20200514-Virtual-Technical-Summit-AASIG-AudioHAL-Workshop-Minutes

1	**## Introduction**
2	Philippe Robin welcomed everyone and handover to Wassim 60 attendees Wassim describes the agenda (slide #2) slide deck for this workshop is here
3	**## Project overview & proof-of-concept demo**
4	Project overview presented by Wassim (slides #4 to #6) Google reference design only covers the interface to an external system but does not define it. By creating a proof of concept our goal is to demonstrate the feasibility of the split of an audio system (internal and external to Android) and to learn lessons on which limitations could possibly the Android interface have, how it could be extended in the future in order to facilitate integration with external automotive systems. Topics relevant for all strategies (internal, external, mixed) and pre-requisites for the proof-of-concept are detailed in slides #7 to #9 We expect today to take opinion from the people on what topics are currently relevant for further discussion today.
5	Proof-of-concept presented by Piotr (slides #13 to #15) • Presentation about the PoC and what have been done • Presentation of the demo, followed by demo running using the emulator, look at demo.md • Lessons learned (look at read.md in github demo repo, repo: android-external-audio-mixing repository

6 Q&A on Proof-of-concept

- Q: What about low latency streams ? Are we confident enough that the low latency streams can be handled in the external system ?
  - proof-of-concept overview, on the right diagram (slide #11), GENIVI audio management can provide the apis to serve low latency audio
  - so there are 3 possible approaches for low latency audio: use the external audio system, use the GENIVI (custom) audio management or use AA audio management
  - yes, there are these 3 options for the third parties developers, however there is a need to sync audio and calculate audio latencies, but this can be done
- Q (Avinash): CPU offloading, what do you mean exactly by this ? Is it considered ?
  - CPU offloading is when the DSP can take care of some action instead of the CPU
  - some audio streams can be decoded using DSP, AA has such a feature to feed DSP with encoded data to play
  - Android would like to prevent 3rd party developers from taking such decisions and rather inform that such streams can be offloaded.
  - For now the proof-of-concept is not considering CPU offloading, we know it's feasible but it's not yet considered in this proof-of-concept
- Q (Spandya) : if audio is played by Android application but external mixer for some reason does not given priority due to other sources , how app layer gets notified about the error ?
  - ideally the external mixer would be used for things that go smootlhy in the audio system like chimes and warnings, usually warnings are overlays that are mixed "on top" meaning they would not stop the entertainment music completely but would be played along with it, in that case the application would be notified that it is an error
  - I am not sure there is a case where the external system will completely overrides the source, if this happens, I admit there must be a notification mechanism toward the app layer
  - there are some internal mechanisms that covers this, this feedback mechanism is to be checked in the proof-of-concept
- Q (Sriram): Can you please explain the diagram on the right (slide #13)?
  - the poc is just starting, this is not a finalized design yet, iteration process
  - our objective is not to modify Android (audio management box in the middle of the diagram) to follow Treble and use it as it is with the logic it provides
  - system level audio management is the name we gave to any system that provides additional logic Android might not be aware of like safety signals or other signals
  - the system level audio management proxy (on the top right of slide #13) is an entity that supports any logic that is not available in Android or any low latency audio that is not supported by Android well enough, this entity will complement Android Audio HAL with an additional HAL that the application could use
  - the way Android can communicate with the system level audio management is explained by the following diagram
    - Piotr Krawczyk can you add / link the diagram you showed at this step in the web conference to the workshop slide deck please ?
- Q (Sriram): What are the socket kind of ports shown in the lower block of slide #13 ?
  - One of the first findings we made with the proof-of-concept is that the network sockets cannot be used directly by the Audio HAL(so the proof-of-concept played its role of experimenting a solution that was not successful), so the network sockets have been deprecated in the proof-of-concept, and we use currently a different connection system
  - the sockets are now realized by the audio relay service that is able to get data from local service and forward them to external TCP sockets (look at slide #15), code for this audio relay is available but quite trivial
  - here you can find build instructions: android-external-audio-mixing/tree/develop/vendor/genivi/modules/audiorelay repository
  - look also at https://developer.android.com/studio/run/emulator, it relies on qemu and cannot run on Virtual Box, it runs in parallel to it
- Q (Sunita): How does this proof-of-concept work in case of a fail-safe scenario ? Can you explain more on system proxy part considering start-up and shutdown use case ?
  - this question cannot be answered in one or two sentences, having an external proxy actually enables us to have early audio before Android is available, we will look into this with our proof-of-concept but we have no detailed answer yet
  - actually in the proof-of-concept, the signal (PCM data) is redirected to 2 devices, i.e. to the audio relay that later on sends the audio data to the PCM writer (slide #15) but also to the original audio device connected to the emulator, during start-up time when network is not available this gives us the possibility to play audio signals because ALSA is already there, this is how we might implement a fail-safe scenario
    - also it is worth mentioning that in a production system, there will be 2 types of HALs, one which is available at start-up with minimum functionalities, that will replaced by a full-fledge HAL when start-up is completed
  - we need to validate this approach with the proof-of-concept

