Table of Contents

Background

Automotive Grade Linux (AGL) is a collaborative open source project that is bringing together automakers, suppliers and technology companies to accelerate the development and adoption of a fully open software stack for the connected car. Being a part of speech expert group, Amazon (Alexa Automotive) team intends to collaborate to help define the voice service APIs in AGL platform.

Definitions

Voice Services

These are standard set of APIs defined by the AGL Speech EG for users and applications to interact with voice assistant software of their choice running on the system. The API is flexible and attempts to provide solution for uses cases that span, running one or multiple voice assistants on the AGL powered car head units at the same time.

Voice Agent

Voice agent is a virtual voice assistant that takes audio utterances from user as input and runs It's own speech recognition and natural language processing algorithms on the audio input , generates intents for applications on the system or for applications in their own cloud to perform user requested actions. This is vendor-supplied software that needs to comply with a standard set of APIs defined by the AGL Speech EG to run on AGL.

Purpose

The purpose of this document is to propose the Voice Service and Voice Agent APIs for Speech Services framework, and get feedback from AGL speech task group..

Disclaimer: This is not a standard. It's only a proposal that is shared to receive feedback and inputs from the AGL Speech Group.

Assumptions

One voice agent will not possess all the capabilities that customer desire. So, more than one voice agent needs to exist on the platform and they need to work together at times to fulfill a customer ask.

One voice agent will not be able to properly detect all the explicit invocation words (wake words) of other voice agents. For example, Amazon best optimizes the machine learning models for the Alexa wake word. Same with other voice agent vendors as well.

Customer Experience Requirements

In the proposal, we would like to address following customer experiences --

Customer A: As a IVI head unit user, I would like to have the following experiences,

- **Experience A: With only one active voice agent.**
  I should be able to select the active voice service agent.
  "Alexa, what's the weather" and "what's the weather" should be routed to the user selected voice agent.

- **Experience B: Multiple active voice agents use cases.**
  - **Invocation**
    - **Explicit:** Speech framework will pick the appropriate agent based on keyword detection to perform a task.
      For this approach to work neatly all the voice agents should have a wake word. If I say, "Alexa, what's the weather", my utterance should be routed to Alexa voice agent on the device.
    - **Multi Modal Interaction**
      - (User) Agent A, Route me Starbucks nearby
      - A list of Starbucks near me is presented to the user.
      - (User) Agent A Select first one OR
      - (User) Clicks on the first item in the list
      - (Agent A) Launches Navigation App with geocode of selected Starbucks
    - **Fallback**: Speech framework will fallback to a different voice agent if the chosen one fails to fulfill a request.
      - Silently route my utterance if the chosen agent fails to perform a task.
      - Inform me using speech that Agent A failed to fulfill the request and so its trying Agent B.
    - **Proactive**: Any voice agent should be able to initiate a dialog or perform an action without a corresponding implicit or explicit user invocation.
      - Based on system behavioral changes. Agent A warns user that "Tire pressure is low" or " Maintenance is due, do you want to schedule it ? ".

- When Agent A does a restaurant reservation, then Agent B can offer a parking spot near that restaurant if its assigned and if it possess that capability.

  - **Multi Turn Dialog Use Cases**
    - No switching to a different voice agent when I am in an active dialog with some other agent. All the user speech will be routed to the voice agent that initiated the dialog.

    - Agent switching based on keyword detection
      - (User) Agent A book flight ticket to Seattle
      - (Agent A) What time you want to leave?
      - (User) Agent B, what time my last meeting ends on Mon
      - (Agent B) 5:30
      - (User) Agent A book ticket after 6:30 pm

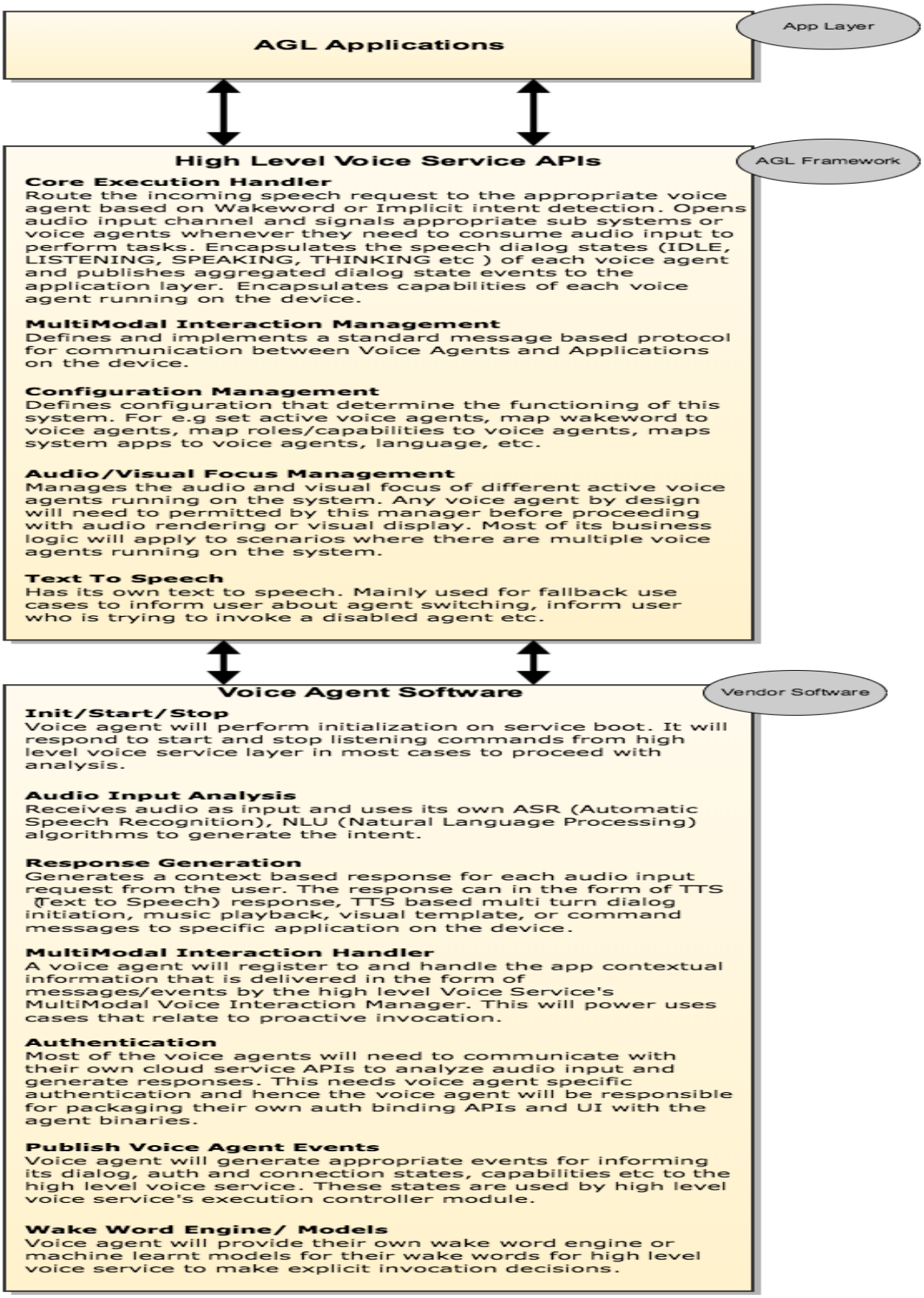
- **Customer B: As a car OEM,**
  - I should be able control the agents that run on my system, their life times and their responsibilities.

