Speech EG Architecture

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 AGL Speech framework, and evolve the same into a full blown multi-agent architecture that car OEMs use to enable voice experiences of their choice.

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 goes with other voice agent vendors.

Customer Requirements

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, whats the weather" and "whats the weather" should be routed to the user selected voice agent.

- **Experience B: Multiple active voice agents use cases.**
    - **Invocation**
        - **Explicit Voice : 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, whats the weather", my utterance should be routed to Alexa voice agent on the device.

        - **Implicit Voice : Speech framework will pick the appropriate agent based on intent detection to perform a task.**
          The speech request should be routed to the best agent responsible for the handling the intent.
            - I should be able to assign appropriate agents to select set of tasks like Navigation, Calling, News, Car control, Local Music, Calendar etc.
            - If I don't explicitly assign an agent to a task, then the system should route my utterance to the best agent capable of handling the request.

        - **Multi Modal Interaction**
            - (U) Agent A, Route me Starbucks nearby
            - A list Starbucks nearby are presented to user.
            - (U) Agent A says Select first one OR 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.
  - 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
  - Agent switching based on intent detection
    - (User) Book flight ticket to Seattle
    - (Agent A) What time you want to leave?
    - (User) What time my last meeting ends on Mon
    - (Agent B) 5:30
    - (User) Alright, 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.

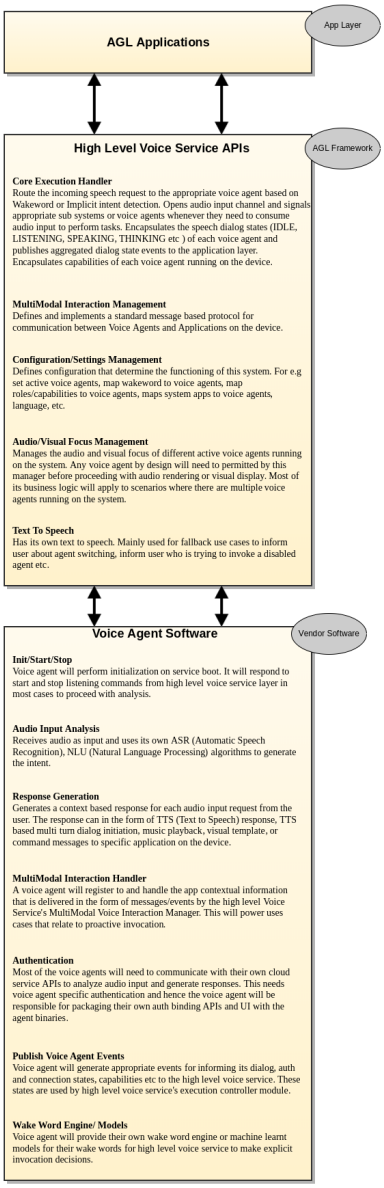
Customer D: As a 3rd party Voice Agent Vendor,

- I should be able to follow the guidelines of the AGL speech framework to plugin my voice agent.
- I should be able to follow the guidelines of the AGL speech framework to plugin my wake word solution.
- I should be able to follow the guidelines of the AGL speech framework to plugin my NLU engine.

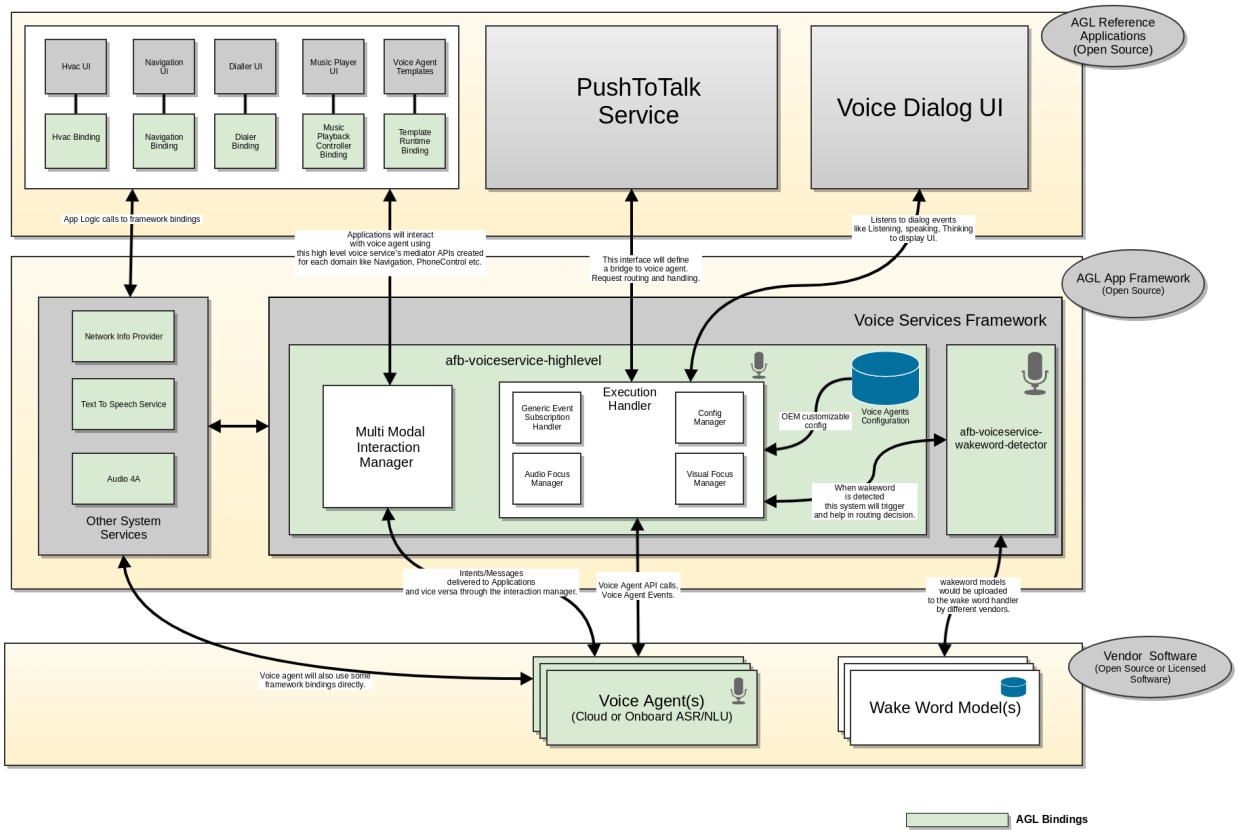
High Level Architecture

Quoting AGL documentation,
http://docs.automotivelinux.org/docs/apis_services/en/dev/reference/signaling/architecture.html#architecture“
“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.”

High Level Components



Architecture



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 implementation 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 afbvoiceservice-highlevel. afb-voiceservice-highlevel binding has subscribed to all the events of active voiceagent-binding and vice versa. afb-voiceservice-highlevel is assigned the audio input role by the audio high level binding.