7 ## **Next Step for the Proof-of-Concept**

8 ### **Audio Manager Integration**

9 GENIVI has an audio manager framework used in Linux environment. It's already in multiple production products. It handles both the operating system audio and the system level audio, it fulfills all the reliability and safety requirements of automotive systems such the early startup features, etc. It's only a framework that allows each vendor to write and link their own logic in a form of a plugin and differentiate w.r.t. use case support. The plugin can run everywhere there is a GENIVI Audio Manager framework.

In our work, we are using Android but still we would like to benefit from both Android and GENIVI Audio Manager, this is why on slide #13, we have the following two Audio Managers

- Android Audio Management
- GENIVI Audio Manager Framework and related App in the system level audio management proxy.

How to integrate these two Audio Managers is still a topic of investigation using the proof-of-concept, we would be interested in getting feedback from the participants

At first glance, GENIVI Audio Manager would take over when the Android Audio management cannot handle certain properties of external streams, GENIVI Android App would be just an assistant that is able to interact directly with external audio subsystems and this will use the GENIVI Audio Manager Framework

10 Q&A on Audio Management

- Q: Will there be a conflict between having two Audio HALs (Android and GENIVI) ?
  - No in our opinion, the App should be able to decide which one to use
- Q (Sriram) : What use cases are being considered for this GENIVI Audio Framework ? Is there some use cases that Android cannot fulfill ?
  - Piotr: Android 11 would have some ways to control the global effects but for now with Android 10 for example, the global effects and equalization are not available system-wise, they could however be handled by GENIVI Audio Manager Framework
  - Wassim: It is valuable to have the GENIVI Audio Manager as an existing solution and reliable framework that can be used for pre-development models until the feature is supported by Android. This can be useful in a transition from predev to production system development
  - Gunnar: But this contradicts to some extent what we said before about the need to have features implemented outside of Android (and not as a temporary solution until Android provides it)
  - Gunnar: The question is really : do we need to make any change outside of Android ?
  - Wassim: The fact that we need to have an external audio management is understood (as explained previously), there are safety features that need to be handled outside of Android.
  - Wassim: The question then is rather: do we need an additional interface to control any custom function that is not provided by the standard Android interface, i.e. do we need this proxy box on the top right (look at slide #13) ?
  - Sriram: Agreed: The question of the blue box (external audio system) at the bottom is understood,
  - Sriram: The point is that there is no valuable use case to make the GENIVI Audio Manager proxy (or GENIVI App) established yet, this is the concern: if we are making something outside of Android we need to make sure we need it, we need to rationalize our proposal because this box on the top right is too generic and abstract and at the same time it relates to hardware and low-level stuff.
  - One can also question which kind of app could access the proxy ?
  - Wassim: The reason why we are developing the proof-of-concept is to answer this kind of questions : do we need this feature (e.g. this proxy) or not ? can we achieve the mapping of all use cases on standard Android not ?
    - By the end of proof-of-concept we expect to have the following :
      - An answer to the question whether we need an outside system or not
      - Proof points for Google Android team that we need an outside system
      - A learning experience as well as a knowledge on how to deal with such use cases relying on an outside system
      - A learning experience about the numbers behind the low latency
      - A more detailed understanding of the use cases
  - Wassim: The proof-of-concept is not a full-blown architecture, it is kind of a placeholder where we can collect all the features that Android cannot support or cannot support easily and understand the Android audio system and its shortcomings
  - Sriram: Agreed, as a community we can help understanding the audio management system and its shortcomings
  - Sriram: But we need to have measurements to quantify shortcomings e.g. for the low latency
  - Wassim: Low latency question, let us talk about it
    - we have two cases for low latency, we have either low latency for signals coming from the car (like the user opens the door) and this kind of low latency can be handled externally. The second case is when an App needs a low latency sound, when we talk about low latency, are we talking about a system device or an Android application ?
  - Sriram: Agreed, there are two ways to look at low latency, there are these stream types which have been very nicely classified both by Android and GENIVI audio managers, but we should first check the system resources used and the numbers "screen by screen" and verify whether the GENIVI Audio HAL does solve the low latency before we can say that it's a solution
- Q (Subramanian): Is there a plan to sync various audio streams, if we are relying on an external device for audio processing ?
  - Yes this is one of the topics we want to consider in the proof-of-concept (look at slide #15, topics requiring assignment : some signals require time synchronisation)
  - there are some protocols that do have time synchronisation like precision time protocol (https://standards.ieee.org/standard/1588-2008.html) or other audio/video protocols, but this has not been thoroughly discussed yet
  - this is an open topic for discussion (Alpine)
- Q (Shivakumaraswamy): How latency info is passed to other decoders like video in order to sync video and audio ? Will this be handled by an external audio mixer ?
  - There is a standard called Audio Video Bridge (AVB). Supporting this protocol would solve a lot of issues. This is a question for Android, do they support AVB ?
  - This can be added as a bullet point to the list of topics
  - But for other protocols that have no AVB, it's not clear
  - AVB has been expanded on in the HW standards with a set of tools encapsulated in the TSN standards
- Q (Pawel): BT telephony: we have a problem here: we need to be part of the audio management but we need as low latency as we can, how does this fit in the global architecture
  - There is no simple answer to this question, it can be realized at DSP level, this is why we did not include this use case in the general architecture
  - perhaps the GENIVI audio HAL is a place where we could provide some improvement on this, but very often the BT audio device is connected directly to the Android partition
  - the problem is also with more than 1 phone: DSP can handles the low latency for this use case but we still need to be connected to the audio management and synchronize with it (to stop some other channels)
  - it is more complicated with the speech capturing which needs also to be low latency and involves many clients in-between
  - We need to experiment this use case with our proof-of-concept

