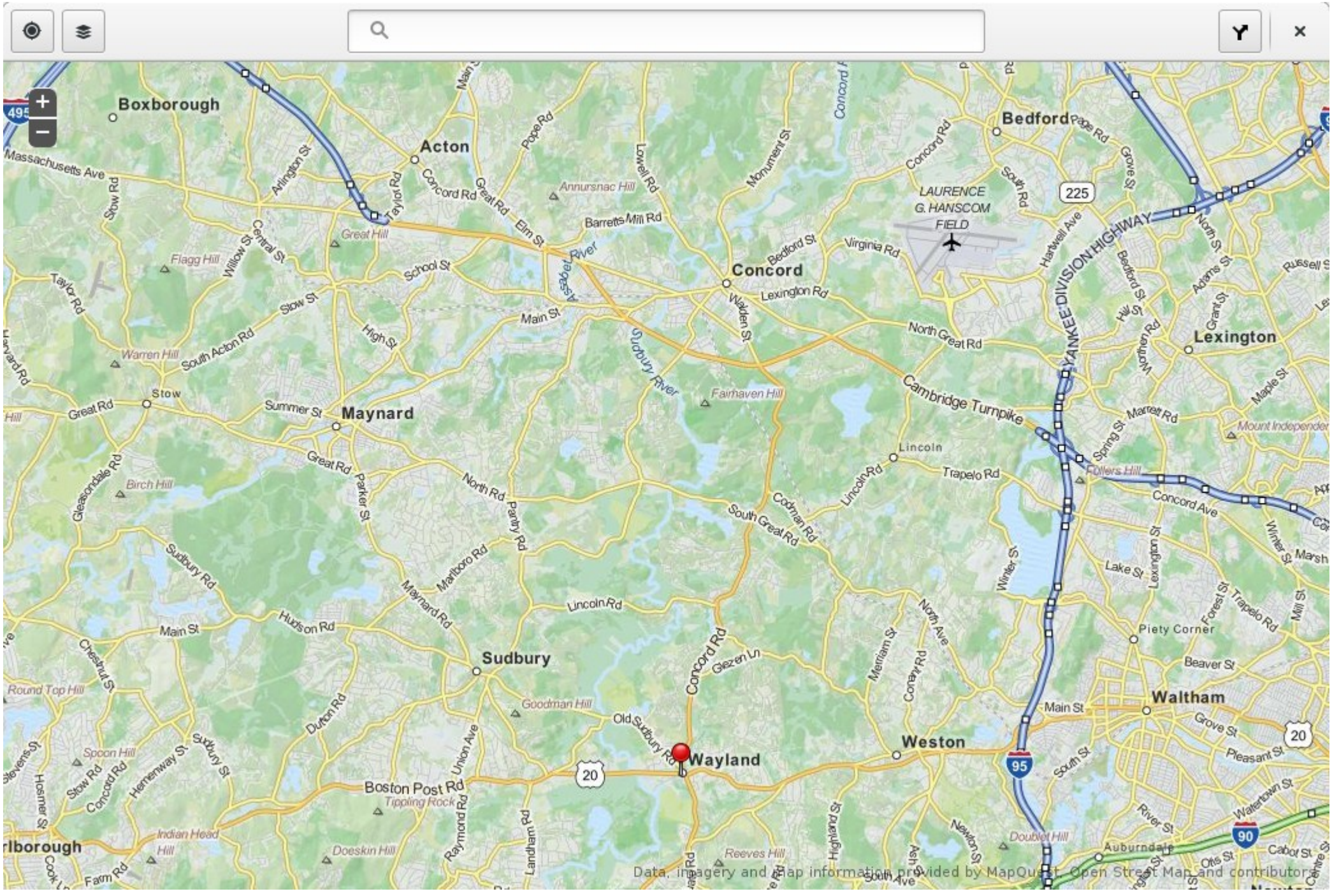