- afb-voiceservice-highlevel **startListening** API will be triggered by Push-To-Talk button invocation.
- afb-voiceservice-highlevel signals afb-voiceservice-wakeword-detector binding running in same binder context to **startListening.** The wakeword detector binding uses PCM APIs to read the audio input.
- afb-voiceservice-highlevel 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-highlevel will wait for a few milliseconds for the afb-voiceservice-wakeword-detector to come back with **on WakeWordDetected** event.
    - If afb-voiceservice-wakeword-detector doesn't detect wake word within the aforementioned timeout, then afb-voiceservice-highlevel will pick the active agent selected by user to process the audio input.
    - If afb-voiceservice-wakeword-detector detects the wake word, then afb-voiceservice-highlevel will get onWakeWordDetected event with the offset of position from which the voice agent should start buffering the audio & with the wake word string that was detected.
        - Then afb-voiceservice-highlevel will call the **startListening** API of the active voice-agent-binding. Along with that it also passes the offset buffering location and the detected wakeword.
- 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-highlevel.
- 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 either cloud based or onboard 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-highlevel 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 domain specific multimodal interaction manager APIs of afb-voiceservice-highlevel binding.
    - If there is an associated UI card to be displayed, voice-agent-binding will send another message to the card rendering application using tmultimodal interaction manager APIs of afb-voiceservice-highlevel binding.

Bindings

The Voice Services architecture in AGL is layered into two levels. They are High Level Voice Service component and vendor software components with Voice Agents and Wake Word detection solutions. In the above architecture, the high level voice service is composed of multiple bindings (colored in green) that will be part of the AGL framework. And the vendor software layer composes of voice agent binding and wake word binding that hosts the vendor specific voice assistant software. The system provides flexibility to voice assistant vendors to provide their software as code or binary as long as they abide by the Voice Agent API specifications.

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.

- Tap-To-Talk Mode
    - In this mode, the user will need to press the tap-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 default 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.

1) agl-service-voice-high

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 is responsible for routing the utterance to the correct voice agent based on various parameters like configuration, wake word detection etc.
- Registers for dialog, connection, auth etc events from voice agents. 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 scenario, 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.

Current Architecture

The following diagrams dives a litter deeper into the low level components of high level voice service (VSHL)

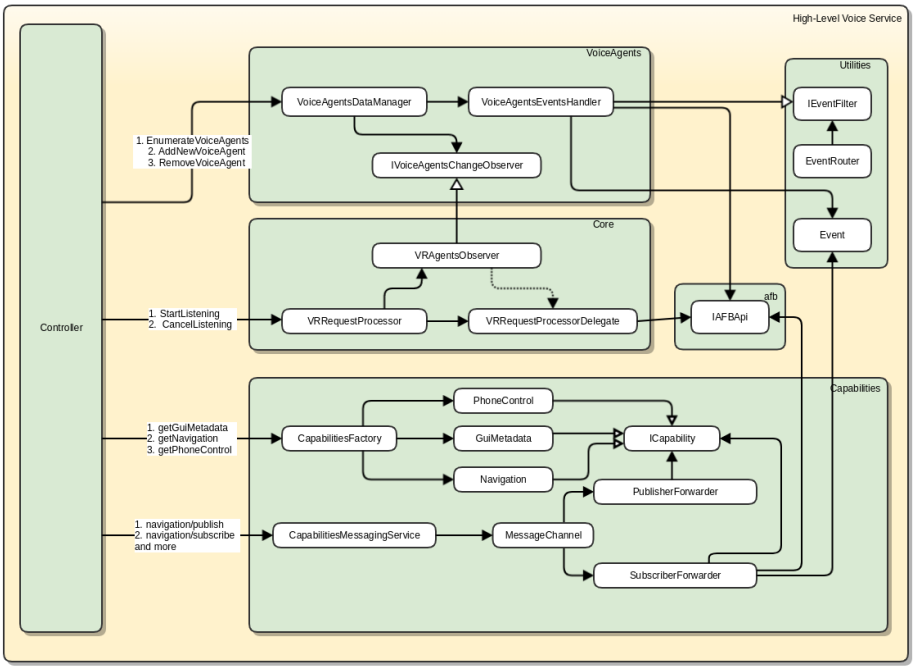
and their dependencies depicted by directional arrows. The dependency in this case can be either through an

association of objects between components or through an interface implementation relationships.

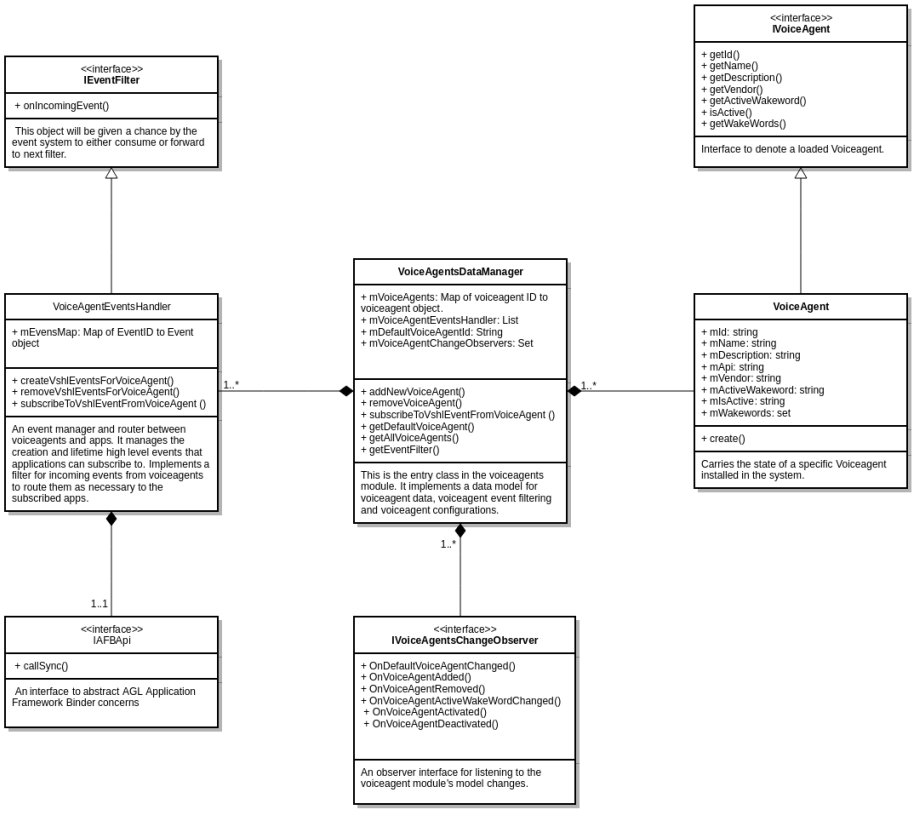
For e.g.,

A depends on B if A aggregates or composes B.

