Proposal for AGL Sound management

AGL F2F meeting @ San Jose
July 2017

Naohiro Nishiguchi
nnishiguchi@jp.adit-jv.com

Advanced Driver Information Technology
Who am I?

- I have more than 10 years experiences in IVI product development for Audio domain
  - Application layer
  - Middleware (e.g. Media playback engine, Beep playback engine)
  - Audio routing management to adapt/configure real customer projects
  - ALSA

- I am audio experts from low-level to high-level application layer covered, and had been working more on customer project.

- Currently, I started more on platform, especially audio routing management framework to be more easily applicable to actual customer projects.
Use cases in automotive
System architecture
Requirement for automotive sound management
Comparison between Advanced ALSA audio agent and Genivi Audio manager
Proposal
Sound manager PoC
Example Use Cases

Active Source Change
- Driver is listening to Mediaplayer in the car.
- Incoming phone call and answer the phone
- IVI system automatically pause Mediaplayer, and then play Phone sound.
- After Phone call is completed, IVI system automatically resume play Mediaplyer

Last Audio (Persistence)
- Driver is listening to Radio in the car.
- Driver turns off/on the engine.
- IVI system automatically start playing Radio.

Mixing & Volume Attenuation
- Driver is listening to Radio in the car.
- Car detect moving objects when parking.
- IVI system automatically mute(or reduce) the volume of Radio, and then start playing Alarm using another speaker.
- After Alarm is completed, IVI system automatically recover the volume level of Radio.

Implicit policy management is required
There are several Hardware architectures as presented by MAZDA in ALS 2017

- **Compact**
  - There are several communication protocol between SoC and others.
  - Sound devices are connected to SoC **directly**.
  - All audio streaming are **visible** and application **can control** audio streaming and volume directly.
    - SoC is master of volume

- **Luxury**
  - There are several communication protocol between SoC and others.
  - Some sound devices are **NOT** connected to SoC.
  - Some audio streaming are **INVISIBLE** and application **can NOT** control audio streaming and volume. e.g. Meter, Camera to AMP
    - External ECU is master of volume

**Several types of hardware have to be supported**
Requirement for automotive sound management

## Requirement mapping

**Use cases & system architecture**

- Active Source Change
- Last Audio
- Mixing & Volume attenuation
- Visible source / sink
- Visible streaming
- Volume master is SoC
- Invisible source / sink
- Invisible streaming
- Volume master is external
- Several communication methods

**Requirements**

- Shall be able to apply business logic implicitly
- Shall have a persistency for sound source and volume.
- Shall know all sound sources and sinks in system
- Shall manage sound route regardless of source and sink location.
- Shall manage volume regardless of the master of volume location.
- Shall be able to be used by applications easily.
- Shall be extensible (independent from specific routing mechanism to control)
- Shall be reliable
## Requirement coverage

<table>
<thead>
<tr>
<th>Requirements</th>
<th>AAAA</th>
<th>GAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shall be able to apply business logic implicitly</td>
<td>??? The feasibility of “Pause” is unclear.</td>
<td>FEASILBE</td>
</tr>
<tr>
<td>Shall have a persistency for sound source and volume.</td>
<td>FEASILBE Cooperation with other components is required</td>
<td>FEASILBE</td>
</tr>
<tr>
<td>Shall know all sound sources and sinks in system</td>
<td>FEASILBE By dedicated binder</td>
<td>FEASILBE  By registration mechanism</td>
</tr>
<tr>
<td>Shall manage sound route regardless of source and sink location.</td>
<td>FEASILBE Dedicated binding development is required for external</td>
<td>FEASILBE Dedicated plugin development is required for external</td>
</tr>
<tr>
<td>Shall manage volume regardless of the master of volume location.</td>
<td>??? How to abstract external source?</td>
<td>FEASILBE Dedicated plugin development is required for external</td>
</tr>
<tr>
<td>Shall be able to be used by applications easily.</td>
<td>It is general in Linux</td>
<td>It is OSS.</td>
</tr>
<tr>
<td>Shall be extensible (independent from specific routing mechanism to control)</td>
<td>Dedicated binding is required</td>
<td>Dedicated plug-in is required</td>
</tr>
<tr>
<td>Shall be reliable</td>
<td>Developing</td>
<td>Already in the market</td>
</tr>
</tbody>
</table>

There is no big deference between AAAA and GAM in requirement point of view
Comparison between Advanced ALSA audio agent and Genivi Audio manager

Architecture point of view

User Interface
- HTML5, QT, GTK, ...