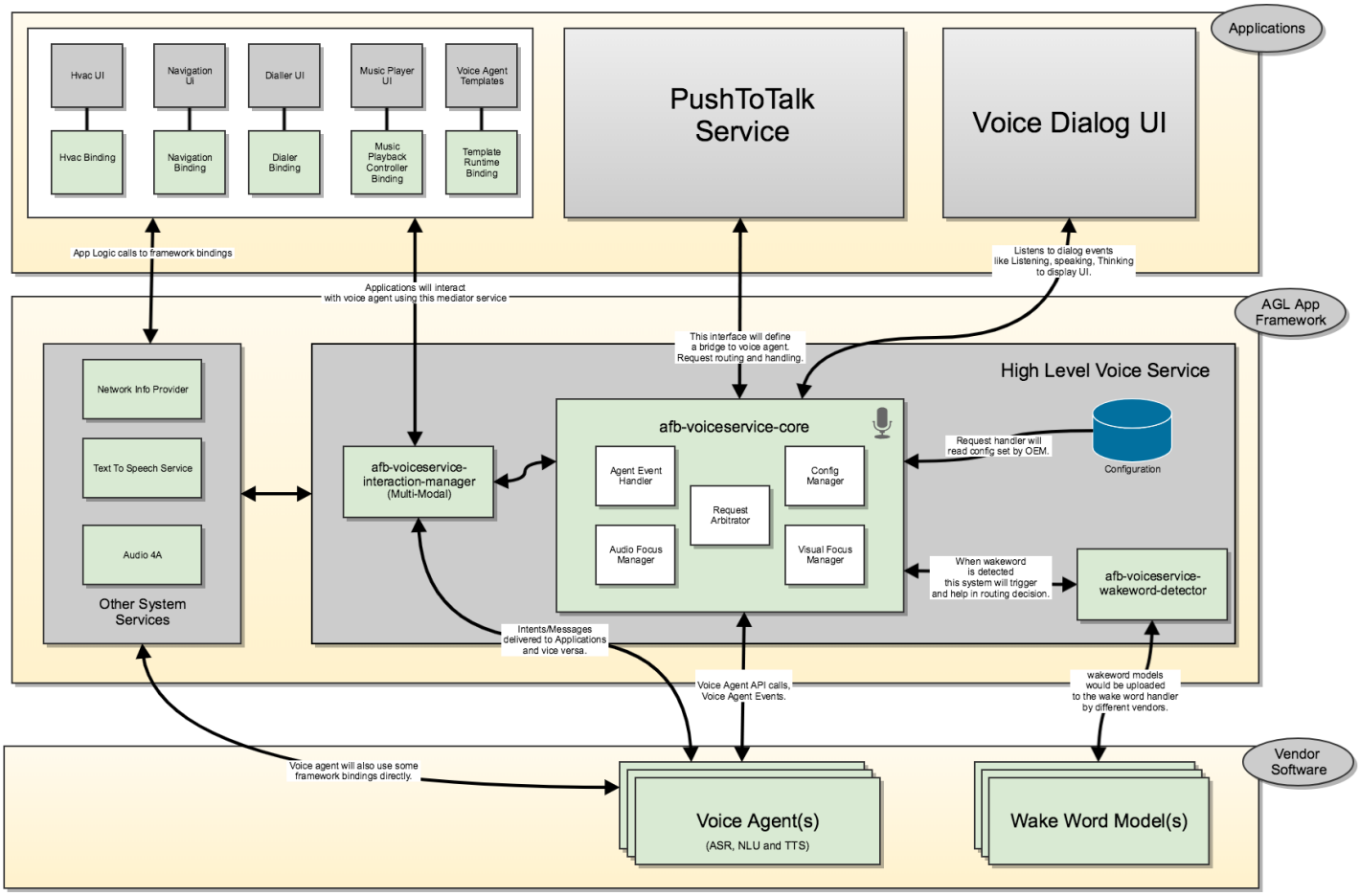
- **Customer C: As an AGL Application developer,**
  - I should be able to use AGL Speech framework's to voice enable my application.

- **Customer D: As a 3rd party Voice Agent Vendor,**
  - I should be able to follow the guidelines of the AGL speech framework to port my voice agent.
  - I should be able to follow the guidelines of the AGL speech framework to port my wake word solution

High Level Architecture

Architecture Tenet: From AGL documentation, “Good practice is often based on modularity with clearly separated components assembled within a common framework. Such modularity ensures separation of duties, robustness, resilience, achievable long term maintenance and security.”


App Layer
AGL Applications
AGL Framework
High Level Voice Service APIs
Core Execution Handler
Route the incoming speech request to the appropriate voice agent based on Wakeword or Implicit intent detection. Opens audio input channel and signals appropriate sub systems or voice agents whenever they need to consume audio input to perform tasks. Encapsulates the speech dialog states (IDLE, LISTENING, SPEAKING, THINKING etc) of each voice agent and publishes aggregated dialog state events to the application layer. Encapsulates capabilities of each voice agent running on the device.
MultiModal Interaction Management
Defines and implements a standard message based protocol for communication between Voice Agents and Applications on the device.
Configuration Management
Defines configuration that determine the functioning of this system. For e.g set active voice agents, map wakeword to voice agents, map roles/capabilities to voice agents, maps system apps to voice agents, language, etc.
Audio/Visual Focus Management
Manages the audio and visual focus of different active voice agents running on the system. Any voice agent by design will need to permitted by this manager before proceeding with audio rendering or visual display. Most of its business logic will apply to scenarios where there are multiple voice agents running on the system.
Text To Speech
Has its own text to speech. Mainly used for fallback use cases to inform user about agent switching, inform user who is trying to invoke a disabled agent etc.
Vendor Software
Voice Agent Software
Init/Start/Stop
Voice agent will perform initialization on service boot. It will respond to start and stop listening commands from high level voice service layer in most cases to proceed with analysis.
Audio Input Analysis
Receives audio as input and uses its own ASR (Automatic Speech Recognition), NLU (Natural Language Processing) algorithms to generate the intent.
Response Generation
Generates a context based response for each audio input request from the user. The response can in the form of TTS (Text to Speech) response, TTS based multi turn dialog initiation, music playback, visual template, or command messages to specific application on the device.
MultiModal Interaction Handler
A voice agent will register to and handle the app contextual information that is delivered in the form of messages/events by the high level Voice Service's MultiModal Voice Interaction Manager. This will power uses cases that relate to proactive invocation.
Authentication
Most of the voice agents will need to communicate with their own cloud service APIs to analyze audio input and generate responses. This needs voice agent specific authentication and hence the voice agent will be responsible for packaging their own auth binding APIs and UI with the agent binaries.
Publish Voice Agent Events
Voice agent will generate appropriate events for informing its dialog, auth and connection states, capabilities etc to the high level voice service. These states are used by high level voice service's execution controller module.
Wake Word Engine/ Models
Voice agent will provide their own wake word engine or machine learnt models for their wake words for high level voice service to make explicit invocation decisions.



The Voice Services architecture in AGL is layered into two levels. They are High Level Voice Service layer and vendor software layer. In the above architecture, the high-level voice service is composed of multiple bindings APIs (colored in green) that abstract the functioning of all the voice assistants running on the system. The vendor software layer composes of vendor specific voice agent software implementations that complies with the Voice Agent Binding APIs.