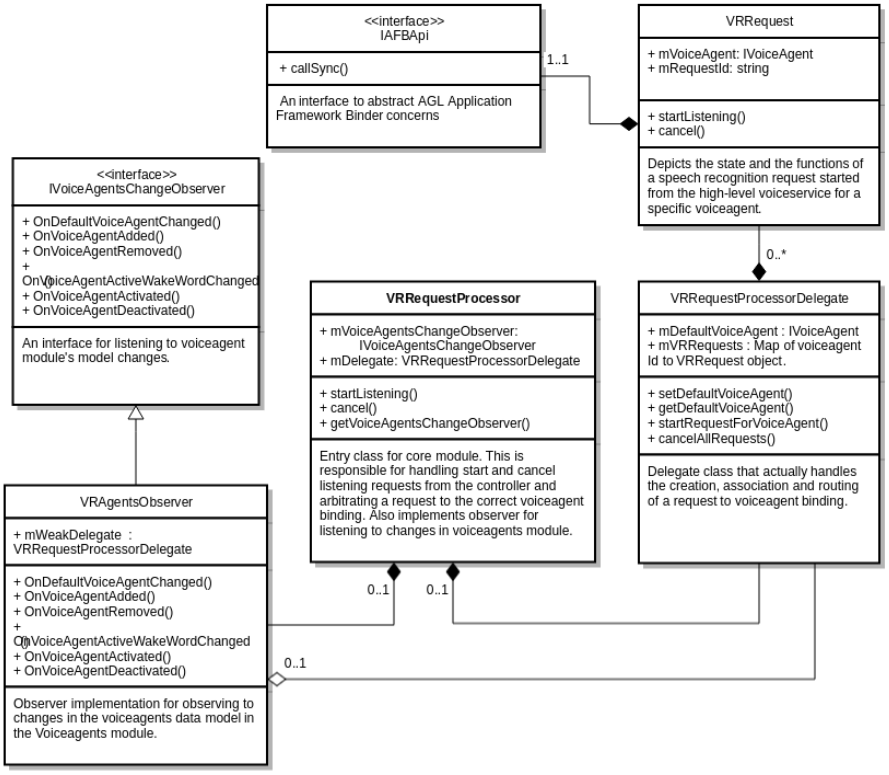
A depends on B if A implements an interface that is used by B to talk to A.