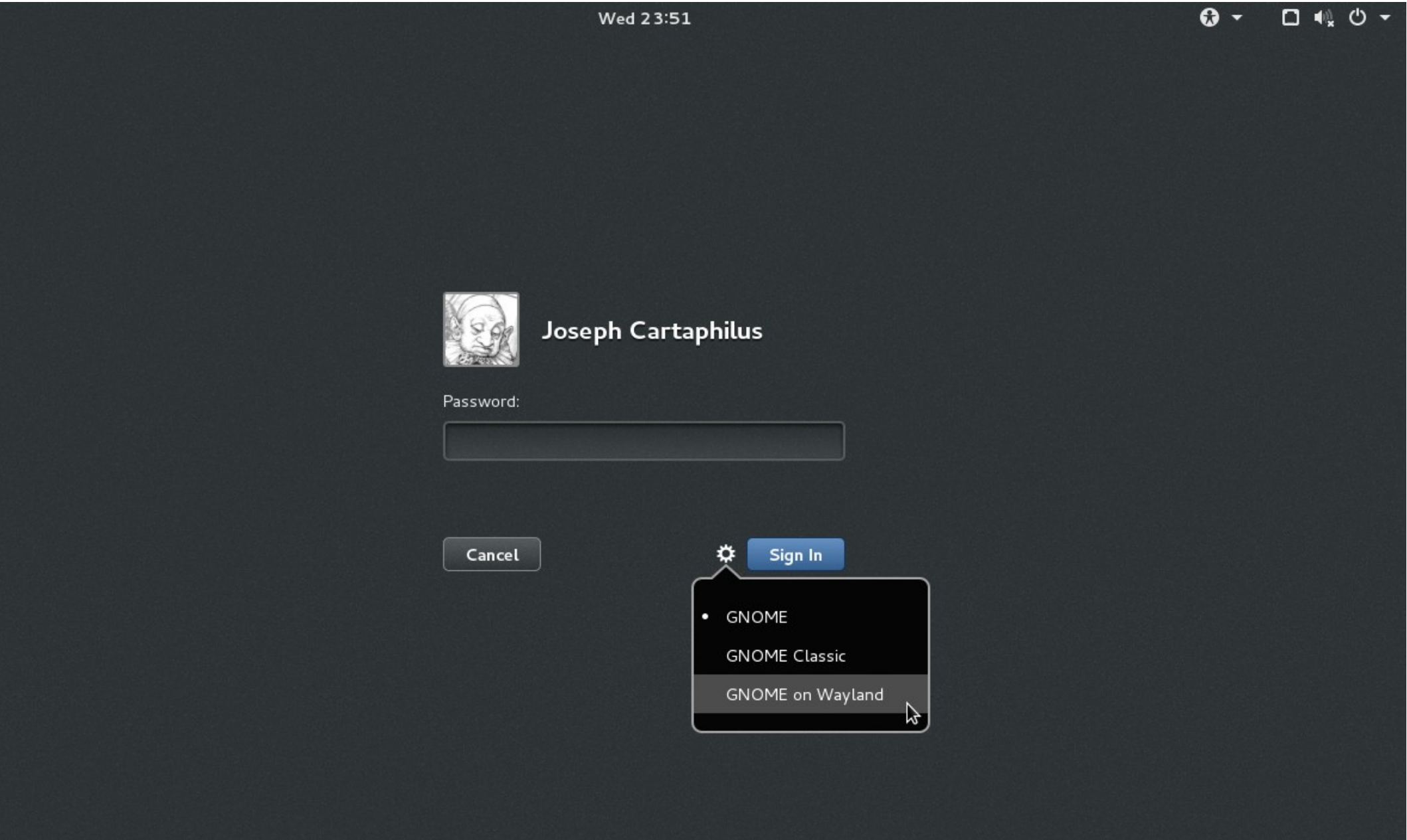
Where's Wayland?

