Technology overview & current options

Collecting information

In-Car

Across car ECUs, networked

e.g. Cluster to IVI, Head-unit to RSE, etc.

- Linux to Linux
  - Surface sharing
    - Wayland / Weston(or other compositor) / Waltham
      + graphics transfer, e.g. h.264 encoded. gstreamer buffers.
  - RAMSES

- Android to/from Linux
  - Surface sharing
    - Currently: Ad-hoc project-specific solutions + remoting within implementation below graphics HAL
    - Wayland to Surfaceflinger API comparison/mapping - could open up for Waltham like solution.
    - RAMSES runs on Android. Both directions to/from have been implemented as prototype.

- RTOS (like Integrity) to X
  - RAMSES supports Integrity
  - Qt support on Integrity. Commercial HMI tools....

- QNX
  - No known shared standards yet - reaching out to team for info

- Other HMI-Tools/frameworks
  - No known shared standards yet.

- Display & GPU
  - Relevant if running on same hardware (virtualization)
  - Upper layers should see this as a normal display or similar abstraction
  - Virtio, covering some graphics(?)
    - virtio-gpu, see this Collabora blog, needs input on functionality and support for automotive h/w
  - Survey options - Hardware that supports virtualization and Hypervisor options

CE-Device

Device to Car (phone apps in dash)

- Surface Sharing
  - Previous attempt at system-level solutions are scattered
    - Not ubiquitous adoption, e.g. Mirrorlink
    - Device-specific solutions, e.g. Android Auto "projection" (mirroring) and Apple CarPlay(TM)
  - Android "projection mode" uncertain, due to Google push to instead do embedded Android Auto

- API remoting (non-graphical and graphical)
  - Smartdevicelink, started with non-graphical car-to-device remoting. Now including device-to-car graphics transfer TODO details
  - Various specific solutions (category unknown)
    - Baidu, MySpin,
    - TODO: GA look at previous analysis.

- Display & GPU Sharing - N/A

Car to Device (car providing an HMI displayed on device)

- API remoting
  - Command-and-control only
    - Because of the convention to update phone apps frequently, an option is to make sure app is up to date with in-car HMI. Distributed HMI can then be simulated by basic API remoting.
  - RAMSES - a renderer on the (Android) device.

- Shared State, Independent Rendering
  - Phone app to/from car.

- Surface Sharing
  - Possibly limited need if synced local HMI + API is used. Still, some still graphics needed, e.g. album art,
  - Video use cases exist (view in-car camera) need solution but it's a straight video transfer, not really distributed HMI

- Display & GPU Sharing - N/A

Car ECU to Cloud?

- Use cases?
  - Screen shot during test/debugging, etc. (a bit out of scope)
Mostly similar to Shared-State solutions

Cloud to Car ECU?
- Anything beyond obvious things? (Album art for media services, all that already built into those protocols)
- Future: Cloud could specify the HMI somehow? Consider modern cloud-driven computer games.

GPU sharing (virtualization)
- Work to do – will feed into Hypervisor Workshop discussion
- Different technologies are in use from entirely s/w based systems, through to hybrids utilising h/w virtualisation support in the GPU or SoC.
- It would be useful to have a high level taxonomy of the options available to help guide technology discussions, e.g. does approach A fit the taxonomy.
- H/w virtualisation
  - IMG PowerVR hardware virtualisation white paper (starting point - pertinent points need to be extracted, e.g. multiple GPU are presented)
  - Renesas R-Car
    - Renesas R-Car Virtualisation Software Package provides documentation and example source code for hypervisor vendors to more easily develop solutions for R-Car SoCs.
    - R-Car Gen 3 has h/w support for multiple command input ports that can be directly connected to each OS. This means a HV is not needed to arbitrate the GPU command flow in an multiple OS environment.
    - Combination of GPU OS id and a dedicated PMMU provides memory protection between OSs.

<table>
<thead>
<tr>
<th>Technologies</th>
<th>Shared State, Independent Rendering</th>
<th>GPU sharing</th>
<th>Display sharing</th>
<th>Surface Sharing Examples: Waltham (gst-pipeline)</th>
<th>API remoting Examples: RAMSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>HV Linux non- Linux (Integrity, QNX, etc)</td>
<td>1) RCarH3, Integrity-HV (Linux/Linux/Integrity, tbd)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2) RCarH3, OpenSynergy COGOS HV (Linux/Linux/RTOS, Linux/Android)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3) RCarH3, SYSGO PikeOS (Linux/PikeOS on H2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4) R-Car H3 QNX (HV) cluster</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5) Qt Neptune 3 UI, multi-screen demo (code, video). Cluster+HVI combination. Since this demo is GPU sharing, the Neptune source code has no particular graphics-sharing code handled by the HV. Noteworthy however, there’s some use of Qt Remote Objects for remote-control</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1) Weston virtual-display and gstreamer plugin: Existing (RCarH3), open source (PoC video *)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2) Renesas Connected Cockpit Truck Demo (R-Car H3, OS: Integrity/Android Oreo, HMI: Altia cluster + custom Android HMI. Overview, Video **)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3) Renesas CANVAS Demo (R-Car H3, Integrity/Linux (video *)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux Linux</td>
<td>(theoretically possible - discussion)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1) R-Car Gen 3 SoCs have h/w features (multiple command input ports, IPMMU and OS id) that allow the separation of OSs in the GPU. An HV is not required to arbitrate the GPU command flow.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1) Weston virtual-display and gstreamer plugin: Existing (RCarH3), open source (PoC video *) see above</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1) Waltham: Existing (RCarH3/M3), open source (implementations: ADIT, Collabora), current implementation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HV Linux HV Linux</td>
<td>1) RCarH3, OpenSynergy COGOS HV (Linux/Linux)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2) RCarH3, Epam Xen HV (Linux/Linux/Android)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux Android</td>
<td>Demonstration shown at Munich AMM. Video link</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AllGo Multi-Display demonstration (video) see minutes 26 July for details</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(*1) Virtual display sharing technology used to transfer map navigation onto cluster seen at 1:40 to 2:10 in video.
(*2) Overviews of specific individual components can be found here.
(*3) Virtual displays can be transferred and transformed between physical displays and OS, e.g. from linux into integrity based cluster.