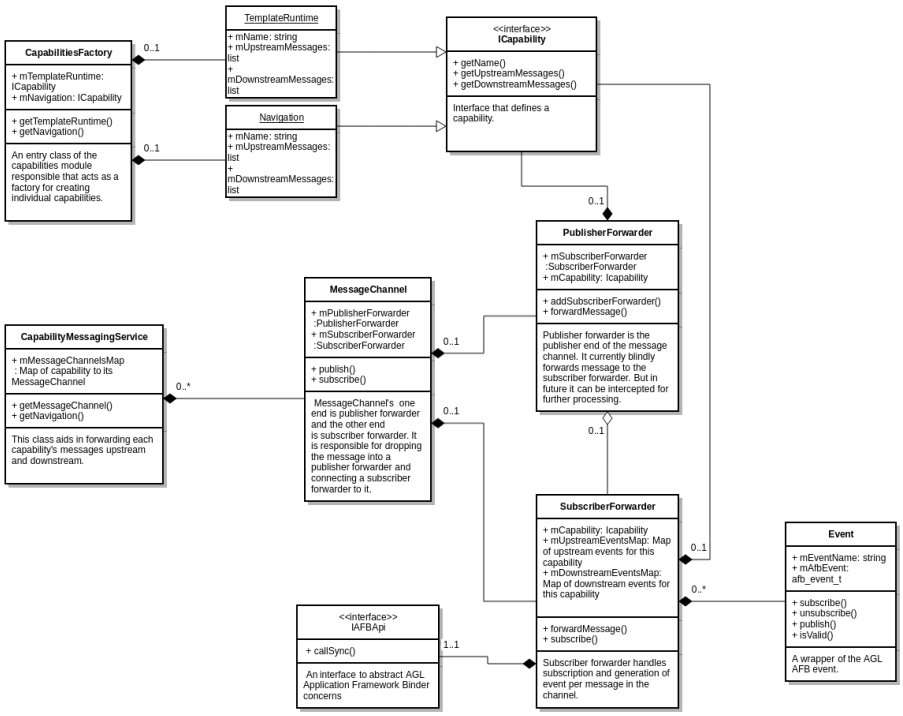
VoiceAgents Module



Core Module



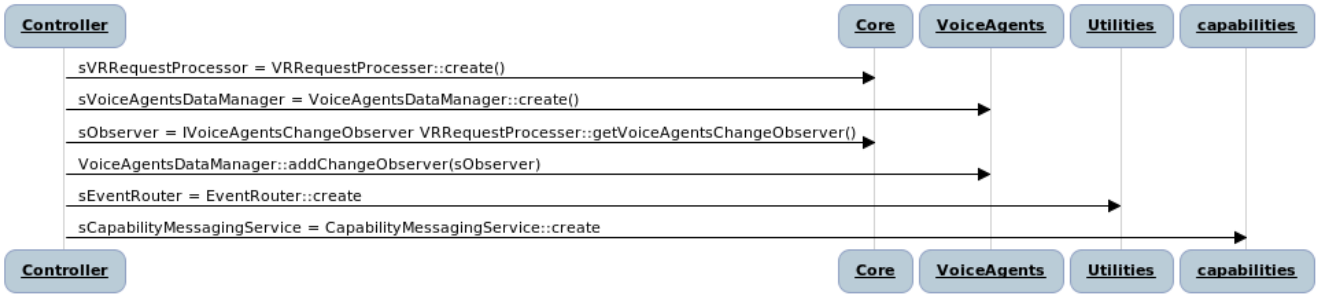
Capabilities Module



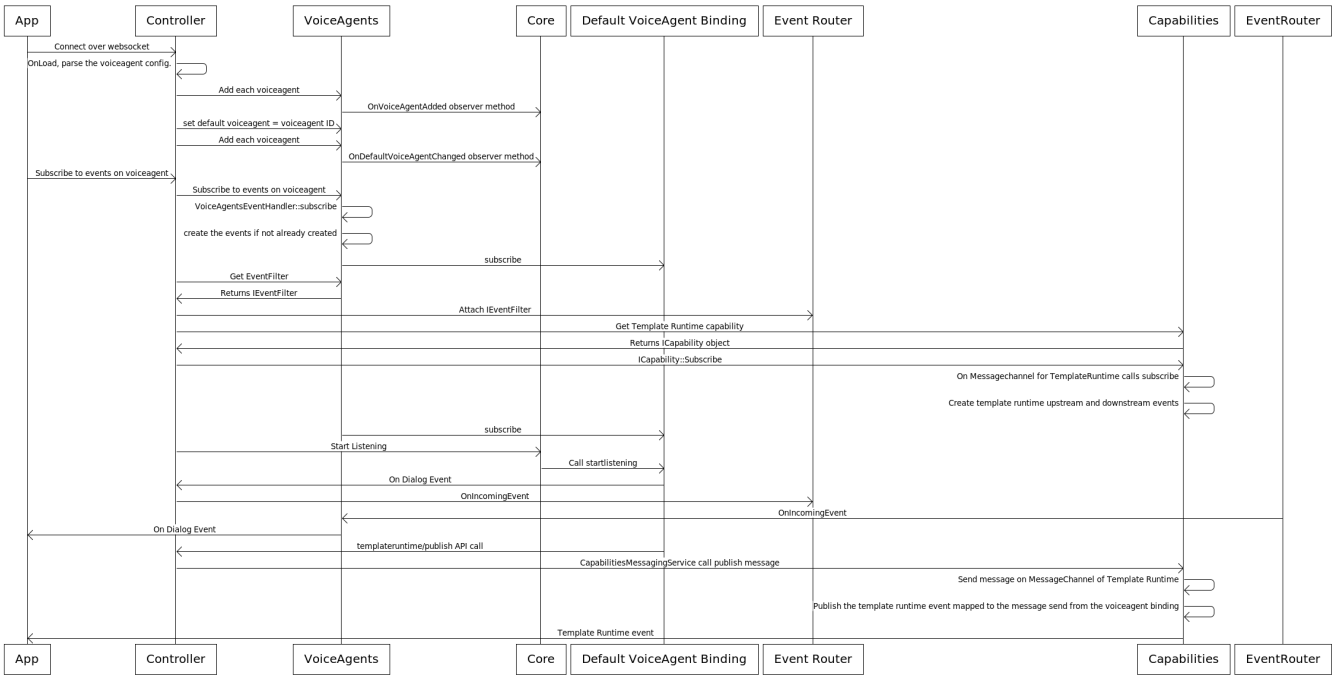
Sequence Diagrams

OnLoad

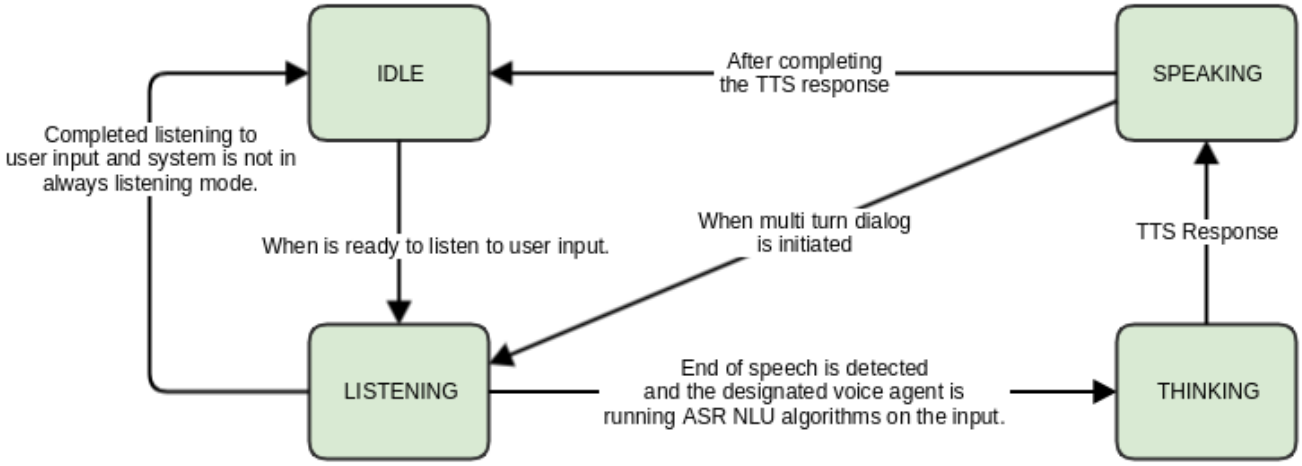
On load the controller will instantiate the entry level classes of each module and inject their dependencies. For e.g Core module observers changes to voiceagent data in VoiceAgent module.



StartListening to Audio Input & Events



State Diagram



API

vshl/startListening

```
vshl/startListening


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.


Request: {

}


Responses: {
  "jtype":"afb-reply",
  "request": {
    "status":"string" // success or bad-state or bad-request
  }
  "response":{
    "request_id": "string" // Request created by this call.
    "agent_id": "string" // Agent to which the request has been proxied.
  }
}
```

vshl/cancel

```
vshl/cancelListening


Cancels the speech recognition processing in the chosen agent.
If agent id is not passed then the cancel request is sent to the default voice agent.


Request:
{

}

Responses:
{
```

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

vshl/subscribe

```
vshl/subscribe

Subscribe/Unsubscribe to voice service high level events.

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

Request:
{
  {
    "type":"array",
    "items" : [{
        "type":"string" // List of events to subscribe to
      }
    ]
  },
  {
    "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.

vshl_dialogstate_event

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

Event Data:
{
  "name" : "voice_dialogstate_event"
  "state":"string"
  "agent_id": "integer"
}


Values for state are
1) IDLE
High level voice service is ready for speech interaction.

2) LISTENING
High level voice service is currently listening.
```

```
3) THINKING
A customer request has been completed and no more input is accepted. In this state, Voice service is working on
a response.

4) SPEAKING
Responding to a request with speech.
```