Close to where I live



Also, in Fedora 21



What is Wayland?

What is Wayland?

- A protocol
- Libraries to support protocol implementation
- weston: A reference compositor
- xwayland: A rootless, nested X server

Principles

“Every frame is perfect”

→ Atomic commits

→ Avoid synchronization

Principles

“Client-side everything”

Fonts

Rendering

Nested Windows

Decorations

Principles

"Clients are isolated"

→ No global coordinates

→ No root window

→ No grabs

→ Privileged clients for special purposes

Principles

“Compositing at the core”

→ Merge compositor + display server

Principles

“Multiple interfaces”

wl_seat

wl_surface

wl_data_device

xdg_shell

Advantages

- Clean out cruft from X protocol
- Simplify protocol
- Sandboxing possible
- Possible to implement zero-copy

Concerns

- If the compositor crashes, your session goes down
- Interoperability – will GNOME Wayland clients be able to run under, say, Hawaii ?
- Driver support – the Nvidia driver does not currently support Wayland

History

Earlier attempts:

- Fresco
- GGI
- DirectFB

History

- 2008: KMS, moving mode setting into the kernel
- 2008: Wayland started by krh (while at Red Hat)
- 2010: Initial GTK+ backend
- End of 2012: Wayland 1.0
- Early 2013: GNOME begins Wayland porting
- Fall 2013: 3.10, experimental Wayland support
- Spring 2014: 3.12, more complete support, using xdg-shell

Wayland in GNOME

- GTK+ has a Wayland backend
- Gnome-shell / mutter is a Wayland compositor
- Using libinput
- Functionality moved from gnome-settings-daemon to gnome-shell:
  - Display configuration
  - Keyboard configuration
  - Color calibration
- Gnome-shell exposes D-Bus APIs
  - Display configuration
  - Taking screenshots and screencasts

Current Status

GTK+ backend is pretty solid

Some remaining gaps:

- Drag-and-Drop
- Input configuration
- Wacom support

F21 Goal: day-to-day usable Wayland session

Why does this take so long?

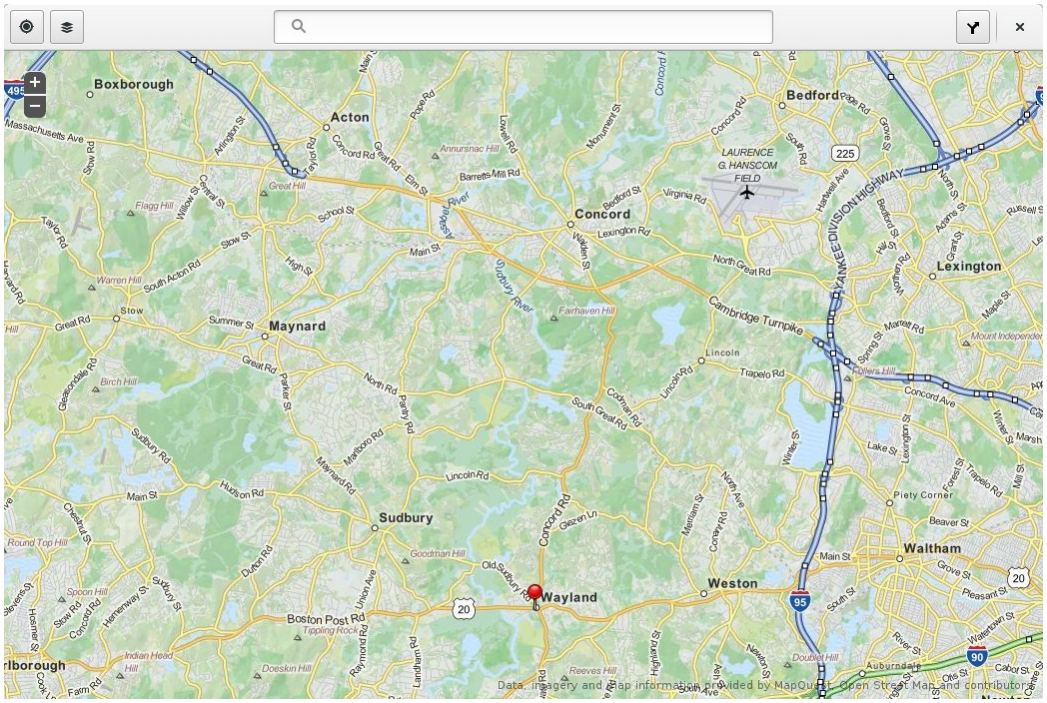
- “No regressions”
- “Features”
- X had 30 years
- Keep X working

Thanks!

Where's Wayland?

On Route 20, outside of Boston. Not to far from where I live, up here in Acton.

Close to where I live



On Route 20, outside Boston. Not too far from where I live, up here in Acton.

Also, in Fedora 21

What is Wayland?

I'll start by giving a brief overview of the Wayland ecosystem and the principles that have guided its develoment

What is Wayland?