11 ### **Extracting Raw Streams**

12 Extracting Raw Streams, a pre-requisite for the PoC (slide #8)

- the BT telephony and speech applications are one of the most complex because at the same time they need extracting streams from Android and inputting streams into Android in case we want handle some use cases externally
- we would like to assess how difficult or easy extracting raw streams is, we will transform all the bullet points of the slide into steps for the proof-of-concept implementation (and determine which one is solved and which one requires attention)
- Piotr: audio policy configuration is something quite interesting and specially in the context of multi-zone audio, like separating context streams and placing them to different audio devices and passing some information about currently playable audio zones
- Wassim: configuring audio system seems to me quite challenging
- Piotr: this is correct and Google is changing things from time to time, it is a quite complex task we should investigate with the proof-of-concept
- Wassim: Do we need some sort of a GUI that shows the connection and help controlling different external raw streams ?
  - Maybe for testing nd learning how it works
  - Yes, it would help us understanding various issues
- We have the same problem with GENIVI AudioManager and its configuration, it's a complicated state machine and maybe we should find a way to visualize it.
- Pawel: It would be good to have some sanity check made for such the complex logic (and the policy conflicts) we can have in the Android Manager or the GENIVI Audio Manager
- Gunnar: I would recommend following rather the GENIVI audio manager approach which is to program the complex logic in the plug-in (ant not using yet another configuration language), code can be readable (much more than XML)
- Wassim: how to make the GENIVI audio manager Android-friendly is a topic for investigation

13 ## **Injecting input streams**

14 (slide #9)

- Wassim: it seems that Android has not limitation about how many external streams can be injected
- Piotr: yet another topic for investigation with the proof-of-concept
- Nadim: there is a new API called the hardware audio source player that links everything inside our system level audio wki page
- this replaces and improves the former audio patch API
- the new API is about mixing internal and external streams with Audio Flinger