vshl_connectionstate_event

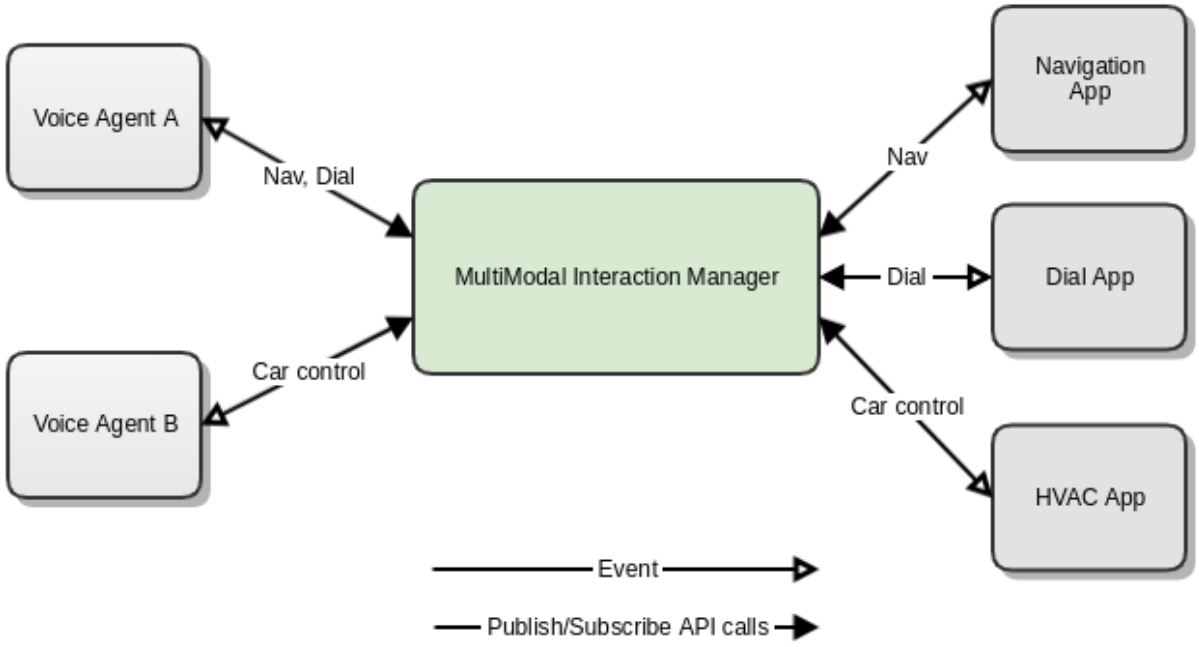
```
Connection state describes the state of the voice agent along with errors.

Event Data:
{
  "name" : "voice_connectionstate_event"
  "state":"string"
  "agent_id": "integer"
}

1) DISCONNECTED
Voice agent is not connected to its voice service endpoint.

2) PENDING
Voice agent is attempting to establish connection to its endpoint.

3) CONNECTED
Voice agent is connected to its endpoint.

4) CONNECTION_TIMEDOUT
Voice agent connection attempt failed due to excessive load on its server endpoint.

5) CONNECTION_ERROR
Captures other network related errors.
```

vshl_authstate_event

```
Auth state describes the state of the authorization of the voice agent with its cloud endpoint.

Event Data:
{
  "name" : "voice_authstate_event"
  "state":"string"
  "agent_id": "integer"
}

1) UNINITIALIZED
Authorization not yet acquired.


2) REFRESHED
Authorization has been refreshed.

3) EXPIRED
Authorization has expired.


4) ERROR
Authorization error has occurred.
```

2) Multi Modal Interaction Manager

An important part of the afb-voiceservice-highlevel binding, that acts as a mediator 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.

RequestId: "{{STRING}}" // The request ID associated with this message.

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.

DIAL

API

vshl/phonecontrol/publish - For publishing phone control messages below.

vshl/phonecontrol/subscribe - For subscribing to phone control messages below.

Messages

Upstream

{

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"]

}

}

{

Topic : "phonecontrol"

Action : "connection_state_changed"

Payload : {

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

"required": ["callId"]

}

}

NAVIGATION

API

vshl/navigation/publish - For publishing navigation messages.

vshl/navigation/subscribe - For subscribing to navigation messages.

Messages

Upstream

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

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

GuiMetadata

API

vshl/guimetadata/publish - For publishing ui metadata messages for rendering.

vshl/guimetadata/subscribe - For subscribing ui metadata messages for rendering.

Messages

Upstream

{

Topic : "guimetadata"

Action : "render_template"

Payload : {

<Yet to be standardized>

}

}

{

Topic : "guimetadata"

Action : "clear_template"

Payload : {

<Yet to be standardized>

}

}

{

Topic : "guimetadata"

Action : "render_player_info"

Payload : {

<Yet to be standardized>

}

}

{

Topic : "guimetadata"

Action : "clear_player_info"

Payload : {

<Yet to be standardized>

}

}

3) Configuration

- 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
- Enable/Disable Fallback Invocation mode
- Enable/Disable Agent Switching during multi turn dialog
- ... more

API

vshl/enumerate_agents

```
vshl/enumerateVoiceAgents


"permission": "urn:AGL:permission:vshl:voiceagents:public"


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
        }
      ]
  }
}
```

vshl/setActive

```
vshl/setDefaultVoiceAgent

Activate or deactivate a voice agent.

"permission": "urn:AGL:permission:vshl:voiceagents:public"


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-highlevel** 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

1) 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 its 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

voiceagent/setup

```
voiceagent/setup

This API is exposed to high level voice service to pass any setup or high level config information like
agent_id to the voice agent.

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

Request:
{
  "agent_id":"integer"
  "language":"string"
}


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

voiceagent/cancel

```
voiceagent/cancel

Stop the voice agent and its currently running speech recognition processes.


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


Request:
```

```
{

}


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

voiceagent/startListening

```
voiceagent/startListening

Start the 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:
{
  "request_id": "string" // Request ID assigned by the high level voice service.
  "language":"string"
  "location":"string"
  "preferred_network_mode":"string" // online, offline, hybrid
  "audio_input_device": "string" // ID of the alsa device to read the input
}


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

Events

voiceagent_endofspeechdetected_event

```
Voice agent will notify its clients that end of speech is detected.
Event Data:
{
  "name" : "voiceagent_endofspeechdetected_event"
  "agent_id": "integer"
  "request_id": "integer" // the request for which the end of speech is detected
}
```

voiceagent-dialogstate-event

```
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"
  "request_id": "string" // The request that caused this dialog state transition.
}


Values for state are
1) IDLE
High level voice service is ready for speech interaction.

2) LISTENING
High level voice service is currently listening.

3) THINKING
A customer request has been completed and no more input is accepted. In this state, Voice service is working on
a response.

4) SPEAKING
Responding to a request with speech.
```

voiceagent_connectionstate_event

```
Connection state describes the state of the voice agent along with errors.

Event Data:
{
  "name" : "voiceagent_connectionstate_event"
  "state":"string"
  "agent_id": "integer"
}

1) DISCONNECTED
Voice agent is not connected to its voice service endpoint.

2) PENDING
Voice agent is attempting to establish connection to its endpoint.

3) CONNECTED
Voice agent is connected to its endpoint.

4) CONNECTION_TIMEDOUT
Voice agent connection attempt failed due to excessive load on its server endpoint.

5) CONNECTION_ERROR
Captures other network related errors.
```

voiceagent_authstate_event

```
Auth state describes the state of the authorization of the voice agent with its cloud endpoint if any.

Event Data:
{
  "name" : "vshl_authstate_event"
  "state":"string"
  "agent_id": "integer"
}

1) UNINITIALIZED
Authorization not yet acquired.
```

```
2) REFRESHED
Authorization has been refreshed.

3) EXPIRED
Authorization has expired.


4) ERROR
Authorization error has occurred.
```

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)

