[WIP] [GSHA] Virtual Display Tech Brief

Summary

Virtual Display provides a display to the system which is not necessarily linked to the real display hardware. Applications and middleware, e.g. system compositor, should be able to use this display as usual. The concrete implementation has to transfer the content of this virtual display and present this on the real display hardware. The Virtual Display concept can be considered a subcategory of Surface Sharing.

Key Characteristics

- Simple to use because it abstracts the details of display implementation for the middleware
- Doesn't provide a fine granular control over the display content but it is still good enough for a lot of use-cases
- Entire display content is handled instead of applications content

Description

Virtual Display describes a concept which can be used to realize distributed HMI use-cases. Important characteristic of the Virtual Display concept is that the entire display content is transferred instead of transferring content from dedicated applications. The system which provides the content should be presented with a display which act likes a physical display, but is not necessarily linked to a physical display, so the middleware and applications can use it as usual. Such a display can be called a Virtual Display. The implementation of Virtual Display on the producer system should be generic enough to look like the normal display and should take care of the transferring the display content to another HMI unit or another system. This basically means a final graphical surface needs to be transferred, therefore it can be considered as a subcategory of Surface Sharing. On the receiver site the content can be handled with more flexibility. It could be directly used as content for a physical display, it could be mapped to one physical layer of the composition hardware or used as part of a composition where it is combined with other available content local to the receiver. This flexibility makes the definition and the separation between different technologies a little blurred. An important characteristic of the Virtual Display concept is that we are handling the entire display content on the producer instead of handling content from dedicated applications.

Example: virtual display in Weston

Open source wayland compositor-Weston provides an example of Virtual Display implementation [1]. Weston can be configured to create a Virtual Display which is not linked to any physical display. Properties like resolution, timing, display name, IP-address and port can be defined in weston.ini file. From this configuration Weston will create a virtual display "area" and all content which will appear in this area will be encoded to M-JPEG video compression format and transmitted to the configured IP address. The current Weston implementation uses corresponding GStreamer plugins for this. The assignment to virtual display will happen in differently depending on the Weston shell that is used. For example, if ivi-shell is used the new virtual display can be identified by the configured display name and the corresponding wayland protocol can be used to put application content on this display. In case of xdg-shell the user can just drag the application with the mice to the corresponding area.

The Weston example doesn't provide any code for the receiver system. In the first reference implementation examples the gstreamer pipeline provided is used to decode and present the content provided by Weston and therefore should run on major Linux distributions. For a production environment it should be fairly easy to write an application to receive the content by using available API's and frameworks on the target system.

Example: virtual display in Android

Android OS has support for creation of multiple virtual displays. The number of virtual displays is limited only by hardware processing capability. Typically Virtual display is used in Android to render content on a remote display. Android provides APIs to create Virtual Displays of any given width /height, dpi and associates a surface with every virtual display created. Android provides interfaces to access the rendered content on that surface, those interfaces can be used from any system app, which needs special permissions. The content can be either encoded as a video of any format supported by the platform or captured as a raw RGB buffer.

Any app can use a virtual display in two ways. It can mirror it's main view or create a different content, second view, and provide this to the virtual display. From Android O onward, apps can be launched on any display (Virtual or physical) dynamically. Android also provides APIs to inject user inputs such as Touch/ keys etc to a specific Virtual Display so that the app running on that display can be controlled. Even 3rd party apps can be launched on virtual displays.

The most prominent use of Virtual Display for content sharing is the implementation of Android Auto Projection. The whole Android Auto projection content is rendered onto a Virtual Display. It is then encoded as a video and sent over USB / WiFi to the head unit.

When more than one display is active (Physical or Virtual), Android imposes certain limitations (for e.g. No support for multiple instance of same app on different displays), they can be overcome by customizing Android framework.

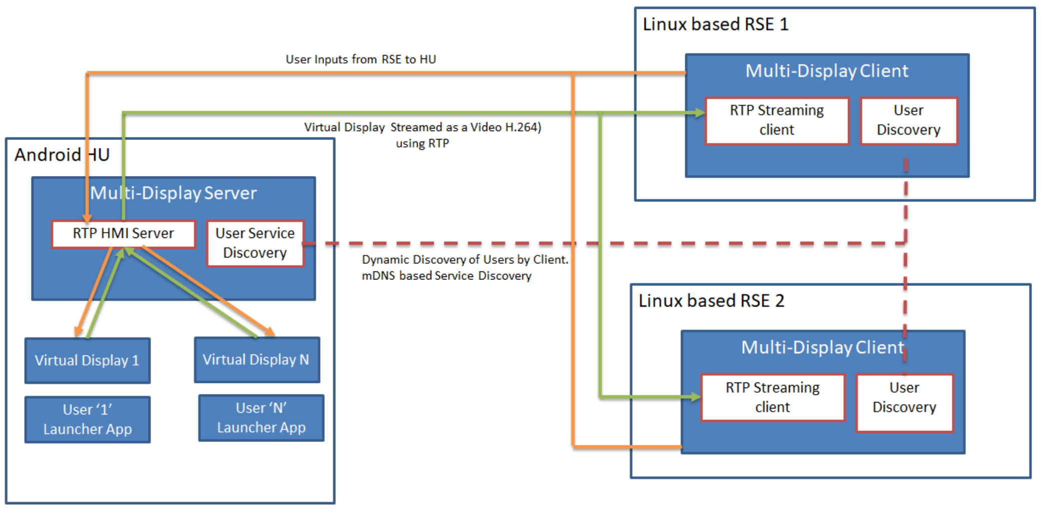
AllGo Case Study:

AllGo had done a demo to see how applications from an Android based HU can be run on remote thin client RSEs connected over ethernet. The key features that was targeted for the demo was

- Allow users to launch and control app on any of the displays
- Allow multiple users to access and control different apps simultaneously.
- Switch apps from one display to another seamlessly depending upon where they are looking / sitting.

Software Overview

The diagram below gives a high level architecture of how the rendering and control of applications on RSE work.



For every unique user, a dedicated Virtual Display and an associated custom launcher is created on the Head Unit.

Multi-display Client running on the RSEs, discover the users using ZeroConf and choose a specific user. Once a specific user is selected, the associated custom launcher for the user is launched on its associated virtual display, which is then encoded as a video and sent out to the specific RSE for rendering, The client also sends back the user input (like touch / keys) to the server, which is injected to the associated virtual display.

In our demo, two users were able to independently launch apps on the two RSEs.

When the passenger wants to switch to another RSE (say he changed seats), he needs to simply select the same user from the other RSE. The server then automatically switches the streaming the associated virtual display content to the newly selected one.

The number of RSEs in the system can be any number. It does not need to match the number of unique users. The number of unique user can also be configured. A video of the demo ca be seen here.

Conclusion

...

References

{1} https://github.com/wayland-project/weston/tree/master/remoting

- [ ] surface sharing tech brief

Document version 2:

GENIVI_ADIT_Vir..._genivi_v2.docx