High Level Work Flow

Experience A: With only one active voice agent at a time. User selected the active agent.

Assumption: Voice agents are initialized and running, and are discoverable and registered with afb-voiceservice-core. afb-voiceservice-core has subscribed to all the events of voice-agent-binding.

- afb-voiceservice-core **startListening** API will be triggered by Push-To-Talk button invocation.
- afb-voiceservice-core opens the audio input channel using audio 4a high level binding APIs, sets up the shared memory location to which it would copy the audio input buffer.

- afb-voiceservice-core requests afb-voiceservice-wakeword-detector to **startListening** and also passes audio input's shared memory location to it.
- afb-voiceservice-core will move from IDLE to LISTENING state. The Voice Dialog UI app would show a voice chrome or similar UI indicating the user to start speaking.
- User starts speaking. afb-voiceservice-core will wait for a few milliseconds for the afb-voiceservice-wakeword-detector to come back with **onWakeWordDetected** event.
    - If afb-voiceservice-wakeword-detector doesn't detect wake word within the aforementioned timeout, then afb-voiceservice-core will pick the active agent selected by user for further processing.
    - If afb-voiceservice-wakeword-detector detects the wake word, then afb-voiceservice-core will check if detected wake word is actually assigned to the active agent or not.
        - If it's not, then afb-voiceservice-core will call TTS service to prompt an audio error response.
        - If yes, then afb-voiceservice-core will call the **startListening** API of the active voice-agent-binding. Along with that it also passes the audio input's shared memory location as parameter, and an offset from which the voice-agent-binding should read the audio data. It will also inform the voice-agent-binding if its expecting end of speech detection event from the voice-agent-binding or not.
- voice-agent-binding, upon receiving **startListening** call, will transition between LISTENING, THINKING, and SPEAKING states and will regularly publish **onDialogStateChanged** event with its current state to afb-voiceservice-core.
- voice-agent-binding will start reading audio input until **stopListening** is called or until end of speech is detected. Once end of speech is detected it responds with **onEndOfSpeechDetected** event.
- Voice agent will perform ASR, NLU and triggers different actionable responses.
    - If the response is music audio playback, voice-agent-binding will interface with AGL framework's media player binding to create an audio channel and stream the audio.
    - If voice-agent-binding needs to present a TTS as response or initiate a multi turn dialog, it will do so by calling audio 4a binding and moving to SPEAKING state. In this case, the afb-voiceservice-core will also move to SPEAKING state. The Voice Dialog UI will present the UI representing the SPEAKING state of the voice agent.
    - If the response needs to command a system application to perform an action like NAVIGATE_TO or CALL, then it will send a topic based message/intent using afb-voiceservice-interaction-manager binding.
    - If there is an associated UI card to be displayed, voice-agent-binding will send another message to the card rendering application using the afb-voiceservice-interaction-manager binding.

Bindings

Below is a technical description of each of these binding in both the levels.

High Level Voice Service

High-level voice services primarily runs in following two well known modes.

- Push-To-Talk Mode
    - In this mode, the user will need to press the push-to-talk button on the car steering wheel to talk to the voice agents.
    - If user doesn't mention the wake word then the utterance will be routed to the active voice agent.
    - If user mentions a wake word, then the utterance will be routed the appropriate agent.
- Always listening Mode
    - In this mode, upon wake word detection the utterance will be routed to the appropriate voice agent.

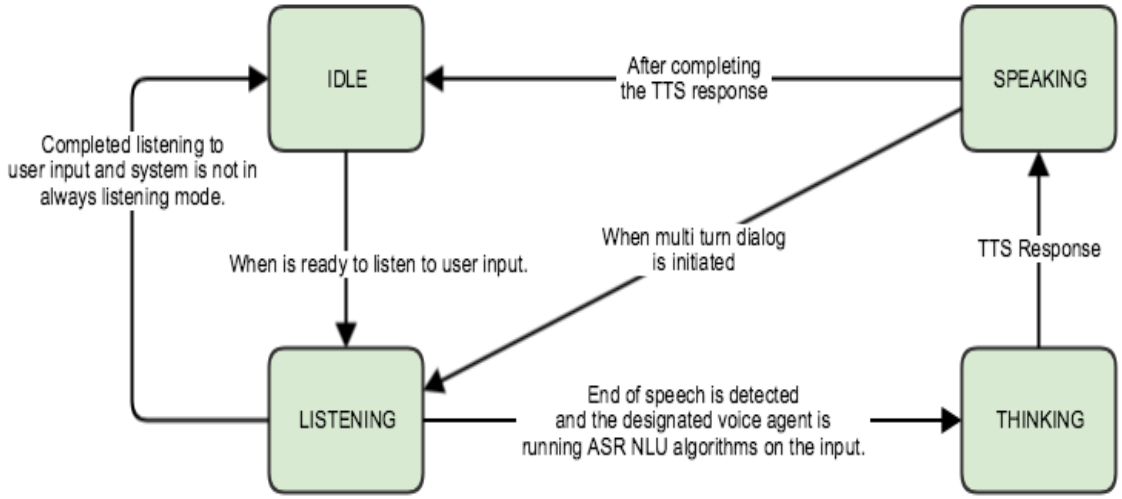
This design makes no assumptions on the mode in which the high-level voice service component is configured and running.

afb-voiceservice-core

This binding has following responsibilities.

- Structurally follows a bridge pattern to abstract the functioning of the specific voice agent software from the application layer.
- The request arbitrator is main entry point to the system. It's responsible for routing the utterances to the correct voice agent based on various parameters like configuration, wake word detection etc.
- Registers for dialog state events from each voice agent that it manages. Maintains the latest and the greatest state of the voice agents.
- Audio/Visual Focus management. Provides an interface using, which the voice agents can request, audio or visual focus before actual rendering the content. In multiple active voice agent scenarios, we can imagine that each agent would be competing for audio and visual focus. Based on the priority of the content, the core should grant or deny focus to an agent. In cases where it grants the focus, it has to inform the agent currently rendering the content to duck or stop. And make audio and visual focus decisions on behalf of the voice agents its managing.

State Diagram



API

1. void voiceservice_hl_core_api_startListening (struct afb_req req)

Starts listening for speech input. As a part of request, common configuration related information is passed.
Note: The config inputs below are just examples and not the final list of configurations.

"permission": "urn:AGL:permission:speech:public:audiocontrol"

```
Request: {
  "language":"string"
  "location":"string"
  "preferred_network_mode":"string" // online, offline, hybrid
  "speech_initiation_mode":"string" // TapToTalk, HoldToTalk
}

Responses: {
  "jtype":"afb-reply",
  "request": {
    "status":"string" // success or bad-state or bad-request
  }
}
```

2. void voiceservice_hl_core_api_stopListening (struct afb_req req)

Stops listening for speech and requests the voice agent to stop processing the speech input. Note that in the default case, this does not need to be called, as the voice service end point will automatically stop the processing when it determines speech has completed or when speech recognition is completed.

"permission": "urn:AGL:permission:speech:public:audiocontrol"

```
 Request: {
}

Responses: {
  "jtype":"afb-reply",
  "request":{
    "status":"string" // success or bad-state or bad-request
  }
}
```

3. void voiceservice_hl_core_api_isAvailable (struct afb_req req)

Check if the voice agents are available and running on the platform.

