The Boat Linux Missed, and How it Can Swim

21 AUGUST 2014

Apparently Linus Torvalds still wants to see Linux conquer the desktop. I agree. I'd like to see it too.

The Pier

I use a Macbook Pro for my personal machine and I now also use a Mac at work, but at my previous job I used a Fedora Linux desktop machine. It wasn't too bad. I know lots of folks hate Gnome 3 but I found it to be a usable, intuitive, and pleasantly minimal windowing environment.

The problem, now as in the past, has always been twofold:

1. Polish is hard.

2. Apps, apps, apps, apps. (And thus developers, developers, developers, developers. The Ballmeister was right, at least on that one.)

   The first is kind of self-explanatory. I ~~finished~~ my most recent project almost two years ago, if that's defined as making it work. But it's not. I'm still working on it because making something work is only about 25% of the way to "done." The rest is making it work *for other people* (excluding extreme technical enthusiasts, which are a small group). To make something work and make it useful to other people means spending multiples of the time it took to make it functional on things like tweaking the UI and massaging the overall user experience.

   I periodically try out Linux distros, and most of the popular desktop ones are *okay* in that department, but not great.

   They're not great because Linux as a whole just doesn't have the resources or the developer base to make them great. The excessive fragmentation of the Linux community into too many distributions (yes I'm on that side of the debate) doesn't help, but ultimately it's not the fundamental reason.

Ultimately you're left with that awful marketing chicken-or-egg problem: nobody uses it because nobody uses it, nobody uses it because nobody develops for it, and nobody develops for it because nobody uses it.

... which brings me to the second point: apps, and lack thereof. Turns out they're related problems. Linux is open source, so if Linux had a massive developer community writing apps for it those developers would also be likely to pitch in and contribute to making the overall platform better.

In developing apps for commercial OSes, there have been many times I've wished I could submit a patch. The ability to draw on the regular developer community to improve the core is one of the strengths OSS has that commercial can't match.

Two Efforts Left Behind

The Linux desktop has been hobbled for a long time by an architectural boat anchor known as the X Window System. It's old. It's bloated. It's crufty. Making it support modern things like ultra-high-rez displays (e.g. Retina) and touch panels is going to hurt real bad.

There are now two parallel efforts underway to replace X with something more modern: Wayland and Mir. One is the "community" effort and the other is an in-house effort by Canonical, the maker of the Ubuntu family of distributions.

The problem with the Linux desktop is that there aren't enough developers and thus not enough people to fix its warts and not enough apps. Now you're going to make it worse by pulling the (albeit old and clunky) X Window System rug out from under the platform? Not only that, but you're going to *fork the already tiny developer community* by doing it *twice*?

I'm not going to retread the Mir vs. Wayland flame war because **they're both wrong**.

Even if Mir didn't exist and Canonical had jumped in with Wayland, the boat would still have been missed.

The Boat That Sailed

Ask yourself: what user-facing platform has the largest developer community?

Mac? iOS? Android? Certainly not. These are too new, and each is its own niche. Windows? *Maybe*, but I think it's quite debatable. It certainly would have been Windows 10 years ago, but today I think it's none of these.

Today I think it's the web.

While the Linux folks were debating Mir vs. Wayland vs. just wrapping more duct tape around X, they failed to realize that **the successor to the X protocol has already been designed, tested, and shipped, and has already drawn the largest and most diverse developer community in the world**.

That successor, of course, is HTML5.

Now let me demolish all the objections.

"It's too slow!"

I think this is a case of premature optimization being the root of all evil. Google's Chromium effort is (mostly) open source and has already fielded a web renderer and JavaScript engine that runs so fast it often outperforms native apps at rich UI tasks.

Step one would be to figure out how to build a windowing environment on top of HTML5. Step two would be to fix whatever performance issues come up.

"It's too heavy!"

Compared to what? X with its five million extensions? Wayland with Qt and GTK running in parallel and a dozen apps running in X11 backwards compatibility mode because every programmer is not going to rewrite their app for the new code base?

"Some apps need native performance!"

Yes, some apps such as games, video players, etc. are going to want direct access to the frame buffer and/or the OpenGL stack. So give it to them. Add a platform-specific extension to your HTML5 engine to allow a piece of the display to be handed down to the underlying controller and addressed directly.

Personally I'd punt this until the core works well. See: premature optimization.

"Other HTML-based desktop efforts have failed."

How many electric car startups failed before Tesla Motors? How many attempts at touch panel mobile devices failed before the iPhone showed the world how it was done and that you didn't need a stylus?