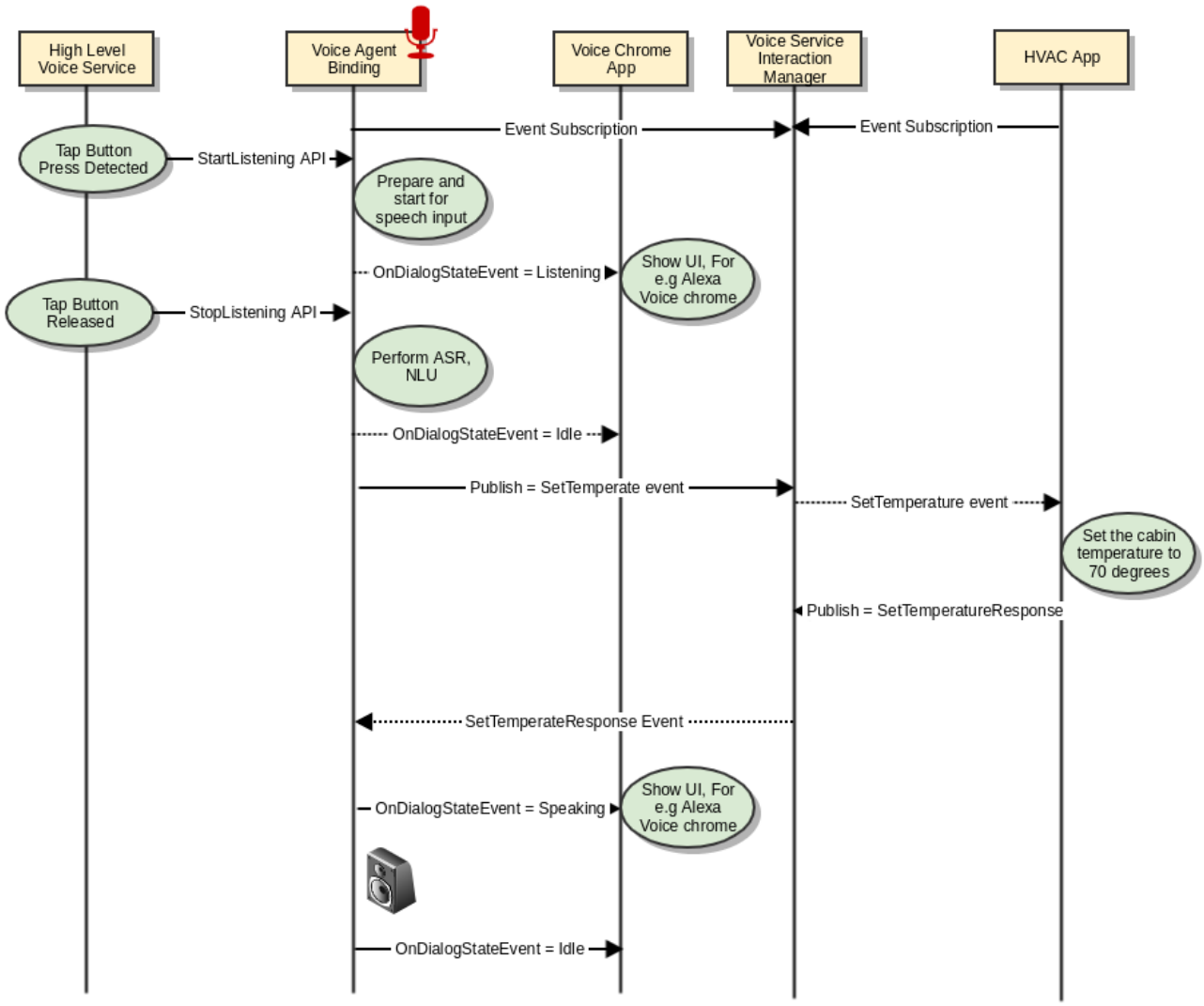
Use cases

1	CC - on/off	Turn on or off the climate control (e.g. turn off climate control)
2	CC - specific temperature	Set the car's temperature to 70 degrees (e.g. set the temperature to 70)
3	CC.- target range	Set the car's heating to a set gradient (e.g. set the heat to high)
4	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)
5	CC - increase / decrease temperature	Increase or decrease the car's temperature (e.g. increase the temperature)
6	CC - specific fan speed	Set the fan to a specific value (e.g. set the fan speed to 3)
7	CC - target range	Set the fan to a specific value (e.g. set the fan speed to high)
8	CC - min / max fan speed	Set the fan to min / max (e.g. set the fan to max)
9	CC - increase / decrease fan speed	Increase / decrease the fan speed (e.g. increase the air flow)
10	CC - Temp Status	What is the current temperature of the car (e.g. how hot is it in my car?)
11	CC - Fan Status	Determine the fan setting (e.g. what's the fan set to?)

Set the cabin temperature

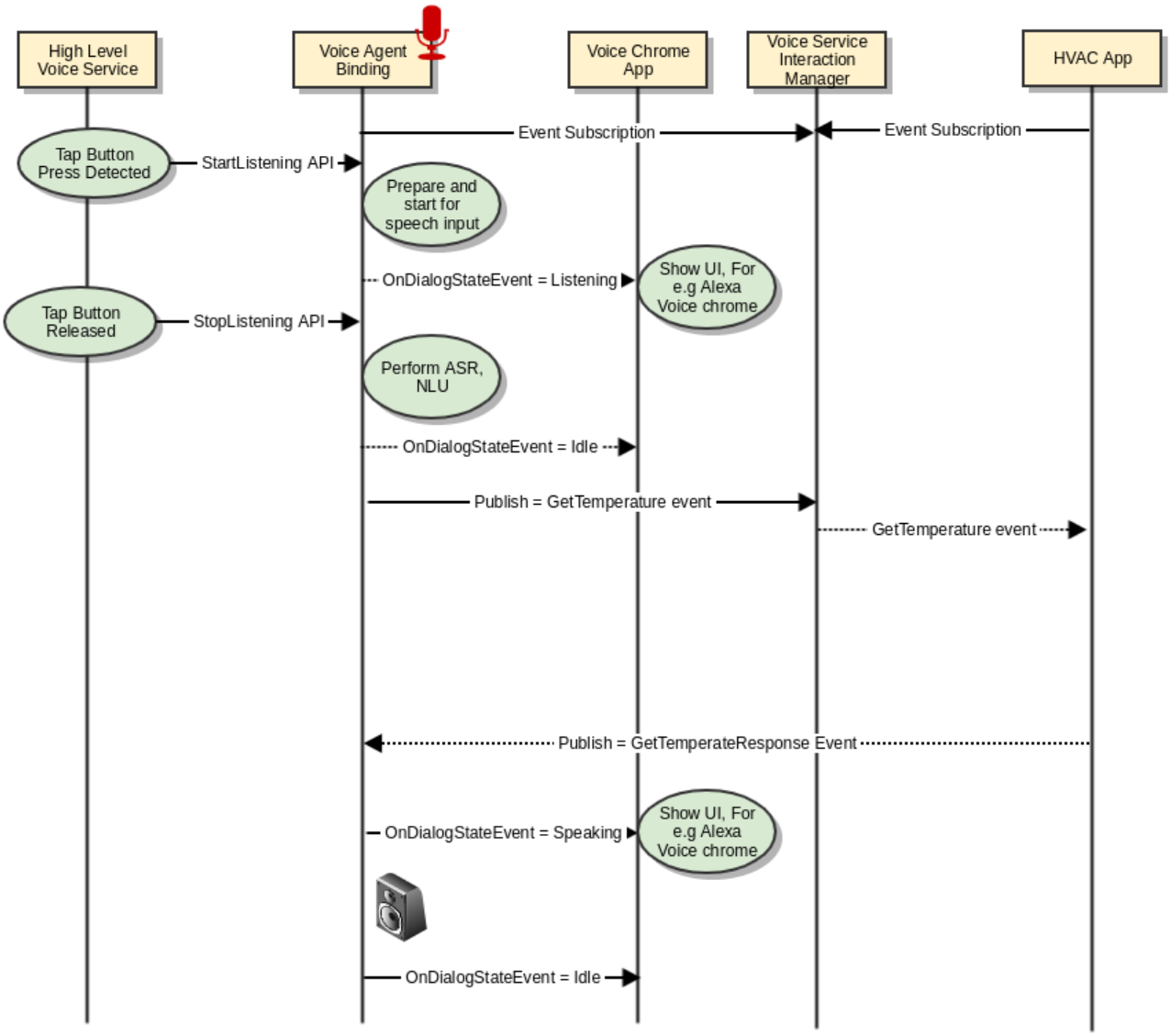
"Set the cabin temperature to 70 degrees"

