Case Study: Vehicle Data Architecture for Connected Car Services at Hyundai Motor Company

Hyundai Motor Company
Seung-jae Lee (seungjae.lee@hyundai.com)
October 2022
Agenda

1. Introduction
   - ccOS Overview
   - Vehicle Service and ccos::HVehicle API

2. ccOS VSM (Vehicle Signal Model)
   - Mission of VSM
   - Key Feature
   - VSM Details
   - VSM Case Study #1 / #2

3. VSM for CCS
   - Introduction to CCS
   - CCS VSM
   - Key Feature

4. Vehicle Data Architecture

5. Q&A
ccOS Overview

- **Connected Car Strategy**
  - We are producing 8 million vehicles per year, and we aim to connect all cars by 2025.

- **What is the ccOS?**
  - We started developing ccOS *(Connected Car OS)* in 2016 to build a connected car service ecosystem and prepare for the future SDV environment.
  - The ccOS based infotainment system was introduced to the market through the Genesis G80/GV80.

### Past Infotainment Platform

- BlackBerry QNX
- Android Gingerbread
- Windows CE
- MeeGo

### Future Infotainment Platform

- ccOS

---

*Image of Genesis cars and Genesis logo.*
ccOS Overview

• ccOS Architecture Design Attributes
  − Layered Architecture
    ▪ to enhance S/W component reusability and hardware portability
    ▪ support for ARM-based processors and x86-based QEMU environments with HAL layer
  − Performance Considerations
    ▪ C++ native software components
  − Security Enhancement
    ▪ Linux kernel security module
    ▪ Strictly managing network resource
Vehicle Service and ccos::HVehicle API

- Vehicle Service
  - Provides abstracted IVN signals to the ccOS Apps as a generalized methodology
  - It covers that need to be handled by the headunit and signals that need to be interacted with the server

Headunit’s AP

ccOS Applications
- Climate App
- CCS App
- ...

ccOS Framework Layer
- ccos::HVehicle

Vehicle Service

ccOS HAL Layer
- libvehiclecan-hal.so
- libvehiclegemu-hal.so

OS Environments
- CAN Micom daemon
- QEMU Daemon

UART
- Micom (CAN Transceiver)
Vehicle Service and ccos::HVehicle API

- ccOS::HVehicle API
  - Our legacy vehicle APIs
    - Define all vehicle signals as C++ classes statically...
    - From a semantic perspective, all signals on the vehicle are defined as classes
    - Difficult to automate to build a test suite, which requires a lot of effort.
    - Build time dependency
  - New vehicle APIs
    - Separation of API for behavior and signal data model
    - Using the Code Generator by VSM(*.vsm) definition
    - Runtime dependency
    - Easier to create connectivity with servers based on defined data models

```cpp
// Example code for cabin::isDoorOpened(HDoorPosition::FRONT_LEFT, Value);

cabin::isDoorOpened(HDoorPosition::FRONT_LEFT, Value);
```

```cpp

getSignal("Vehicle.Cabin.Door.Row1.Driver.Open", Value);
```
ccOS VSM

- Vehicle Signal Model (VSM)
  - Main mission to standardize vehicle monitoring and control as an interface
  - Same starting points from COVESA Vehicle Signal Specification’s domain taxonomies
  - VSM provides a standard interface for vehicle integration of ccOS App
  - Over 2800+ signals that for vehicle integration have already been defined

< Reference: Taxonomies::Vehicle Signal Specification [covesa.github.io] >
ccOS VSM

• VSM Key Feature
  – VSM’s Rule Set and Data Definition method follows VSS
    *e.g., YAML syntax, using root node name with “Vehicle.*”*
  – Rule Set
    ▪ Node Type
      : branch, sensor, actuator, attribute, getproperty, setproperty
    ▪ instance, aggregate concept is not considered
    ▪ Node Name Rules
      : Defined the node name in terms of classification of control and sensor’s target (★)
      : Use the same node name with sensor/actuator, getproperty/setproperty
  – Binding IVN signal relationship to VSM Node
    ▪ Regular Relationship Support (1:1 Case)
    ▪ Multiple Relationship Support (N:1 Case, 1:M Case)