I'm guessing these previous efforts failed because they were too early and the platform wasn't mature yet (notice that I said HTML**5**, not HTML), or they failed in execution.

It's not the idea, or at least not *just* the idea. It's also the timing and the execution. Now is the time, and of course for it to work it would have to be done *well*. A bad implementation of a good idea is still... a bad implementation of a good idea.

"What about all our X apps?"

What about them? Make them work in a backward compatibility mode. Embed tag to the rescue?

"Web UIs suck!"

Again as opposed to what? The hodge-podge of Qt/GTK/whatever bloat that currently dominates the Linux desktop?

As I said: it would have to be done well. Part of that would be picking a UI design paradigm and sticking to it, at least as a de-facto standard. In addition to developing the platform, the project would also need to come up with a set of user interface metaphors and standard design tricks and codify them in the form of a guidelines document.

"The desktop is dead!"

No it's not. It's no longer the hottest growth sector, but it's not dead and it's not going to die unless mobile OSes dramatically change.

When I say desktop I don't necessarily mean "big hulking box." I also mean laptops, and I could see the laptop shrinking down into something the size of a smartphone that connects to external monitors and keyboards when you need it. But it will still be a "desktop."

The people comparing "desktop -> mobile" to "mainframe -> mini -> micro" and arguing that these are similar transition scenarios are engaging in faulty reasoning by analogy. That is, unless something changes.

The people who argued that PCs would "never" displace the mainframe made that argument on the basis of computing power, and they were wrong. But the problem with mobile isn't computing power. My iPad has enough CPU and RAM to do about 75% of what I do with my much beefier laptop, and I can imagine that gap closing within the decade. The problem with mobile is that the operating systems are feudal and enforce a *lot* of limitations that categorically exclude whole classes of work, not to mention imposing their own limits on the kinds of developers and business models that can operate. Changing that would require re-architecting the trust model of mobile OSes.

It would also require Apple and Google to forego massive amounts of app store revenue, which means it's probably not going to happen.

That means the desktop is probably here to stay unless somebody decides to give up their 30% commission on all platform app sales to "defeudalize" their mobile OS.

"Didn't Google already do this with Chromebook?"

No. Chromebooks are designed to be thin clients *only*. That's not what I'm talking about here. I'm talking about a new desktop interface to a machine of any size, thick or thin, and where most of the stuff you use is probably running locally.

Swimming To the Boat: A Desktop Linux Cheat Sheet

I am not volunteering to do this. I'm busy trying to help redecentralize the net and turning a side project in that area into a business. But I'll throw the idea out there for anyone who "gets it" and wants to try.

For the brave and motivated, here is a cheat sheet that I think hits all the major points you'd have to hit to "execute well" on a next-generation Linux desktop:

- **You must solve the app isolation problem.**

  All desktop OSes, both commercial and free, have a fundamentally broken security model. They're multi-user, which is good, but as it turns out user isolation is the less important form of isolation most users really need. Most desktop systems have only one user after all. The isolation they need is **app isolation**.

  Right now on my Mac most of the authors of most of the apps I use basically have my account, if not root on my machine. If they wanted to send me a software update with hidden code to pwn me, they could do so and I'd probably never know.

  Lack of app isolation also creates the problems of "OS rot" and installability (or lack thereof). Windows has this problem *bad*, given that it really has no standard mechanism for package management. Apple solved it about half way with their .app bundle paradigm. Most apps have all their files bundled into a folder-that-acts-like-a-file, which means that there is no "installer" in most cases and uninstalling means just trashing the app. Linux has RPM, DPKG, and friends, which are complicated and ugly hacks to solve this architectural problem.

Look at what Docker and Sandstorm are doing at the architectural level. Of the two Sandstorm is probably the most applicable to this space.

In fact, if Sandstorm wanted to morph from a personal cloud accessable via the web project into a next-generation Linux desktop project, they could. All they'd need is a web-as-desktop interface to their web-as-platform cloud service. They could do this while still also doing the other thing. It'd be like the original X Window System vision: run apps locally or remote, thick client or thin.

- **Don't reinvent the wheel.**

  Use Chromium or Gecko or something. Take one of those and slap it on top of a thin screen and interface device layer. Don't write a whole new HTML5 renderer for God's sake.