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



Get the cabin temperature

how hot is it in my car?



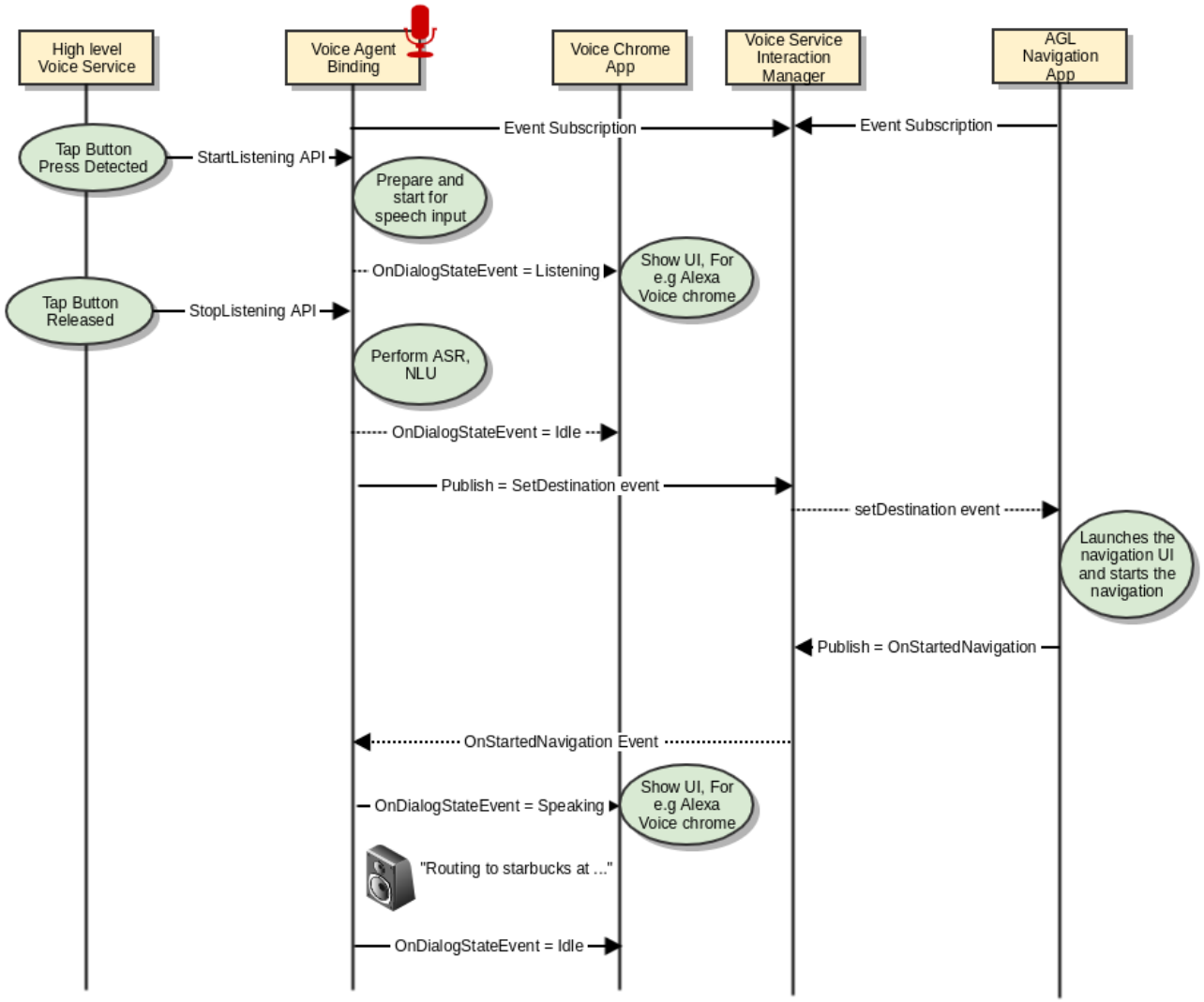
Navigation

Use cases

1	Set Destination	Notify the navigation application to route to specified destination. For e.g "Navigate to nearest star bucks" "Navigate to my home"
2	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.
3	Suggest Alternate Route	Suggest an alternate route to the user and proceed as per user preference. For e.g.