15 ## **Next steps and next milestones content**

16

- Wassim: how would be split the features for the next milestones
- Piotr: IMHO in order to move forward, we need to discuss hardware support and move away from the emulator
- Wassim: We haven't talked about hardware support so this can be a point of discussion
  - maybe some participants have a particular platform in mind, for now on we will go with the platform that is mostly used by the active participants
  - the way we map the architecture of slide #4 onto actual hardware is to be defined for the next milestone
- Piotr: offloading using a DSP is also something we should address for the next milestone
- Pawel: does Android provide some generic DSP support (with APIs for DSP) or is it completely out of Audio HAL ?
- Wassim: there are extensions for Khronos group like OpenGL that support hardware accelerations
- Pawel: there is a need to express which streams go to which entry points of the DSP and for which purpose (equalization, volume normalization, etc.), this is all about audio configuration, I am not sure if Android supports such audio routing at the DSP level
- Piort: audio effects should be used for this purpose, audio effects have a separate HAL that can realized with DSP functionalities, as for today the application processor can handle easily mp3 decoding but a better approach is to send the compressed stream (mp3 / mp4 containers) to the DSP for decompressing and playing (and not to use the decompression libraries on the application processor)
- Wassim: shows https://source.android.com/devices/media and underlines the integration of custom codecs like the OpenMAX codecs (https://www.khronos.org/openmax), this would be the right interface to support hardware acceleration in a generic way
- Piotr: agreed, this relates to the multimedia framework actually
- Wassim: as said already, in my opinion the next milestone would be to use a real hardware for running the proof-of-concept demo instead of adding more features to the emulator, personnally I like the fail-early approach: try, fail and learn using a real hardware
- Piotr: we need to use a hardware that have already some audio management support in order to integrate with the audio manager and check how to support a kind of hybrid mode where we mix the Android player with some content (from an external mic for instance)
- Wassim: next milestones could be organized as follows
  1. implementation of the demo on real hardware
  2. full audio path in real hardware
  3. implementation of feature partitioning inside and outside Android
     a. Piotr: would include BT telephony in there
- Q. What hardware do you have in mind?
  - Raspberry PI - it is not an official Android hardware
  - Gunnar: for the GENIVI AASIG we set up a development platform that uses the following hardware: NXP iMX8 and Renesas H3 +Kingfisher board, I would recommend using those for the audio proof-of-concept because they have all the audio support needed
  - Gunnar: we identified also the HiKey 960 as a low-cost board
  - Marius: we stopped using HiKey boards because they are not compliant with the GAS requirements, also the NXP board is only supported by older versions of Android, and we discarded it as well, in the end we are using Intel NUC 6 & 7 and Qualcomm 8195 and 820
  - Gunnar: we only got support from NXP and Renesas for setting the "official" AASIG development platform
  - Wassim: we have to be harware independent
  - Gunnar: everyone is free to take the code and port it to whatever board is available to them
  - Wassim: as a transition to the next topic in the agenda, I remind that in the milestone #3 above, we will not be able to support all features and this is why we need to revisit the list of prioritized topics with today's participants

17 ## **Review of List of prioritized topics for the Audio HAL**

18

- Wassim: this list enumerates the activities that as a group the Audio HAL will try to tackle
- For the next milestones, we investigate these topics using the proof-of-concept and see if the proof-of-concept can help solving them and determine which strategy is better
- list updated online with a vote column introduced
- miscellaneous remarks
  - Point 13: Bluetooth
    - Sriram: how do we plan to address this ? this is very much hardware dependent,
    - Pawel: my concern with BT telephony is more with the latency because the latency is usually handled inside the DSP and this does not even go to operating system level (just maybe some control messages reach the operating system), my question is how this will be handled with Android because all applications need to be informed that something is going with the telephone and need to react
    - Sriram: in my opinion, your concern has nothing to do with acceleration and more to do with routing
  - Dependencies with hardware
    - Sriram: it would be good to identify which topics are hardware dependent and require special resources and which topics are not hardware dependent
    - Piotr: as GENIVI we cannot implement things that are hardware dependent, this is the role of hardware vendors to do that
    - Sriram: agreed, then we have to identify the constraints for each activity in the list, it is worth mentioning that some use cases can be fully implemented in software even if we can have in the end a solution that is hardware dependent
    - Wassim: is there any disadvantage using a full software implementation ?
    - Piotr: mobile telephony experience provides some answers, we need to balance between performance / efficiency / latency and power consumption and ease of porting
    - Pawel: agreed, the latency in telephony is critical, we need to have that loop in 200ms with filtering and everything, you have an input from the BT chip, you go the operating system and back from it, it is very hard to achieve this 200ms time constraint
    - Wassim: it is up to the project to determine whether a feature will be implemented in hardware or software, as a community project we should try to minimize harware dependencies and focus rather on software implementation, however for a particular feature like the BT telephony it might be worth using a hardware dependent implementation