"permission": "urn:AGL:permission:speech:public:accesscontrol"

```
Request: {
}

Responses:
{
  "jtype":"afb-reply",
  "request":{
    "status":"string" // success or bad-state or bad-request
  }
  "response": {
    "available":"boolean"
  }
}
```

4. void voiceservice_hl_core_api_event_subscribe (struct afb_req req)

Subscribe/Unsubscribe to voice service high level events.

"permission": "urn:AGL:permission:speech:public:accesscontrol"

Request:
```
{
 {
  "type":"array",
  "items" : [{
     "type":"string"
   }
  ]
 },
 {
  "subscribe":"boolean"
 }
}
```

Responses:
```
{
 "jtype":"afb-reply",
 "request":{
  "status":"string" // success or bad-state or bad-request
 }
}
```

Events

High Level Voice service layer subscribes from similar states from each of the voice agent's and provides an agent agnostic states back to the application layer.

For e.g., if Alexa is disconnected from its cloud due to issues and Nuance voice agent is connected, then the connection state would be still reported as CONNECTED back to application layer.

1. Dialog state describes the state of the currently active voice agent's dialog interaction.

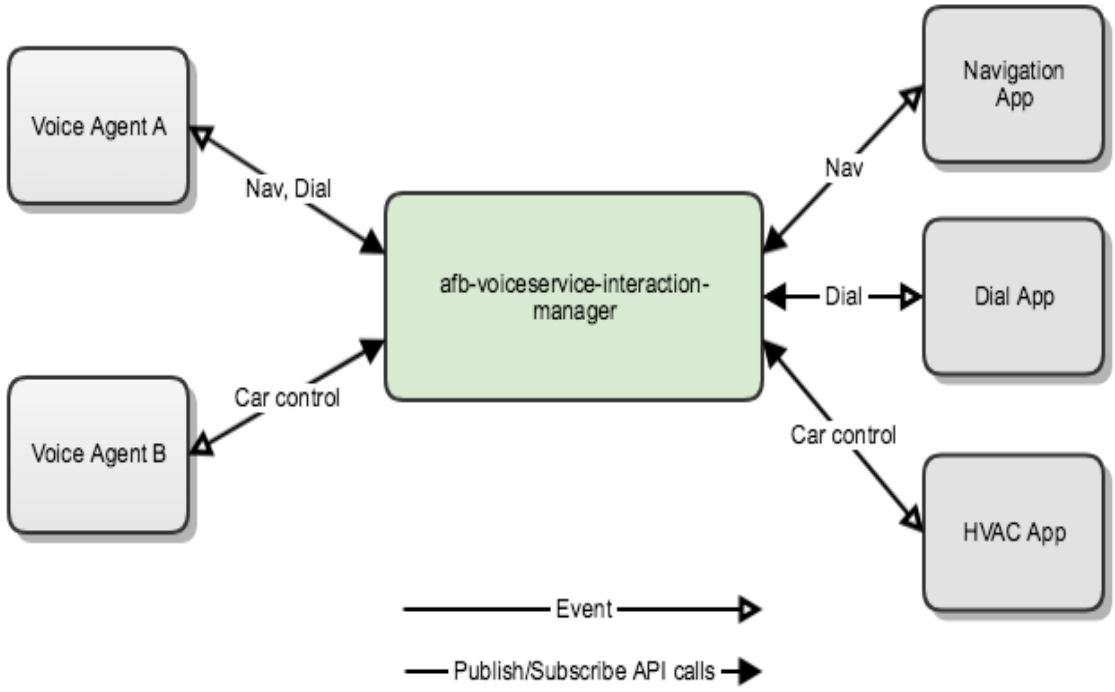
Event Data:
```
{
 "name" : "voiceservice-hl-dialogstate-event"
 "state":"string"
 "agent_id": "integer"
}
```

Values for state are
IDLE: High-level voice service is ready for speech interaction.
LISTENING: High-level voice service is currently listening.
THINKING: No more input is needed at this point. In this state, a Voice Agent is working on a response.

SPEAKING: Responding to a request with speech..

afb-voiceservice-interaction-manager

An important binding of the high-level voice service that acts as a interaction mediator or a message broker between the voice agents and applications. The mode of communication is through messages and architecturally this binding implements a topic based publisher subscriber pattern. The topics can be mapped to the different capabilities of the voice agents.



Message Structure

{

Topic : "{{STRING}}" // Topic or the type of the message

Action: "{{STRING}}" // The actual action that needs to be performed by the subscriber.

Payload: "{{OBJECT}}" // Payload

}

Topics

Voice Agent to Applications are downstream messages and Applications to Voice Agent are upstream messages.

Voice agents and applications will have to subscribe to a topic and specific actions within that topic.

1. Phone Control

Downstream

```
{
  Topic   : "PhoneControl"
  Action : "DIAL"
  Payload : {
     "callId": "{{STRING}}", // A unique identifier for the call
     "callee": { // The destination of the outgoing call
        "details": "{{STRING}}", // Descriptive information about the callee
       "defaultAddress": { // The default address to use for calling the callee
          "protocol": "{{STRING}}", // The protocol for this address of the callee (e.g. PSTN, SIP, H323, etc.)
          "format": "{{STRING}}", // The format for this address of the callee (e.g. E.164, E.163, E.123, DIN5008,
etc.)
          "value": "{{STRING}}", // The address of the callee.
       },
     "alternativeAddresses": [{ // An array of alternate addresses for the existing callee
          "protocol": "{{STRING}}", // The protocol for this address of the callee (e.g. PSTN, SIP, H323, etc.)
          "format": "{{STRING}}", // The format for this address of the callee (e.g. E.164, E.163, E.123, DIN5008,
etc.)
          "value": {{STRING}}, // The address of the callee.
       }]
     "required": [ "callId", "callee", "callee.defaultAddress", "address.protocol", "address.value" ]
  }
}
```

Upstream

```
{
  Topic   : "PhoneControl"
  Action : "CALL_ACTIVATED"
  Payload : {
```

"callId": "{{STRING}}", // A unique identifier for the call

"required": ["callId"]

}

}

{

Topic : "PhoneControl"

Action : "CALL_FAILED"

Payload : {

"callId": "{{STRING}}", // A unique identifier for the call

"error": "{{STRING}}", // A unique identifier for the call

"message": "{{STRING}}", // A description of the error

"required": ["callId", "error"]

}

}

Error codes:

4xx range: Validation failure for the input from the @c dial() directive
500: Internal error on the platform unrelated to the cellular network
503: Error on the platform related to the cellular network

{

Topic : "PhoneControl"

Action : "CALL_TERMINATED"

Payload : {

"callId": "{{STRING}}", // A unique identifier for the call

"required": ["callId"]

}

}

2. NAVIGATION

Downstream

```
{
  Topic   : "Navigation"
  Action : "SET_DESTINATION"
  Payload : {
     "destination": {
        "coordinate": {
          "latitudeInDegrees": {{DOUBLE}},
          "longitudeInDegrees": {{DOUBLE}}
        },
        "name": "{{STRING}}",
       "singleLineDisplayAddress": "{{STRING}}"
       "multipleLineDisplayAddress": "{{STRING}}",
    }
  }
}

{
  Topic   : "Navigation"
  Action : "CANCEL_NAVIGATION"
}
```

3. TO-DO

Playback Controller

Car control (Door lock/unlock, Thermostat etc.)

Agent specific UI Template rendering

afb-voiceservice-settings

- Provides mechanism for OEMs to configure its functionality. OEMs should be able to configure
- List of active agents.
- Assign roles and responsibilities of each agent.