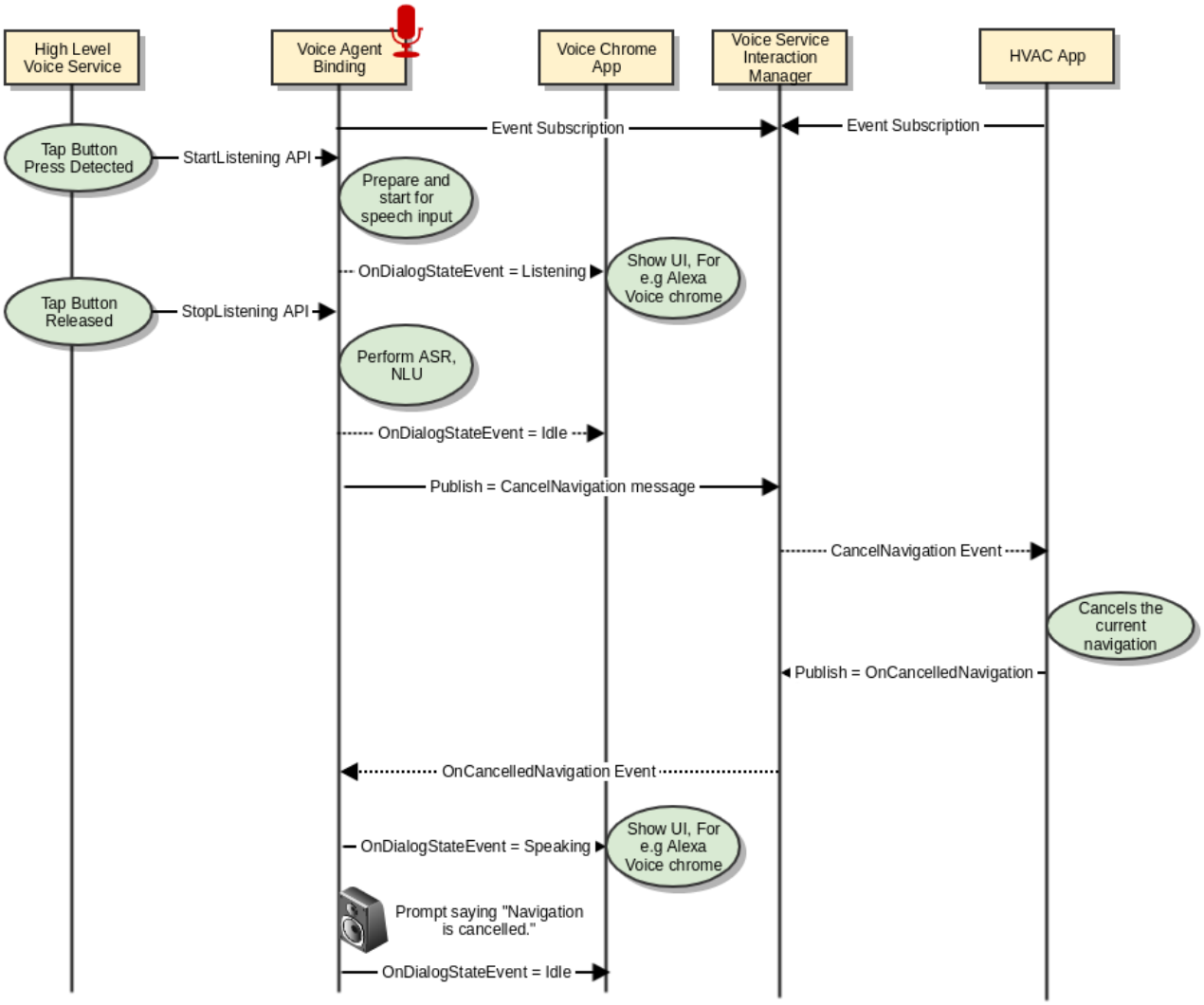
		"There is an alternate route available that is 4 minutes faster, Do you wish to select?" When user says "No", then continue navigation when user says "Yes", then proceed with navigation.

Set Destination



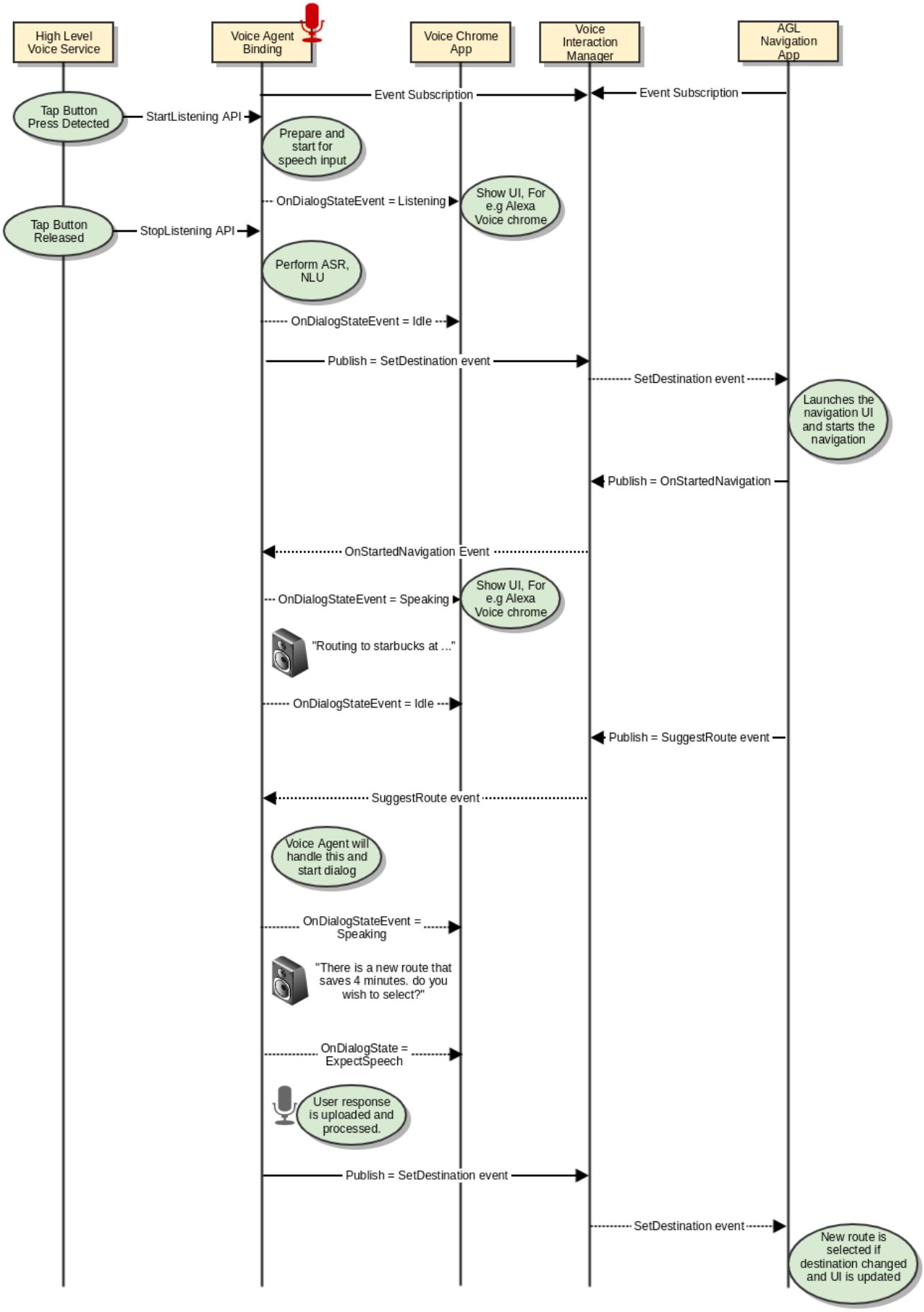
Cancel Navigation

Voice initiated



Select Alternate Route

This use case is currently unsupported by Alexa. Its a high level proposal on how the interaction is supposed to work. Alternatively, AGL navigation app can use STT and TTS API (out of scope for this doc) with some minimal NLU to enable similar behavior.


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

Technical References & Demos

Technical Video Presentation of AGL Speech Framework High-Level Architecture and Live Demo with Alexa Integration.

Alexa Demo on Renesas board.