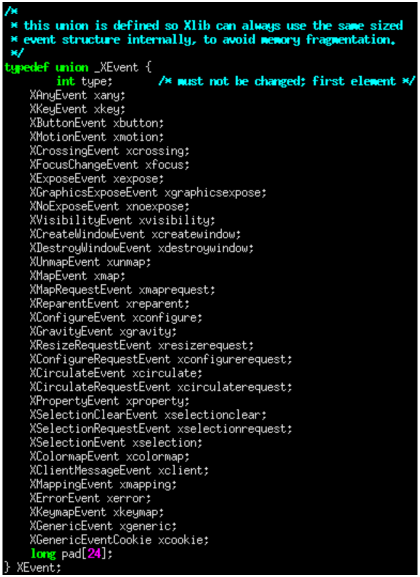
- In C, ADTs can be emulated with "tagged unions".
-

```
#define TAG_OK 0
#define TAG_ERR 1
struct result {
    int tag;
    union {
        void* ok;
        void* err;
    } value;
};
```

- Tagged unions don't (and can't) enforce that the field used is consistent with the tag.

typedef union _XEvent

Xlib (the client library for X11) uses tagged unions for XEvent.



```
/*
 * this union is defined so Xlib can always use the same sized
 * event structure internally, to avoid memory fragmentation.
 */
typedef union _XEvent {
        int type;       /* must not be changed; first element */
    XAnyEvent xany;
    XKeyEvent xkey;
    XButtonEvent xbutton;
    XMotionEvent xmotion;
    XCrossingEvent xcrossing;
    XFocusChangeEvent xfocus;
    XExposeEvent xexpose;
    XGraphicsExposeEvent xgraphicsexpose;
    XNoExposeEvent xnoexpose;
    XVisibilityEvent xvisibility;
    XCreateWindowEvent xcreatewindow;
    XDestroyWindowEvent xdestroywindow;
    XUnmapEvent xunmap;
    XMapEvent xmap;
    XMapRequestEvent xmaprequest;
    XReparentEvent xreparent;
    XConfigureEvent xconfigure;
    XGravityEvent xgravity;
    XResizeRequestEvent xresizerequest;
    XConfigureRequestEvent xconfigurerequest;
    XCirculateEvent xcirculate;
    XCirculateRequestEvent xcirculaterequest;
    XPropertyEvent xproperty;
    XSelectionClearEvent xselectionclear;
    XSelectionRequestEvent xselectionrequest;
    XSelectionEvent xselection;
    XColormapEvent xcolormap;
    XClientMessageEvent xclient;
    XMappingEvent xmapping;
    XErrorEvent xerror;
    XKeymapEvent xkeymap;
    XGenericEvent xgeneric;
    XGenericEventCookie xcookie;
    long pad[24];
} XEvent;
```

The bug that took weeks to resolve (resolved)

- It turns out that I was checking the tag on the XEvent, but then casting it to the wrong substructure.
- The fix was essentially this:

```
  if evt.get_type() != SelectionRequest {
      return false;
  }
- let event: &XSelectionEvent        = unsafe { transmute(evt) };
+ let event: &XSelectionRequestEvent = unsafe { transmute(evt) };
```

Questions?

Thanks

- RCOS
- Professor Goldschmidt
- Professor Moorthy
- The Mozilla Project
- Sean O'Sullivan
- Red Hat Incorporated