<table>
<thead>
<tr>
<th>Perspective</th>
<th>Brake</th>
<th>TurnSignal</th>
</tr>
</thead>
</table>
VSM Details

- IVN Signal - VSM Node Binding Rules
  - Supporting Regular Relationship (1:1 Case)
  - Supporting Multiple Relationship (N:1 Case, 1:M Case)
    - Multi IVN signals combines to One VSM node (N:1 Case)
    - One IVN signal updates to Multiple VSM node (1:M Case)

<table>
<thead>
<tr>
<th>Options</th>
<th>Defined Rule</th>
<th>Case Study</th>
</tr>
</thead>
</table>
| 1:1 Case | IVN and VSM has 1:1 correspondence | For checking the heating status of the handle
  - IVN: StrWhlHtrSwSta (Rx)
| N:1 Case | Can be expressed as one node by combined expression (exclusive condition or If-else) | Sender ECU could be different, but it combine to single VSM
  - IVN: StrWhlHtrSwSta (Rx), StrWhlHtrSwSta_v2 (Rx)
| 1:M Case | One IVN signal is represented in various VSM node, it may be defined as Alias Node. | IVN: TrnSigLmpLtBlnkngSta, TrnSigLmpRtBlnkngSta (Rx)
VSM Case Study #1

• Case Study : Headlamp system warning and lamp open-circuit warning
  – IVN signals requirement and use-case based scene analysis
    ▪ Each headlamp has an open circuit warning signal or system level warning signal
    ▪ Front headlamp has an up-light, down-light and turn-light on the left and right sides
    ▪ Three different IVN signal types (Type A/B/C)
    ▪ To design a leaf node with Vehicle.Body.Lights.Front.Left. as a parent
    ▪ Need to inform the customer of the detailed trouble shooting when error occurred

< Type A >

< Type B >

< Type C >
VSM Case Study #2

- Case Study: Turn light signal
  - IVN signals and scene analysis
    - There are 4 turn-light signal lamps on vehicle
    - The left and right blink independently
    - Turn-light signal can be operated from side to side
      (TrnSigLmpLtBlnkngSta / TrnSigLmpRtBlnkngSta)
    - Signal value is always on while turn-light signal is flashing
    - Hazard lamp blinks 4 turn-light signal together, with individual signal, not sharing turn-light signal
  - Review neighbor VSM node
Introduction to CCS

• Connected Car Service
  – Our connected car service was launched in 2003 with MOZEN service
  – Support for safety security features, route searching, concierge services, media streaming, etc.
  – Next Generation CCS includes near-real-time vehicle status transmission
CCS VSM

• Main mission to standardize vehicle monitoring and control as an interface

• Extracting data model data commonly used from the perspective of implementing connected car services

• Abstract if the state model is complex, includes semantic abstraction of transmitting value

• CCS VSM provides a standard data model for vehicle integration with connected car

• Over 300+ signals that for vehicle integration have already been defined for Smartphone Application
CCS VSM

- CCS VSM Key Feature
  - CCS VSM Rule Set and Data Definition is the subset of Vehicle VSM
  - Rule Set
    - Node Type
      - branch, sensor, *Rule based Reporting Policy (In Progress...)*
  - Binding IVN signal relationship to CCS VSM Node
    - Refining the valid value and apply it to CCS VSM node
  - Legend
    - : Vehicle VSM Domain Only
    - : Shared VSM Node
    - : CCS VSM Domain Only
VSM Case Study #3

- Case Study: Headlamp State to VSS?
  - Conversion from Vehicle VSM node to CCS VSM
    < Type A >

    < Type B >

    < Type C >

  - VSS Neighboring node

< CCS VSM >
HMC’s Vehicle Data Architecture

Vehicle Domain
- ccOS Application & CCS Service
  - CCS App
- ccOS Framework
  - Vehicle VSM
- ccOS HAL
  - libvehicle-hal.so

OEM’s Cloud Service Domain
- ccOS Application & CCS Service
  - VSS (OEM)
- CCS Backend
  - Broker
  - Vehicle Status
- CCS VSM

3rd party domain
- 3rd Party Playground
  - VSS
- VSS Data
  - VSS (3rd Party)

Vehicle Data Architecture
Thank you :-)