- Language setting.
- Default Agent Selection.
- Enable/Disable Fallback Invocation mode
- Enable/Disable Agent Switching during multi turn dialog
- And More.

API

1. void voiceservice_hl_settings_api_enumerate_agents(struct afb_req req)

"permission": "urn:AGL:permission:speech:public:accesscontrol"

Enumerates and return an array of voice agents running in the **system**. This might be need for the applications like settings to be able to present some UI with a list of agents to enable/disable, show status etc.

Request:{
}

```
Responses: {
  "jtype":"afb-reply",
  "request": {
    "status":"string" // success or bad-state or bad-request
  }
  "response": {
    "type":"array",
    "items" :
      [
        {
          "name":"string",
          "description":"string",
          "agent_id":"integer" // Voice agent ID
          "status":"string" // enabled, disabled
        }
      ]
  }
}
```

2. void voiceservice_hl_settings_api_set_active_state(struct afb_req req)

Activate or deactivate a voice agent.

"permission": "urn:AGL:permission:speech:public:accesscontrol"

```
 Request:
{
  "agent_id":"integer"
  "is_active":"boolean"
}

 Responses: {
  "jtype":"afb-reply",
  "request":
  {
    "status":"string" // success or bad-state or bad-request }
  }
}
```

afb-voiceservice-wakeword-detector

- Provides an interface primarily for the core **afb-voiceservice-core** to listen for wakeword detection events and make request routing decisions.
- This binding will internally talk to or host voice assistant vendor specific wake word solutions to enable the wake word detection.

Voice Agent Vendor Software

voice-agent-binding

- The API specification of voice agent is defined in this document. All the vendor specific voice agent bindings will follow the same specific to integrate with the high-level voice service.
- Voice Agent will listen to audio input when instructed by the high level voice service.
- Voice Agent will run its own automatic speech recognition, natural language processing, generates intents to perform requested action.
- Voice Agent will have it’s own authentication, connection and dialog management flows. And generates events to notify the high-level voice service of its state transitions.
- Voice Agent will use the high level voice service's interaction manager to command system applications to perform tasks, like Route to a specific geo code, Dial a Number, Play music etc.

API

1. void voiceagent_api_init(struct afb_req req)

This is called when the binding is loaded for the first **time**. In this method, the voice agent will need to perform the necessary initialization routines and subscribe its clients to all its events. The Alexa voice agent for example would instantiate the Alexa Auto Core, set up necessary interface implementations and provide path to custom configuration input.

"permission": "urn:AGL:permission:speech:public:accesscontrol"

Request: { }

Responses: {
 "jtype":"afb-reply",
 "request":{
  "status":"string" // success or bad-state or bad-request
 }
}

2. void voiceagent_api_start(struct afb_req req)

Starts the voice agent and passes the unique identifies that voice agent would provide in all its events and voiceservice_hl_api would refer to it.

"permission": "urn:AGL:permission:speech:public:accesscontrol"

Request:
{
 "agent_id":"integer"
}

Responses:
{
 "jtype":"afb-reply",
 "request":{
  "status":"string" // success or bad-state or bad-request
 }
}

3. void voiceagent_api_stop(struct afb_req req)

Stop the voice agent and its currently running processes. After calling Stop, no other APIs will work until Start is called again.

"permission": "urn:AGL:permission:speech:public:accesscontrol"

Request: {

}

Responses:

```
{
  "jtype":"afb-reply",
  "request":{
    "status":"string" // success or bad-state or bad-request
  }
}
```

4. void voiceagent_api_startListening(struct afb_req req)

Start the listening for speech input. As a part of request, common configuration related information is passed. Once the voice agent is ready it will respond with voiceagent-onstartcapture-event.
Note: The config inputs below are just examples and not the final list of configurations.

"permission": "urn:AGL:permission:speech:public:audiocontrol"
Request:

```
{
  "language":"string"
  "location":"string"
  "preferred_network_mode":"string" // online, offline, hybrid
  "speech_initiation_mode":"string" // TapToTalk, HoldToTalk
  "audio_input_path": "string" // name of the shared memory that contains audio input
}
```

5. void voiceagent_api_stopListening (struct afb_req req)

Stops listening for speech and requests the voice agent to stop processing the speech input. Note that in the default case, this does not need to be called, as the voice service end point will automatically stop the processing when it determines speech has completed or when speech recognition is completed.

"permission": "urn:AGL:permission:speech:public:audiocontrol"

Request:

```
{
}
```

Responses:

```
{
  "jtype":"afb-reply",
  "request":{
    "status":"string" // success or bad-state or bad-request
  }
}
```

Events

1. Voice agent will notify high-level voice service that its ready to received audio input.

Event Data:

```
{
  "name" : "voiceagent-onstartcapture-event"
  "agent_id": "integer"
}
```

2. Voice agent will notify its clients to stop capture immediately.

Event Data:

```
{
  "name" : "voiceagent-onstopcapture-event"
  "agent_id": "integer"
}
```

3. Voice agent will notify its clients that end of speech is detected.

Event Data:

```
{
  "name" : "voiceagent-onendofspeechdetected-event"
  "agent_id": "integer"
}
```

4. Dialog state describes the state of the currently active voice agent's dialog interaction.

Event Data:

```
{
  "name" : "voiceagent-dialogstate-event"
  "state":"string"
  "agent_id": "integer"
}
```

Values for state are,

IDLE: Voice Agent is ready for speech interaction.

LISTENING: Voice Agent is currently ready to listen for audio input.
THINKING: A customer request has been completed and no more input is accepted. In this state, Voice service is working on a response.
SPEAKING: Responding to a request with speech.

Domain Specific Flows

Note: The role of high level voice service binding is not depicted in the below flows for ease of understanding. All the flows are triggered assuming that user chose "hold to talk" to initiate the speech flow. There will slight modifications as explained in the high level workflow above if tap to talk or wake word options are used.

Climate Control (CC)

