Ideas on looking for a new Driver Model

Egbert Eich, SUSE Labs, SUSE Linux Products GmbH
eich@suse.com
1. Current Problems

Current driver model designed in late ’90 by XFree86

First Release in 2000

Design goal: let the driver control any aspects of the configuration.
- Move common code to optional helper functions.
- Inherited limitations from mi and ?fb layers and core protocol.

Supported hardware of the late 90’s:
- Single analog display output with
depth 8, 16, 24
- 2D acceleration

Feature: was able to operate several independent graphics chips.

Later added:
- XV video scaler support
- Support for multiple (2) display outputs with independent CRT controller
2. Limitations and Problems

- Multiple output devices per chip

- Multiple types of output devices:
  - Analog (VGA style)
  - DVI (digital)
  - Video bridges (TV)

- Mode selection too simplistic: drivers do their own

- Missing hot plug support for output devices

- Limited support for switching output channels on the fly
2. Limitations and Problems [cont.]

- Many features not configurable ‘on the fly’.

- In general: ‘Code is cheap!’ - register bit banging is not!

- X is not alone: other software needs mode setting, too!
  - Text console
  - standalone DRI
  - Xgl
2. Limitations and Problems [cont.]

- No support for hotplugging graphics devices bootstrap procedure makes it impossible to add devices on the fly:
  - Probe()
  - PreInit()
  - ScreenInit()

- 2D accel model not suitable for RENDER

- Modern video drivers consist of drivers for different components
  - no clear driver internal interfaces
  - Migration to new driver model difficult
3. A Model for the Future

☐ Take driver out of the Xserver
  ◦ create a separate video output driver module/project
  ◦ provide library/daemon: daemon record information about the driver/HW state, library to provide interface to applications (like Xserver), perform that change hw state (mode etc), provide a low level 2D acceleration, video scaler etc. functionality.

☐ Let Xserver take ’passive role’:
  ◦ Mode selection happens between a UI and the driver. Pass video mode information to Xserver to adjust itself to underlying mode.

☐ Create a thin DDX that to interface with driver API to take mode information.
4. How to get there?

- Not possible in a single step

- Preparations to be made in existing structure.

Phase 1

- Look thru DDX: migrate HW related code (bus, address range mapping into a separate layer.)
4. How to get there? [cont.]

- As a driver maintainer
  - Identify different driver components (mode setting, 2D acceleration, DRI, video ...) and understand their interrelations.
  - Identify which parts communicate with the other layers of X (PreInit(), ScreenInit()) Video, XAA, DRI ...

- Separate X specific parts functionally from driver internal code.

- Create well defined driver internal interfaces

- Modify drivers to implement these interfaces

- Identify driver components that can be share between drivers (RAMDAC, Video and FP bridges etc)
4. How to get there? [cont.]

- Provides opportunity to clean up drivers and 'discover' 'junk code'

- Easy to do for drivers that are currently maintained.

- What do we do about unmaintained drivers?
  - If they are simple enough it may be easy to do.
  - Possible to do without special knowledge if hardware for testing is available.

- We may loose some drivers.
4. How to get there? [cont.]

Phase 2

- Define external API for external library

- Add interfaces to software to work with this new DDX:
  For Xserver: prepare DDX to work with this

- Migrate selected drivers

- Test software (new Xserver DDX) with selected drivers
4. How to get there? [cont.]

Phase 3

- Port over the remaining drivers