CCS Proof-Of-Concept - Work Breakdown Structure

Implementation roadmap

- milestone 1 - Spring GENIVI Virtual Tech Summit (12-14 May)
- milestone 2 - internal milestone (early Q3 - 17 July)
- milestone 3 - Fall All Member Meeting (Virtual) 26-30 October
- milestone 4 - January 2021 – (originally planned for CES, now general milestone)

Communication framework architecture

This page lists the different components and tasks to be completed for the architecture discussed on the Vehicle data exchange protocols page. The table references to the following diagram, with components that are considered in scope for the PoC highlighted with green color.

Components

Vehicle
Definitions:
StateStorage and VSSFeeder is expected to be combined into one component in the implementation.

SignalStoreIF: Done by reading/writing sqlite database transactions, and using the notification feature to notify components that update occurred.
StateStorage: Essentially implemented by sqlite instance
SignalFeedIF: This is an internal implementation detail now since StateStorage/VSSFeeder is essentially one program
Interface details

(OEM Cloud) GraphQLDatabaseIF: (Interface between VSS2-Database Resource Mgmt (GraphQL server))

The interface is expected to be accessing the database itself. In order to give the GraphQL server full freedom to query data intelligently, it should connect directly to the SQL database.

Description of work

Apache License is more preferred by participants.

<table>
<thead>
<tr>
<th>#</th>
<th>Component</th>
<th>Plan / Activities</th>
<th>Status of chosen software components (Implementation details of PoC)</th>
<th>Alt. SW components / Implementation for Production Systems (WIP, future)</th>
<th>Owner</th>
</tr>
</thead>
</table>
| 1  | In-vehicle State storage (laptop simulator) | • Custom code + in-memory database (+ persistence) + connection to feeder  
• Either locally cached during uptime of system or  
• Stored and indexed in a database to allow access to historical data  
• Selection of database  
• Implementation of database schema  
|    |                               |                                                                                  | • Custom control code (Python or C++)  
• sqlite binding: OK for PoC  
|    |                               |                                                                                  | Implemented by a shared sqlite database file. The "API" is simply interacting with the database using SQL statements and/or sqlite bindings.  
Note: statestorage executable in a one-shot thing that sets up the database. The gen2-server accesses the actual database directly using sqlite binding.  
|    |                               |                                                                                  | • Custom code  
|    |                               |                                                                                  | CCS-108 - Develop production-grade version of in-vehicle storage | CCS-109 - Reach out Kuksa project for in-vehicle state storage components | Keith                          |
|    |                               |                                                                                  |                                                                                                                                   | TO DO | DONE |
In-vehicle VSS2 translation (a.k.a. VSS feeder) (laptop simulator)

- Implementation of SOME/IP client?
- or simple simulator
- or Vehicle Driving Simulator (LG? OpenDS? or GENIVI GitHub version?)
  - set up simulator, make it drive around track, get list of available signals, write code to convert signals to VSS.
  - Signals are fed over DDS to AutoWare or Apollo autonomous driving stacks.
  - Sim needs a high-end graphics machine
- Look if VSS or VSD should play a part
  - New: live_simulator which is feeding a real-time playback of existing timestamped data.

Live-simulator implemented by Ulf feeds data taken from an existing "ovds" database and feeds it into statestorage. Example ovds database to feed live simulator now exists.

Additional options:
- Custom code, feeding simple simulated data
- Then (future) run full vehicle driving simulator
  - continue driving-simulator track and "playback" simulator in parallel.

Develop In-vehicle VSS2 translation (a.k.a. VSS feeder)

Develop proof-of-concept version of in-vehicle VSS2 translation

Develop in-vehicle data packaging

Develop proof-of-concept version of in-vehicle data packaging

Develop in-vehicle data server

Develop proof-of-concept version of in-vehicle data server

Develop production-grade version of in-vehicle data server

Develop production-grade version of in-vehicle data server

In-vehicle Data Package

- Collecting VSS2 data into snapshots and bundles according to Value measurement formats
- Presenting data packages to the Data server
- Possibly closely related to the State storage schema

CCS-113 - Develop in-vehicle data packaging

CCS-114 - Develop proof-of-concept version of in-vehicle data packaging

In-vehicle Data server (laptop simulator)

- Data server implementation for W3C Gen2
  - Reference implementation exists in GitHub MEAE-GOT
  - Ulf can work together with someone how to connect to an existing API of "state storage"
  - also talk to Kuksa project - VISS+REST server

Gen2 implementation now uses ovds.db file.

Use server directory from W3C_VehicleSignal InterfaceImpl

CCS-117 - Develop proof-of-concept version of in-vehicle data server

Contents updated Dec 11, 2022.

In-vehicle Data server (laptop simulator)

- Data server implementation for W3C Gen2
  - Reference implementation exists in GitHub MEAE-GOT
  - Ulf can work together with someone how to connect to an existing API of "state storage"
  - also talk to Kuksa project - VISS+REST server

Gen2 implementation now uses ovds.db file.

On-demand requests are fetched from database. Timed subscriptions are also supported, i.e. send updates at regular intervals.

There is not yet implementation of a trigger to the gen 2 server from the database when a new value is written (SQLite supports trigger in theory).

Vehicle client current does get-requests on a timer (polling) but could in the future also exercise the subscription feature in the protocol.

