Replacing AGL WindowManager

Daniel Stone
daniels@collabora.com
Hi, I'm Daniel

Graphics lead at Collabora
Open-source consultancy est. 2005
Wayland core developer
Outline and agenda
Outline and agenda

- Current WindowManager and HomeScreen APIs
- Comparison with Wayland protocols
- Plan to merge WindowManager into compositor
- Open questions and support
WindowManager API
WindowManager overview

- WindowManager system service allows external process to control window positioning and policy
- Acts as 'proxy' between app and Wayland server
- Provides mapping between AGL app and Wayland IVI protocol
- Applies OEM WM/UI policy and tells Wayland server what to display
- WM is a critical system process: if it crashes, no new content will be shown!
Current WindowManager design

Application (e.g. Navi)

Binder

Wayland

Window Manager service

Wayland (custom)

HMI controller module (window management)

ID agent module (policy DB)

libweston (core Wayland)

IVI shell (window management)

Weston core (libweston wrapper)

Open First
Issues with WindowManager

- WindowManager uses separate protocol (Binder)
- Synchronisation between two protocols required
  - create surface with WM surface, get surface ID, give surface ID to Wayland server
- Custom integration means it is more difficult to write AGL apps!
- Integration of app FW & Wayland main loops required
Issues with WindowManager

• WM API duplicates many features available in Wayland protocol
• Possible for app/toolkit to receive conflicting messages
  - example: active state in xdg_toplevel Wayland interface plus WM activated signal
  - WM does not control Wayland server: cannot synchronise
  - application and toolkit may 'fight' on conflict
Issues with WindowManager

• Many duplicated APIs:
  • WM setRole vs. XDG set_app_id
  • WM area vs. XDG configure
  • WM activateWindow vs. XDG activated
  • WM syncDraw vs. wl_surface frame
  • WM focus vs. wl_touch focus
  • WM Screen vs. wl_output/xdg_output
  • WM setRenderOrder vs. wl_subcompositor
Issues with WindowManager

- Most duplicated APIs are deficient compared to Wayland
- Wayland APIs allow for dynamic and hotplug situations
- WindowManager APIs are not atomic: does not allow to wait until all UI ready, reconfigure & show together
Issues with WindowManager

- Unclear definition of multiple window manager co-operation (remoting/multi-ECU case)
- Dynamic output and streaming management not possible
- **Security**: policy DB can be overridden by client app!
- More difficult to debug: multiple processes, multiple protocols
Suggested changes
Summary of suggested changes

- Remove WindowManager API: replace with Wayland core
- Replace HomeScreen API with integrated Wayland 'shell' design
  - compositor plugin: policy (e.g. restriction), integrated with WM
  - client process: UI rendering (e.g. home screen)
  - possible to launch HS service securely!
Summary of suggested changes

- Implement new `agl_xdg_extension` Wayland protocol
- Replace HomeScreen API with integrated Wayland 'shell' design
  - compositor plugin: policy (e.g. restriction), integrated with WM
  - client process: UI rendering (e.g. home screen)
- possible to launch HS service securely!
WindowManager changes #1

- Application (e.g. Navi)
  - Binder
  - Wayland

- libweston (core Wayland)

- Windows Manager service
- OEM HMI customisation
- Helper library
- AGL Reference Compositor

Open First
WindowManager changes #2

Application (e.g. Navi) → Wayland → libweston (core Wayland)

- App framework protocol
- OEM HMI customisation
- Helper library
- AGL Reference Compositor

Open First
Layering of Wayland protocols
Open questions
Open questions: window manager

- Is there an example case for split-application UI?
  - e.g. Navi UI provided by process #1, Navi content provided by process #2
  - some extension of XDG Wayland protocols required to realise this
  - upstream community should be happy to receive change

- Is there an example implementation of window manager 'policy DB'?
  - documented in HMI/WM spec but not present in code
Open questions: home screen

• How should the home screen API be designed?
• Current HomeScreen and WindowManager APIs are very separate, but depend on each other
• What home screen implementations do we have today?
• Who could help with porting current AGL home screen architecture to new design?
Open questions: home screen

• HomeScreen architecture appears to duplicate core app-framework functionality
• Should launching applications and services be part of core App FW? (Launching can be required for other uses.)
• Home screen transitions must take care of global vehicle state (stop video when gear engaged)
Open questions: timeline

• Deprecating WindowManager/HomeScreen apps requires change in AGL UCB core, AGL demo UI/HS, AGL demo apps, ISV apps …

• What timeline is realistic to make these changes?

• Do OEMs have a requirement for old APIs to remain?
  - implementing two services together is technically difficult
Open questions: support

• Changing WM/HS API requires support from UI and app developers, OEMs shipping AGL
  - do we have this support?

• Changing compositor requires support from reference-platform BSP
Next steps

- Continue gathering use cases from OEMs and ISVs
- Publish document listing AGL required Wayland extensions
  - XDG shell for clients, Wayland alpha-compositing protocol for blending, etc
- Develop homescreen UI architecture with input from OEMs
- Port reference AGL apps (homescreen, mixer, Navi, etc) to new architecture
Thankyou!

daniels@collabora.com