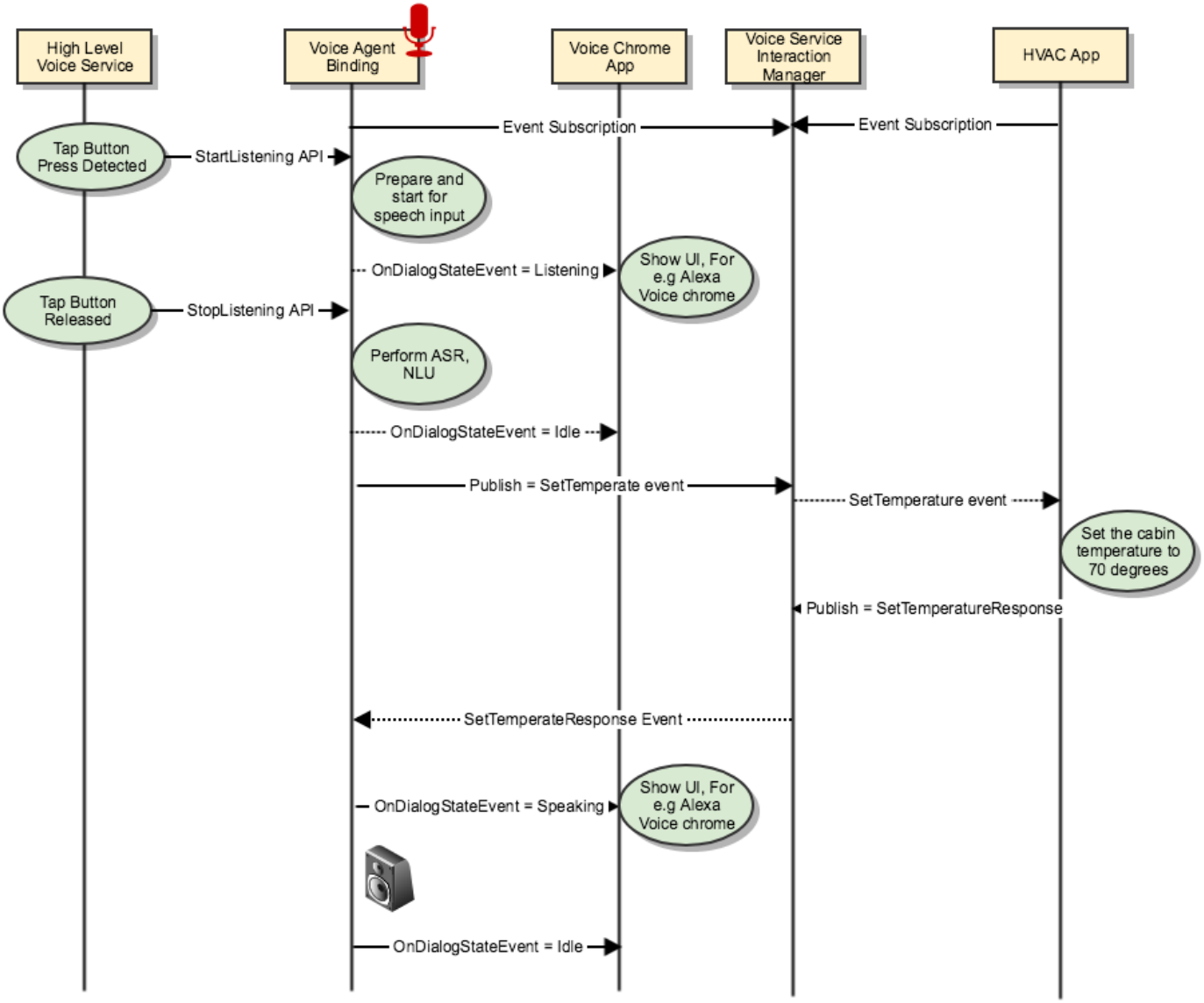
Use Case	Description
CC - on/off	Turn on or off the climate control (e.g. turn off climate control)
CC - specific temperature	Set the car's temperature to 70 degrees (e.g. set the temperature to 70)
CC.- target range	Set the car's heating to a set gradient (e.g. set the heat to high)
CC - min / max temperature	Set the car's temperature to max or min A/C (or heat) (e.g. set the A/C to max)
CC - increase / decrease temperature	Increase or decrease the car's temperature (e.g. increase the temperature)
CC - specific fan speed	Set the fan to a specific value (e.g. set the fan speed to 3)
CC - target range	Set the fan to a specific value (e.g. set the fan speed to high)
CC - Temp Status	What is the current temperature of the car (e.g. how hot is it in my car?)

CC - Fan Status	Determine the fan setting (e.g. what's the fan set to?)

Set the cabin temperature

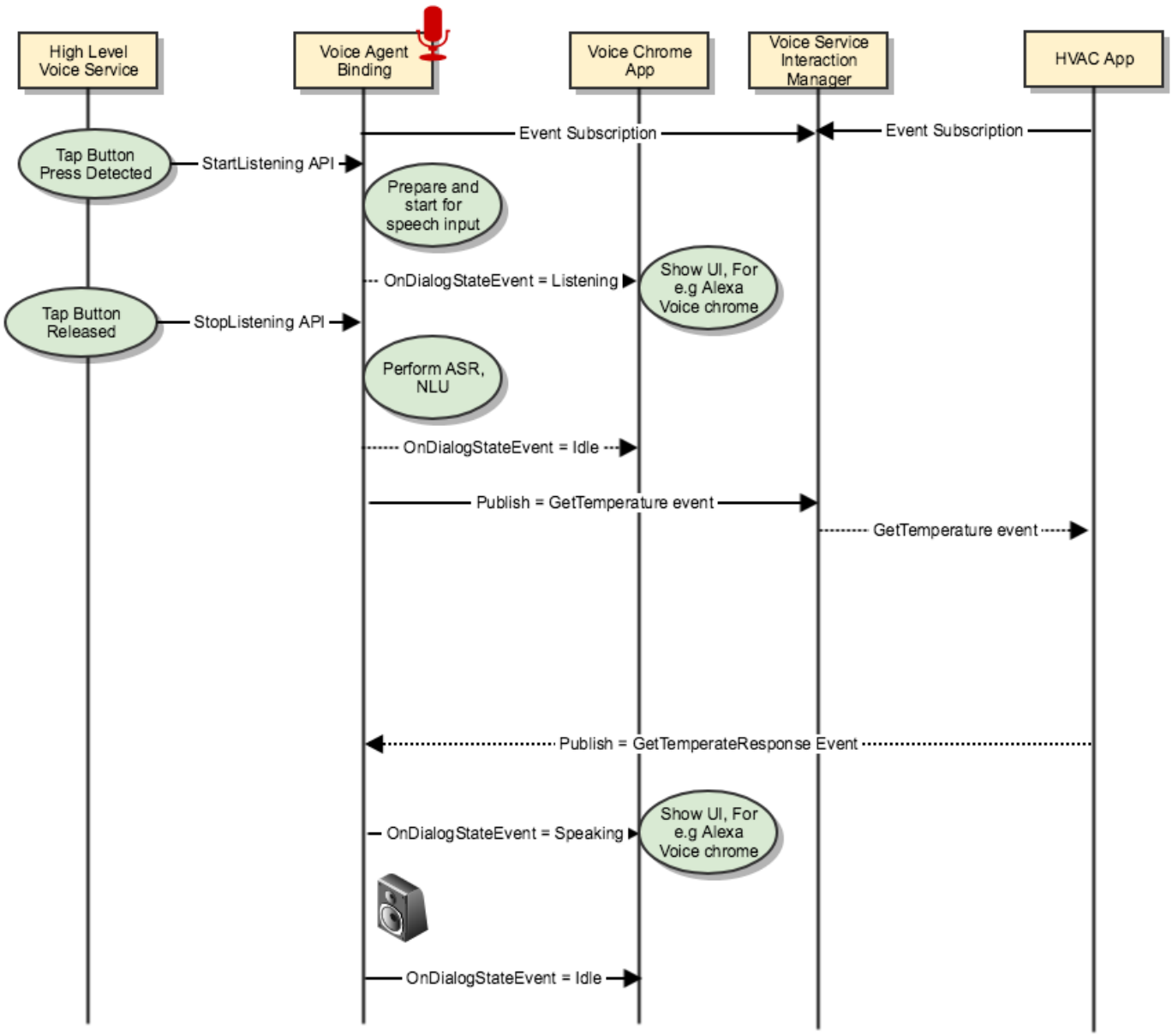
"Set the cabin temperature to 70 degrees"

"Set the car's temperature to max or min A/C (or heat)