Display Sharing Categories

Categories are “formally” defined on the main project page, so this is more of a reminder. But here we add also example technologies and some more details.
• **GPU sharing**
  The GPU can be used from multiple operating systems, so it is shared. Concurrent access to the physical GPU has to be controlled by the hypervisor, hardware or other means which are implementation specific.
  - Studied Technologies:
    - *<The discussion of GPU sharing is best held in the Hypervisor Project, where most of the relevant experts are>*

• **Display sharing**
  The physical display can be shared across multiple operating systems. HW compositor unit composites final display buffer from HW Layers of each OS. This requires virtualization of the display controller hardware.
  - Studied Technologies:
    - *Since this is mostly hardware specific we have not (yet) looked at it much.*
    - *Layer-Management is a related software abstraction, but intended a for single-system*

• **Surface sharing**
  Operating systems exchange graphical (bitmap) content. Then, each OS has full flexibility to use this content. In some cases, the compositor API is made available remotely, e.g. Wayland/Waltham.
  - Studied technologies:
    - *Waltham*
    - Consider: SmartDeviceLink (graphics sharing for Navigation and similar apps)
    - Qt Remote Objects are not designed specifically for graphics - more for generically mirroring any data object. It was noted however, that if the object is a QImage, the feature should in fact synchronize bitmap data. (The feature is unlikely to be optimized for graphics transfer),
    - Renesas Weston compositor virtual-display and gst-recorder plugin...
      - *Renesas Weston compositor virtual-display and gst-recorder plugin*
        - virtual-display allows the creation of a composited display window the correct size which can then be remoted
        - gst-recorder GStreamer plugin in Weston Compositor encodes the virtual display using H.264 and transmits. Receiver uses GStreamer to receive and decode the stream, which can then be composited as required.
      - **Pros:**
        - *No modification required to Weston on receiver side.*
        - *If layer control is required the decoded stream can be transformed as required on receiver side in gfx framework of your choosing.*
        - *Video encoding minimises network data and bandwidth*
        - *Virtual display means display size can be made only as large as needed.*
        - *Technology has multiple applications, e.g. has been used in Waltham implementation for gfx data transfer.*
      - **Cons:**
        - *Delay of about 2-3 vsync to encode, transmit, receive, decode and composite the stream.*
        - *Large display containing a lot of animated objects will impact network bandwidth*
    - *R-Car M3 PoC in AGL demonstrator. Open source: links TBA*
      - *Two M3 ECUs. Navigation and mapview app encodes mapview image using gst-recorder and transfers via ethernet (udp). Receiver mapview image, decodes the data and displays moving map image between dials of cluster display.*
      - *See the slides presented at the ALS 2018 talk “Remote Access and Output Sharing Between Multiple ECUs for Automotive”*

• **API Remoting**
  Transfer API calls, corresponding to “drawing commands”, or other abstract graphics representation from one ECU to another, to be executed on the GPU of the receiving ECU.
  - Studied technologies:
    - RAMSES
    - Qt WebGL streaming is somewhat similar. The sending side has an OpenGL representation (typically created using Qt abstractions), which is transferred to a WebGL capable renderer. ([blog post](https://example.com/blog_post), [blog #2 - cinematic experience example](https://example.com/blog_post2), feature merged into Qt 5.10)
    - Note: SmartDeviceLink. While SDL design is generally about making remote API calls, the graphics exchange is surface sharing, as far as we know. A kind of API-remoting for operations as opposed to graphics

• **Shared state, independent rendering**
  Each system has independent graphics systems and bitmap information. The systems only synchronize their internal state and exchange abstract data. Based on this shared data, each system independently render graphics to make it appear like they are showing the same or related graphics.
  - Studied technologies/examples:
    - Harman presentation at AMM, also in Tech Brief
    - Qt Remote Objects provide a generic way to mirror/synchronize internal state. If both sides are using Qt, it would be a natural tool.