Application Binder
- Application Business Logic
- Imported AGL audio API (ws-client/unix/usr/share/wsock/audio)

AGL Audio Binder
- Exported AGL Audio API (High Level Binding)
- HAL (Hardware Abstraction Binding)
- Optional Unicorns Binding
- Alsa Low Level Binding
- Sound Controls
- Lib asound (alsa-lib)
- PCM
- Sound Card
- MOST/Eth-AV

Daemon
- Command plugin
- Controller plugin
- Routing plugin ALSA
- Routing plugin MOST/Eth-AV
- lib asound (alsa-lib)
- Sound card
- MOST / Eth-AV

GAM
Comparison between Advanced ALSA audio agent and Genivi Audio manager

**Architecture point of view**

<table>
<thead>
<tr>
<th>Interface</th>
<th>High level Binding</th>
<th>Command plugin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy</td>
<td>ALSA UCM + ???</td>
<td>Controller plugin</td>
</tr>
<tr>
<td>Adaptation</td>
<td>HAL, Optional Unicens Binding</td>
<td>Routing plugin ALSA and MOST/Eth-AV</td>
</tr>
<tr>
<td></td>
<td>Dedicated plug-in development is required for external</td>
<td>Dedicated plug-in development is required for external</td>
</tr>
<tr>
<td>Play streaming</td>
<td>App -&gt; ALSA</td>
<td>App -&gt; ALSA</td>
</tr>
</tbody>
</table>

There is no big deference in architecture point of view. Complexity of both is not different.
We can realize requirements whichever we choose AAAA or GAM
- There is no big difference between AAAA and GAM in requirement and architecture point of view.
- Realization with AAAA has already been realized by GAM.

I suggest using GAM for sound policy management because:
- In order to develop sound management of AGL earlier, it is better to use existing software.
  - GAM is already integrated in AGL
  - Interface and sequence of GAM are already specified and published.
  - GAM has been already evaluated in the market.
- We should focus on integrating new technologies (e.g. UNICENS, CAN communication or something) to sound management of AGL, rather than implementing new back end of sound management.

We already started PoC development to apply GAM to Binder
PoC

- For CES2018, we are developing sound manager PoC with TOYOTA.

Schedule

**AGL HMI FW development – TMC CY2017 roadmap**

<table>
<thead>
<tr>
<th>AGL activity</th>
<th>2017 - 1st Half</th>
<th>2017 - 2nd Half</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window Manager Development</td>
<td>15 May HMI-FW</td>
<td>1/Jan CES2018</td>
</tr>
<tr>
<td>(Mentor)</td>
<td>Draft 1 Jun</td>
<td></td>
</tr>
<tr>
<td></td>
<td>27 June</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 July</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11-13 July AGL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F2F US</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 Aug</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 Sep</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15 Sep</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1/Jan CES2018</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 Oct</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 Nov</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 Dec</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sound Manager PoC Development</td>
<td>WM1. Decide</td>
<td>WM5. Complete</td>
</tr>
<tr>
<td>(Mentor)</td>
<td>Functionality</td>
<td>WM (Server+Client)</td>
</tr>
<tr>
<td></td>
<td>SM1. Decide</td>
<td>SM4. First Demo</td>
</tr>
<tr>
<td></td>
<td>Functionality</td>
<td>SM5. SM PoC (Server+Client)</td>
</tr>
<tr>
<td></td>
<td>to AGL</td>
<td>to AGL</td>
</tr>
<tr>
<td></td>
<td>WM4. First Demo</td>
<td>WM5. Complete</td>
</tr>
<tr>
<td></td>
<td>WM3. Share APls</td>
<td>WM (Server+Client)</td>
</tr>
<tr>
<td></td>
<td>to AGL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WM4. First Demo</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WM5. Complete</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WM (Server+Client)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>HMI FW presentation with SM PoC demo &amp; WM Demo in AGL AMM Munich by TMC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend:
- Critical Milestones
- General Milestones
- Completed
- TBD
- Dependency