Get the cabin temperature

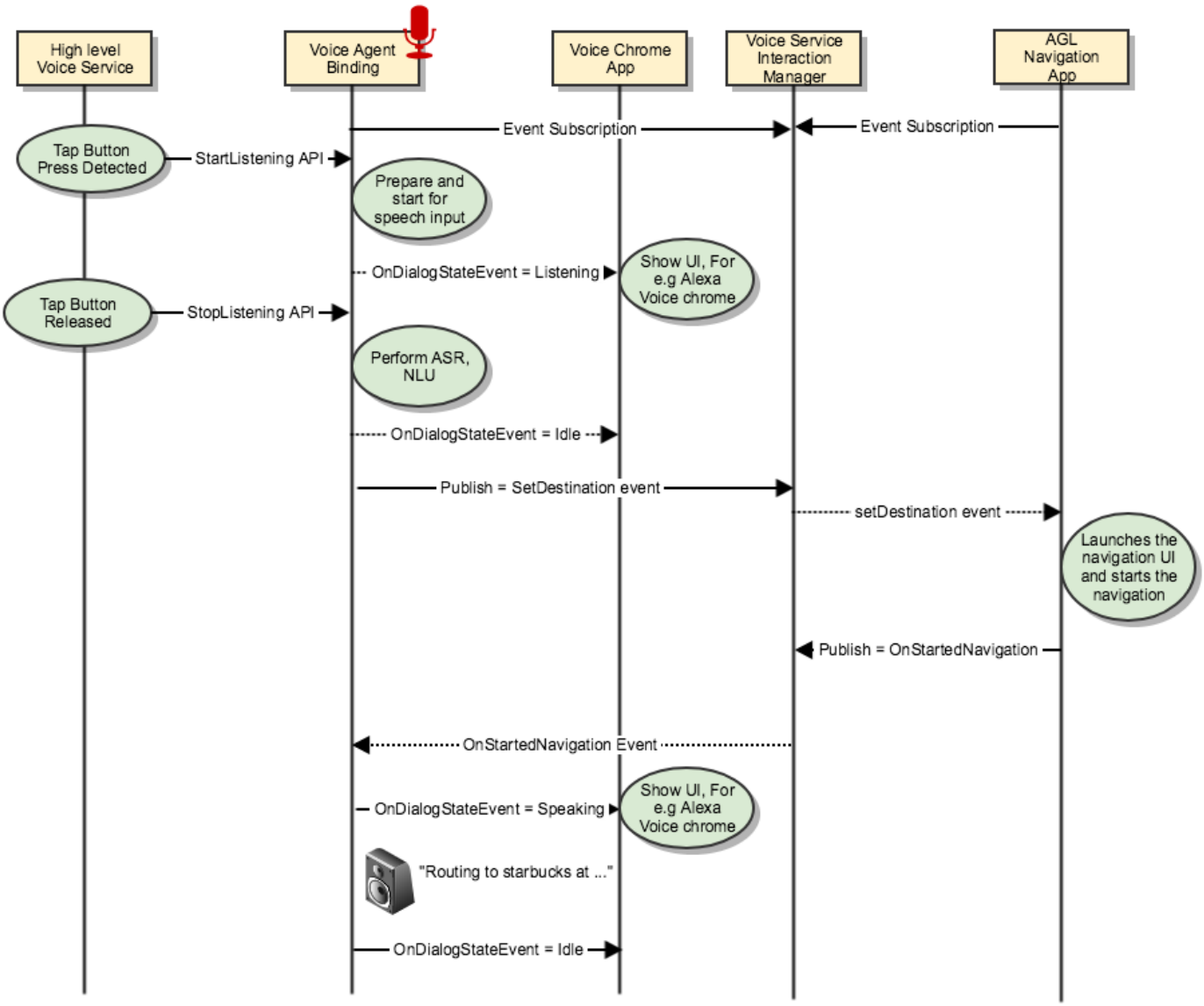
how hot is it in my car?



Navigation

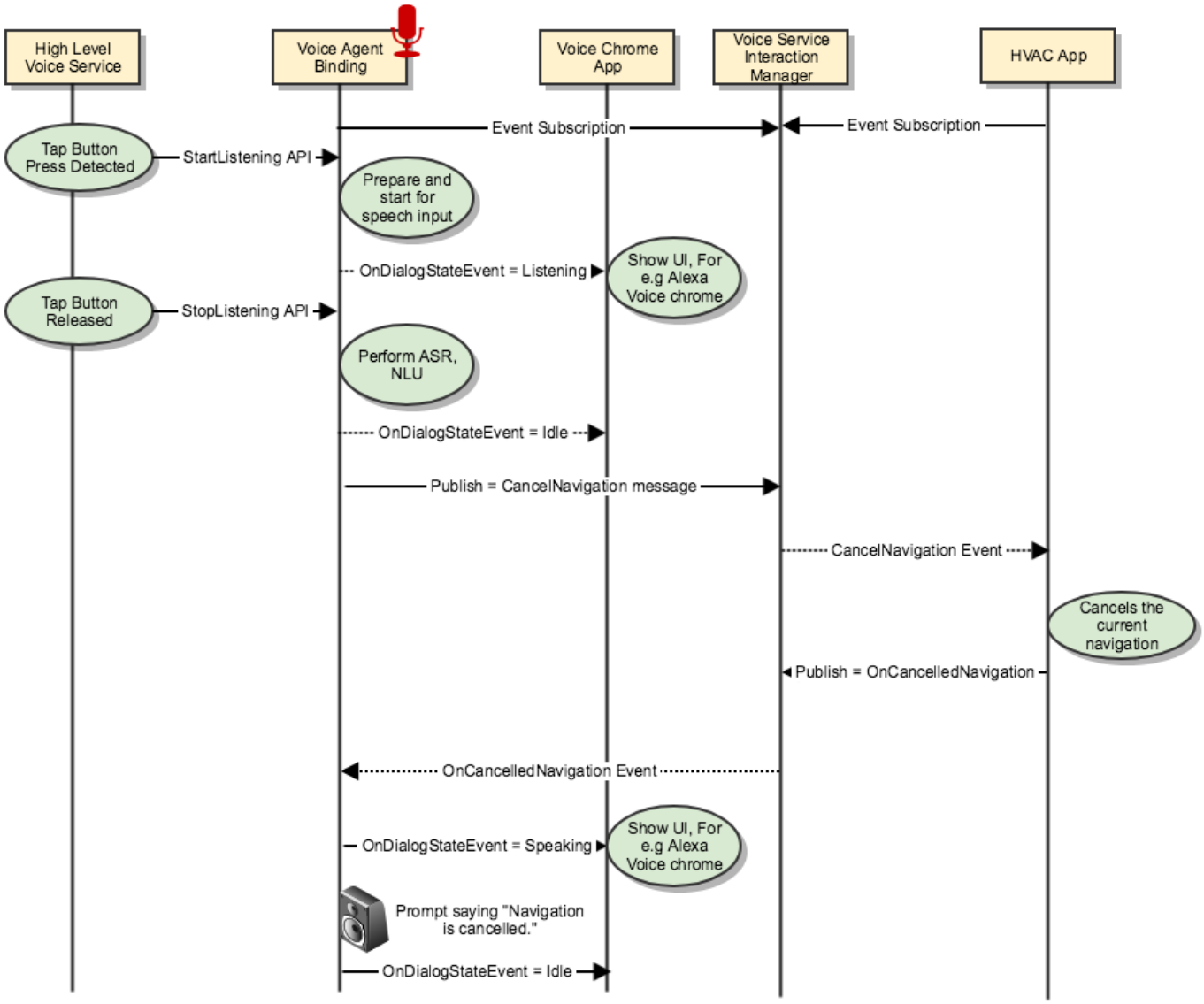
Set Destination	Notify the navigation application to route to specified destination. For e.g "Navigate to nearest star bucks" "Navigate to my home"
Cancel Navigation	Cancel the navigation based on touch input or voice. For e.g User can say "cancel navigation" User can cancel the navigation by interacting with the navigation application directly on the device using Touch inputs.
Suggest Alternate Route	Suggest an alternate route to the user and proceed as per user preference. For e.g. "There is an alternate route available that is 4 minutes faster, Do you wish to select?" When user says "No", then continue navigation when user says "Yes", then proceed with navigation.