OEM Cloud Vehicle client

- W3C Gen2 protocol (VISS WebSocket Pub/Sub)
  - Options:
    - Sanjeev involved in writing the client using Go-lang.
    - Curl script
    - Custom code + HTTP library (e.g. written in python) custom code + libcurl binding
    - Client directory from MEAE-GOT / W3C
    - No clear answer. Depends on use-case. What is the rate of data for example?
    - Sanjeev: it would make sense to write also client in Go.
  - This program shall also store the data into the data lake program (or directly into the database used as data lake)

Written in Go, some similar code as in W3C Gen 2 reference server

Demo includes this function already... In ccs-client repo.
Client reads data from data server in vehicle via the VSSv2 protocol, and writes it into the OVDS database.

Vehicle client current does get-requests on a timer (polling) but could in the future also exercise the subscription feature in the protocol.
<table>
<thead>
<tr>
<th>6</th>
<th>OEM Cloud</th>
<th>VSS2 database</th>
</tr>
</thead>
</table>
| **CCS-121** - Develop OEM cloud VSS2 database | • Selection of libraries and database  
  • Define database schema. Start with Ulf's proposal for DB schema |
| Postgres was discussed. Use SQLite first. |
| Currently the translation path to UUID is currently in a separate database, compared to the signal database. A future possibility is two tables in a single database, making it possible to use SQL JOIN statements. |

<table>
<thead>
<tr>
<th>7</th>
<th>OEM Cloud</th>
<th>Identity management</th>
</tr>
</thead>
</table>
| **CCS-126** - Develop OEM cloud identity management | • Implementation of end-user login and authentication using OpenID  
  • Lots of candidates. Responsible implementer should propose a good alternative. |
| TODO | Later, programming language preference could be influenced by the programmer |

<table>
<thead>
<tr>
<th>8</th>
<th>OEM Cloud</th>
<th>Access management</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CCS-127</strong> - Develop OEM cloud access management</td>
<td>• Implementation of authorization between end-user and 3rd party application or Neutral Server using OAuth2</td>
<td></td>
</tr>
<tr>
<td>TODO</td>
<td>Later, programming language preference could be influenced by the programmer</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9</th>
<th>OEM Cloud</th>
<th>Resource Management (= Data server API)</th>
</tr>
</thead>
</table>
| **CCS-128** - Develop OEM cloud resource management (= Data server API) | • Implementation of basic API management  
  • Resource Mgmt is the ExVe naming.  
  • Implementation of a GraphQL API using the VSS2 schema  
  • Interface to the client, is of course defined by GraphQL+=schema.  
  • Interface to database (GraphQLDatabaseIF) is described above |
| GraphQL Apache Apollo? | Located in vss-graphql repo |
| • Has one schema.  
  • Does not have tool to regenerate schema anymore. Manually done for now by Kevin/High Mobility.  
  • Has Apollo server in docker  
  • Needs to implement the connection from GraphQL (Apollo) to database. It could use the REST protocol of the OVDS server, or directly SQLite database.  
  • **Important to fix!!!** |

In production more likely to be an object store database instead of a RDBMS.
Sanjeev looking at Apache ecosystem and Hortonworks /Cloudera platform capabilities.

In PROGRESS

IN PROGRESS

DONE

DONE

DONE

DONE

DONE

DONE

DONE

DONE

DONE

DONE

DONE

DONE

DONE

DONE

DONE
<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
<th>Text</th>
</tr>
</thead>
</table>
| 10   | Neutral Server Data Marketplace | - Separate instance that consumes the OEM Cloud OAuth2 and GraphQL APIs
  - We could implement a very simplistic neutral server using the published API of High Mobility, but an open implementation.
  - Security, consent and other complicates things. Leave those details out.  
CCS-131 - Develop neutral server marketplace  
TO DO  
Kevin |
| 11   | 3rd Party Application | - Example applications exist on High Mobility’s GitHub
  - One instance that consumes the Neutral Server/Data Marketplace APIs
  - Use an example application from HM
  - For example an app that just shows the data (written in Node.js)
  - One instance that consumes the OEM Cloud OAuth2 and GraphQL APIs
  - Modify the application to use the OEM API directly  
CCS-133 - Develop 3rd party application  
IN PROGRESS  
Kevin |

In `vss-graphql-client-swift` repository

- This could show a way for 3rd party applications to access the OEM /GraphQL interface directly but possibly a more limited REST-API is more realistic for 3rd party access.  

**TODO:** Expose a neutral-server API to third party applications.  
Nothing in particular to develop. It would just be a proxy (for the GraphQL and REST) – same technologies as our current OEM connection would be used by the Neutral Server.  

- CCS-132 - Develop proof-of-concept version of neutral server marketplace  
TO DO  
Kevin |
| 11   | 3rd Party Application | - New sample app
Connecting directly to OEM cloud for now, via GraphQL.  
For connecting to Neutral server instead:

- Example/standard API work needed.  
  The example apps are using High Mobility’s API. See above  

- Need discussion/definition of what the open API should be.  
  The API would be quite transparent if VSS is exposed directly, but the Neutral Server API might expose different functions. Also, aggregation...  

But enriched functions / anonymized aggregated data might be provided by Neutral Server.  
See above  

- CCS-134 - Develop proof-of-concept version of 3rd party application  
DONE  
Kevin |