- A protocol
- Libraries to support protocol implementation
- weston: A reference compositor
- xwayland: A rootless, nested X server

The Wayland protocol defines what it means to be either a Wayland compositor or a Wayland client. Clients speak the protocol to the compositor over a socket – much like X. The Wayland protocol is specified in an XML file, and Wayland provides a tool that can generate code from the XML description.

Clients normally don't implement the protocol themselves; instead they use toolkits, such as QT or GTK+. The toolkits don't implement the protocol entirely on their own either, they use the wayland libraries that help with this task.

I'll talk more about what Wayland is in GNOME, later.

Principles

"Every frame is perfect"

→ Atomic commits

→ Avoid synchronization

Frame here means the screen contents that get drawn for an individual screen refresh – those typically happen every 16 milliseconds.

This does not mean that every frame is actually perfect, but rather that the protocol is designed in a way that makes it possible to achieve animation, resizing, etc without any glitches.

Atomic commits mean that a client attaches all the ingredients needed for the next frame: buffers, transformations, opaque region, input region, damage, etc to the surface, and then calls commit to put it all in place at the same time. This is also sometimes described as double buffering. The important point is that the compositor always has a full set of up-to-date data for each surface, and never a mixture of old and new.

Principles

“Client-side everything”

Fonts

Rendering

Nested Windows

Decorations

- Fonts were already client-side in X for the longest time (nobody uses core protocol fonts anymore, and fontconfig won over alternative approaches like STSF long ago)
- Rendering is also effectively client-side under X today. Nobody uses the core X protocol to draw circles, polygons or lines
- Nested windows – rarely used in modern toolkits, GTK+ has client-side windows that are implemented in GDK
- Decorations – this is the newest (and still somewhat controversial) functionality to move to the client-side. GTK+ introduced client-side decorations last year in version 3.10. One big advantage of doing decorations client-side is that we don't need any synchronization between client and server rendering anymore, e.g during window resizes.

Principles

"Clients are isolated"

- → No global coordinates
- → No root window
- → No grabs
- → Privileged clients for special purposes

This has always been one of the big problems with X – it is very hard to isolate clients from each other if they can just render on the root window and grab each others keyboard input.

If you've read lwn's report on Jaspers Wayland status update that he gave last week at Guadec in Strasbourg, he had some impressive examples of exploits like a key logger that records your password while you unlock the lock screen.

Examples of operations that are open to every client under X, but require a privileged client under Wayland: screen shots, accessibility tools, input methods

Sandboxing under X basically requires Xnest

Principles

“Compositing at the core”

→Merge compositor + display server

In X, compositing was added as an afterthought to a non-composited windowing system.

Under X, people quickly recognized that having the window manager and compositing manager in separate processes does not make much sense. All X compositors nowadays are 'compositing window manager'.

Merging the compositor,window manager and display server into one simplifies things greatly, since there is no longer a need to have protocol for communicating between these two, and they can share their state, instead of duplicating it in 2 (or even 3) processes.

Principles

“Multiple interfaces”

wl_seat

wl_surface

wl_data_device

xdg_shell

These are just some examples, there are more interfaces in the Wayland protocol (e.g. wl_pointer, wl_keyboard, wl_touch).

The wl_ interfaces listed here are mandatory, the xdg_surface interface is meant for desktop use cases; IVI and embedded use cases don't use it.

The interfaces can be versioned independently, the system is flexible enough to accommodate new development and different user experiences (e.g. IVI systems are generally not using xdg_shell, they have their own shell. Maybe it would make sense to maximize the speedometer, but...

This is similar in many ways to X extensions.

Advantages

- Clean out cruft from X protocol
- Simplify protocol
- Sandboxing possible
- Possible to implement zero-copy

Examples of cruft have been mentioned: core fonts, rendering, multiple screens

Simplification: do away with all the wm/compositor/display server protocol

Jasper showed some convincing examples last week of X applications stealing passwords from the login screen. There have been attempts at fixing these problems in X, but the resulting security extensions are unwieldy and cumbersome. You really want a system that is designed with application isolation as a goal from the start

Zero-copy: client allocated kernel buffer, passes it to the compositor, who then uses it for rendering

Concerns

- If the compositor crashes, your session goes down
- Interoperability – will GNOME Wayland clients be able to run under, say, Hawaii ?
- Driver support – the Nvidia driver does not currently support Wayland

With X, you can have a session limp along with the display server, after the window manager crashed; with Wayland, that is not the case – compositor and display server are one, so if your compositor crashes, your session goes down. This may be an important different for us developers, but the difference for end users is less important – if all your windows suddenly loose your decorations, and you can't switch focus anymore, most 'normal' people would just go for the power button.