Set Destination



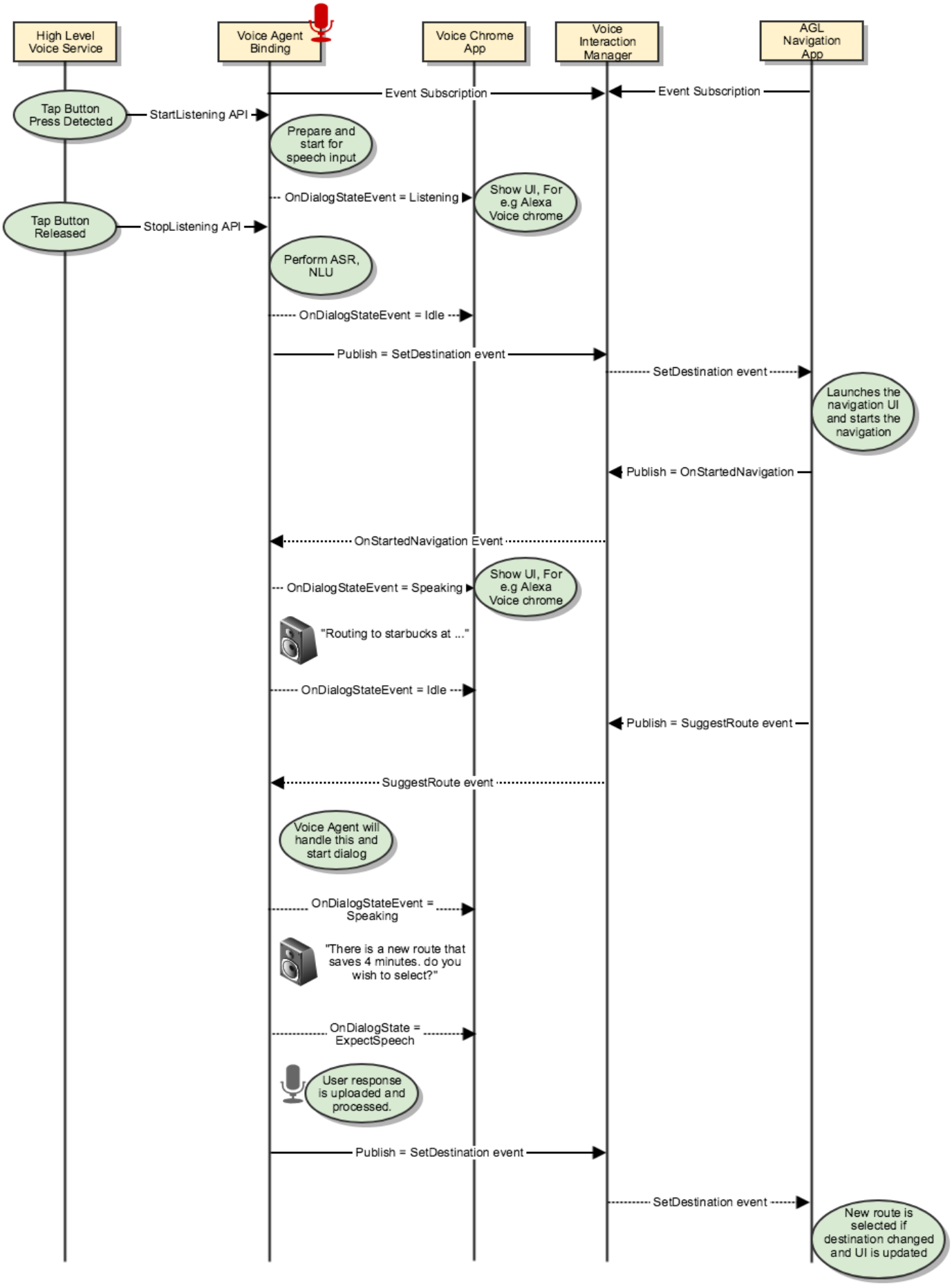
Cancel Navigation

Voice initiated



Navigation App has a alternate route

Below is a high level proposal on how the interaction is supposed to work when system application want to initiate a dialog through voice agents. Alternatively, AGL navigation app can use STT and TTS API (out of scope for this doc) with some minimal NLU to implement the same use case.


High Level Voice Service
Voice Agent Binding
Voice Chrome App
Voice Interaction Manager
AGL Navigation App
Event Subscription
Event Subscription
Tap Button Press Detected
StartListening API
Prepare and start for speech input
OnDialogStateEvent = Listening
Show UI, For e.g Alexa Voice chrome
Tap Button Released
StopListening API
Perform ASR, NLU
OnDialogStateEvent = Idle
Publish = SetDestination event
SetDestination event
Launches the navigation UI and starts the navigation
Publish = OnStartedNavigation
OnStartedNavigation Event
OnDialogStateEvent = Speaking
Show UI, For e.g Alexa Voice chrome
"Routing to starbucks at ..."
OnDialogStateEvent = Idle
Publish = SuggestRoute event
SuggestRoute event
Voice Agent will handle this and start dialog
OnDialogStateEvent = Speaking
"There is a new route that saves 4 minutes. do you wish to select?"
OnDialogState = ExpectSpeech
User response is uploaded and processed.
Publish = SetDestination event
SetDestination event
New route is selected if destination changed and UI is updated

FAQ

1) How will the initial connection be established between a new voice-agent-binding and voice-service-hl-binding?

There are three possible options to establish the initial connection

1. voice-service-hl-binding service can search and discover different voice agents if App framework provided a mechanism to do so.
2. The new voice-agent-binding can register itself with voice-service-hl-binding using an API call assuming that there is a runtime support to add the **required-api** config to the speech-recognizer-low service. Refer to http://docs.automotivelinux.org/docs/apis_services/en/dev/reference/af-main/2.2-config.xml.html on how to add it in the package manifest. Also we don't prescribe this in order to avoid creating strong cyclic dependency between bindings.
3. Manually add the dependencies of each voice-agent-binding in the config of voice-service-hl-binding. This is our preferred approach since service discovery is not supported by AGL application framework.

2) Should the voice agent create an audio input channel instead of getting the audio attachment from the voice service layer?

That discussion is definitely on cards. The only downside of each voice agent opening an audio input stream is that the use case of routing the speech request to voice agents based on keyword detection can't be achieved. We will refine as we get more inputs and feedback from the group.

3) What about TTS (Text to Speech) ?

Text to Speech is internal to a voice agent implementation. If a specific use case requires a TTS response, then the corresponding voice agent is responsible for generating the audio file with TTS response and its playback.

4) Who will maintain the AGL high-level service binder and app integration?

Engineers from Konsulko group (https://www.konsulko.com/) will help Amazon and Nuance implement and maintain the high-level service binding, support bindings and AGL app integration.