If you wanted a MVP (minimum viable product), you could get there quickly by just using X for now. Just have the HTML5 thingy pop up as the only "window." You might even be able to salvage all the video display and input driver work that's been put into X by going in and stripping out all the X cruft you don't need, turning it into nothing more than a driver management layer for your desktop renderer.

- **Optimize later.**

  Get the design right, then fix the performance and native display access stuff.

  I'll tell you one you'll probably need to fix soon though. Right now I don't think any of the web toolkits give IFRAMEs (what you're probably going to use for windows) the same level of subprocess or threading and memory model isolation they give to "tabs." You'll probably have to fix that. Who knows? Chromium or Gecko or WebKit or whatever might even accept your patch if you do it well.

- **User experience user experience user experience user experience user experience user experience...** (picture me jumping around all sweaty like Steve Ballmer in that video... or ok maybe don't picture that...)

Look at OSX, iOS, Android, and Windows. Good artists copy, great artists steal. Take the design paradigms that work and fashion from them something that is simple, intuitive, and a pleasure to use. Make that the de-facto UI standard and document it as a series of guidelines and build everything in your core desktop environment around that.

Don't try to be clever. Good design is usually straightforward and simple and is often very minimal (but not *so* minimal it's missing key features).

- **Punt on the eye candy.**

  Apple's next OSX has translucent toolbars. They're *purdy*. That kind of thing is neato but it's a distant second to making your stuff work and getting basic user experience and UI usability issues nailed.

  Make it work, then make it intuitive and a pleasure to use, then make it fast, and only *then* worry about little designer beanie cap things like gratuitous use of translucency.

Unless, of course, it's very easy. If it's really easy to make something pretty do it. I'm just saying don't let it get in the way of more important priorities the way so many OSS UI efforts have done in the past. Not naming names, but look for anything "skinnable" that's still hard to use.

Oh, and speaking of "skins," punt on that too. UI skins usually suck, even when the big guys do it. Consistency is good. It's not just a matter of aesthetics. It also means that your brain doesn't have to do the work of comprehending a new UI surface for every app. It can remember a set of common idioms -- a visual interface language -- and use them everywhere. This unit economy of interface metaphor is one of the too-often-forgotten cornerstones of good UI design.

- **Ride on Corporate Largesse**

  You need apps, otherwise nobody will use your desktop environment. A bunch of apps are not going to magically appear. Yet since you've built your new desktop environment around the most popular UI platform in the world, there's already a ton of web apps out there in "the cloud" that are ready to be used.

So make them first class citizens.

Make it easy for a user to pop up an instance of Facebook, Google Docs, or some other popular web app in a window as if it were one of their desktop apps. Make interacting with that app easy too, maybe even doing things to seamlessly integrate it (to some extent). Go ahead and ship with the most popular options as defaults in your app menus. (Or who knows... you might be able to net a financial contribution from these companies for doing this. Try!)

No they're not open source and yes I am being a hypocrite since my major interest right now is in *re-decentralizing the Internet!* But you've gotta be a little pragmatic. When a user downloads and installs your environment they're going to need to get work done. Don't make it hard on them and don't push them to inferior alternatives chosen based on ideological purity tests. Let them use the good stuff, and then maybe build new free and libre good stuff later that they can migrate to if they want more control over their own data.

It would be good to let users know whether an app was local or remote, both to remind them that Google Docs isn't "theirs" and for other security and usability reasons. One idea would be to color the window frame differently for a remote app, hinting to the user that this is not running on their local machine. It would also probably mean that certain features wouldn't be available, like local filesystem access.

There's also a lot of corporate largesse in the web UI library space: things like Twitter's Bootstrap and Facebook's React. (It'd probably be a good idea to to survey all the best options for JavaScript UI libraries, pick one, and stick with it.)

- **Make an App Store**

  Make an app store. They're easy. They're convenient. Users like them.

  Since this is free and open, make it possible for people to create their own app stores and users to "subscribe" to them. Just ship with yours as the default.

Someone Please Do This

I'd really love for someone to steal every one of those ideas. While you're at it, feel free to ignore the stuff that's wrong. I'm sure I didn't get everything right.

If such a thing were built, it would *instantly* have not only a minimal set of apps but **the largest community of developers in the world.** They already know the platform and the UI language(s) (JavaScript, ClojureScript, GWT, etc.). They're all ready to build apps, or even just to port the web apps they've already built to run in local containers.

Now you've got a Linux desktop that could really go somewhere. It's got developers, apps, and is written in a language and toolchain everyone already knows.

Adam Ierymenko

Read more posts by this author.

Orange County, California

Share this post