Interoperability will hopefully be less of a concern when xdg-surface and xdg-shell interfaces settle down (these have been very actively developed until now

The noveau driver works fine, but many people want to use the proprietary driver. We are working with Nvidia to get them to provide APIs that can be used to implement Wayland compositors

History

Earlier attempts:

- Fresco
- GGI
- DirectFB

This is really the pre-history.
Fresco was an experimental system originally developed by the X consortium in the early 90s. Later renamed to Berlin and then to Warsaw
GGI is the general graphics interface (also early 90s) was an attempt to get graphics into the kernel
Some of these projects had some success, but none came close to replacing X

History

- 2008: KMS, moving mode setting into the kernel
- 2008: Wayland started by krh (while at Red Hat)
- 2010: Initial GTK+ backend
- End of 2012: Wayland 1.0
- Early 2013: GNOME begins Wayland porting
- Fall 2013: 3.10, experimental Wayland support
- Spring 2014: 3.12, more complete support, using xdg-shell

I've looked it up, we had an F10 feature called KernelModesetting

Original motivation for kernel modesetting was better boot experience, without jarring flicker. Plymouth was started as part of the same effort

I remember seeing a rotating flowers demo over Kristians cube wall, back in 2008 or 2009. I also remember discussing client side decorations with him – I was not at all convinced back then

When we got ready for the porting effort, we had a coordinating meeting in Kristians kitchen (March 2013, he was still living in Cambridge at the time). It was a bad snow day, I had to pick up Jasper and it took us almost 2 hours to get into Cambridge (normally a 35 minute ride)

Xdg-shell is an interface that aims to collect expected functionality for a 'desktop toplevel window'

Wayland in GNOME

- GTK+ has a Wayland backend
- Gnome-shell / mutter is a Wayland compositor
- Using libinput
- Functionality moved from gnome-settings-daemon to gnome-shell:
  - Display configuration
  - Keyboard configuration
  - Color calibration
- Gnome-shell exposes D-Bus APIs
  - Display configuration
  - Taking screenshots and screencasts

Switching back to technical details, lets look briefly at the internals of Wayland in GNOME

I've already mentioned that GTK+ has a Wayland backend – that was fairly easy, GTK+ has had a frontend/backend split all along

Gnome-shell and mutter are of course the GNOME compositor/window manager/desktop shell. This code base has a winding history, and did not have a clean frontend / backend separation. A lot more refactoring was necessary here; since we want to keep the X code paths working – starting over from scratch would be much easier.

Libinput is shared with weston, Hans will talk about it

Gnome-settings-daemon used to do a lot of direct interaction with the X server, for things like display, keyboard, color configuration. All of that is moved into gnome-shell.

Current Status

GTK+ backend is pretty solid

Some remaining gaps:

- Drag-and-Drop
- Input configuration
- Wacom support

F21 Goal: day-to-day usable Wayland session

Drag-and-drop will not be in place for f21; input has gotten a lot better – acceleration and palm detection are in libinput now; we don't have configuration for it yet. Wacom support is being added to libinput as well, but I am not 100% sure if it is realistic to get that into F21 at this point. Go to Hans de Goede's talk about the 'Wayland Input Status' this afternoon to learn more !

Why does this take so long?

- “No regressions”
- “Features”
- X had 30 years
- Keep X working

We have a polished and well-working desktop, we don't want to endanger that. If the Wayland port is a success, users will not notice that they are not running X.

We can't stop adding features and doing new development – things like hi-dpi support, touch, animations, etc are all happening in parallel to Wayland porting.

In some areas X protocol and api are mixed up with GTK+ API, and need to be detangled

Moving a lot of code around: things where previously gnome-settings-daemon would talk to the X server (eg to configure monitors), are now moved into gnome-shell, which offers a d-bus api.

Keyboard handling also moves entirely into the compositor.

Thanks!