Mentor Automotive
Sound manager PoC - Architecture -

**Architecture**

- **User Interface**
- **Application Binder**
- **Sound manager Binding**
  - **Command plugin**
  - **Routing plugin**
- **Daemon**
- **Controller plugin**

**Legend**
- Need to develop
- Already exist

- **App business block**
- **Routing plugin**
  - ALSA
  - MOST
- **GAM**
- **AMB**
- **PCM**
- **CTL**
- **lib asound (alsa-lib)**
- **Sound card**
- **MOST / Eth-AV**

We can also make this using UNICENS
## Interface List

<table>
<thead>
<tr>
<th>#</th>
<th>Interface</th>
<th></th>
<th>For command plugin</th>
<th>For routing plugin</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>connect (sourceID, sinkID, &amp;mainConnectionID)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>disconnect (mainConnectionID)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>setVolume (sinkID, volume)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>volumeStep (sinkID, volumeStep)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>setSinkMuteState (sinkID, muteState)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>getVolume (sinkID, &amp;mainVolume)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>getListMainConnections (&amp;listConnections)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>cbNewMainConnection (mainConnection)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>cbRemovedMainConnection (mainConnectionID)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>cbMainConnectionStateChanged (mainConnectionID, connectionState)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>cbVolumeChanged (sinkID, volume)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>cbSinkMuteStateChanged (sinkID, muteState)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>cbNewSource(source)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>cbNewSink(sink)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>asyncAbort (&amp;handle)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>asyncConnect (&amp;handle, &amp;connectionID, sourceID, sinkID, connectionFormat)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>asyncDisconnect (&amp;handle, connectionID)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>asyncSetSinkVolume (&amp;handle, sinkID, volume, ramp, time)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>asyncSetSourceState (&amp;handle, sourceID, state)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ackConnect (handle, connectionID, error)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>ackDisconnect (handle, connectionID, error)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>registerSink (&amp;sinkData, &amp;sinkID)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>registerSource (&amp;sourceData, &amp;sourceID)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>hookInterruptStatusChange (sourceID, interruptState)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>hookSourceAvailabilityStatusChange (sourceID, &amp;availability)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>ackSetVolumes (handle, &amp;listvolumes, error)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Sound manager PoC - sequence -

- **Sequence**

  - **Application**
    - **Sound manager Binder**
    - **GAM**
    - **alt registration type**
      - [Dynamic]
        - `registerDomain()`
        - `registerSource()`
        - `cbNewSource()`
        - `registerSink()`
        - `cbNewSink()`
      - [Static]
        - `registerSource()`
        - `cbNewSource()`
        - `cbNewSink()`
    - **alt registration type**
      - [Dynamic]
        - `registerSink()`
      - [Static]
        - `registerSink()`
Sound manager PoC - sequence -

Sequence

sd Request

Application → Sound Manager Binder
connect() → connect()
cbNewmainConnection() → cbNewMainConnection()
asyncConnect() → asyncConnect()
open() → open()
ackConnect() → ackConnect()
setSourceState() → ackSetSourceState()
start playing() → ackSetSourceState()

sd Release

Application → Sound Manager Binder
disconnect() → disconnect()
cbMainConnectionStateChanged() → cbMainConnectionStateChanged()
setSourceState() → acksetSourceState()
stop Playing() → acksetSourceState()
asyncConnect() → asyncConnect()
cbNewMainConnection() → cbNewMainConnection()
setSourceState() → acksetSourceState()
close() → close()
asyncDisconnect() → asyncDisconnect()
removedMainConnection() → ackasyncDisconnect()
EOF
There are major 4 patterns of arbitration(policy) in automotive

1. The latter source win
   Discards the former source and output the latter source.

2. The latter source win and the former source pause
   Pauses the former source and output the latter.

3. The latter loose
   Continues former source and discards the latter source.

4. The latter source is put on hold
   Continues former source and puts latter source on hold

Queuing management is required