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, Leipzig, Germany (last week of October)
- milestone 4 - CES 2021 (January 2021)

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
ValueStoreIF:  
ValueQueryIF:

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>Work</th>
<th>Software Components / Implementation details (PoC)</th>
<th>Software 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  
[ CCS-106 ] - Develop in-vehicle state storage | • Custom control code (Python or C++)  
• sqlite binding, OK for PoC  
[ CCS-107 ] - Develop proof-of-concept version of in-vehicle storage | • Custom code  
[ CCS-109 ] - Develop in-vehicle state storage components | Keith |
| 2  | In-vehicle VSS2 translation (a.k.a. VSS feeder) (laptop simulator) | • Implementation of SOME/IP client?  
• or simple simulator  
• or Vehicle Driving Simulator (LG7? 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 VSI or VSD should play a part  
[ CCS-110 ] - Develop in-vehicle VSS2 translation (a.k.a. VSS feeder) | • Custom code, feeding simple simulated data then (future) run full vehicle driving simulator  
[ CCS-111 ] - Develop proof-of-concept version of in-vehicle VSS2 translation | Keith |

CCS-106 - Reach out kuksa project for in-vehicle state storage components

CCS-109 - Reach out Kuksa project for in-vehicle VSS2 translation / VSS feeder components

TO DO
| 3 | In-vehicle Data Package | - Collecting VSS2 data into snapshots and bundles according to Value measurement formats<br>- Presenting data packages to the Data server<br>- Possibly closely related to the State storage schema<br>� CCS-113 - Develop in-vehicle data packaging TO DO | Protocol defined within the CCS project<br>Start with results of Apache NiFi track to fulfill this need. | Future development. No support currently to transfer a Data Package in W3C Gen2<br>� CCS-115 - Develop production-grade version of in-vehicle data packaging<br>Gunnar |
| 4 | In-vehicle Data server (laptop simulator) | - Data server implementation for W3C Gen2<br>� Reference implementation exists in GitHub MEAE-GOT<br>� Ulf can work together with someone how to connect to an existing API of "state storage"<br>� (also talk to Kuksa project - VISS+REST server)<br>� CCS-116 - Develop in-vehicle data server TO DO | Use server directory from W3CVehicleSignalInterfaceImpl<br>� CCS-117 - Develop proof-of-concept version of in-vehicle data server DONE | Ulf<br>� CCS-114 - Reach out Kuksa project for in-vehicle data server components<br>DONE |
| 5 | OEM Cloud Vehicle client | - W3C Gen2 protocol (VISS Websocket Pub/Sub)<br>� Options:<br>� Investigate Sanjeev’s work for reusable client<br>� Curl script<br>� Custom code + HTTP library (e.g. written in python) custom code + libcurl binding<br>� Client directory from MEAE-GOT/W3C ?<br>� No clear answer. Depends on use-case. What is the rate of data for example?<br>� Sanjeev: It would make sense to write also client in Go.<br>� This program shall also store the data into the data lake program (or directly into the database used as data lake)<br>� CCS-118 - Develop OEM cloud vehicle client TO DO | Written in Go, some similar code as in W3C Gen2 reference server<br>� CCS-119 - Develop proof-of-concept version of OEM cloud vehicle client DONE<br>� CCS-120 - Develop production-grade version of OEM cloud vehicle client<br>Ulf |
| 6 | OEM Cloud VSS2 database | - Selection of libraries and database<br>� Define database schema<br>� Start with Ulf’s proposal for DB schema<br>� CCS-121 - Develop OEM cloud VSS2 database IN PROGRESS<br>Gunnar: I have some experience setting up Postgres, so this is a reasonable option. Ulf: A Go library to support Postgres exists, so this is only a small rewrite.<br>� CCS-122 - Develop proof-of-concept version of OEM cloud VSS2 database<br>� Implemented in in ccs-client repo.<br>� Sanjeev looking at Apache ecosystem and Hortonworks /Cloudera platform capabilities. | Custom control code (Python or C++)<br>� Sqlite binding, OK for PoC, or other database such as postgresql.<br>� Need to define database schema first pro posal is here<br>� In production more likely to be an object store database instead of a RDDB.<br>� CCS-123 - Reach out Geotab to learn the Geotab platform capabilities<br>DONE<br>� CCS-124 - Investigate Apache NiFi/Spark and other technologies for data architecture<br>DONE<br>Ulf |
| 7 | OEM Cloud Identity management | - Implementation of end-user login and authentication using OpenID<br>� Lots of candidates. Responsible implementer should propose a good alternative.<br>� CCS-126 - Develop OEM cloud identity management TO DO | Later. programming language preference could be influenced by the programmer<br>� CCS-127 - Develop proof-of-concept version of OEM cloud VSS2 database<br>� Implemented in in ccs-client repo.<br>� Sanjeev looking at Apache ecosystem and Hortonworks /Cloudera platform capabilities. | -- |
| 8 | OEM Cloud Access management | - Implementation of authorization between end-user and 3rd party application or Neutral Server using OAuth2<br>� CCS-127 - Develop OEM cloud access management IN PROGRESS<br>� CCS-128 - Reach out Geotab to learn the Geotab platform capabilities<br>DONE<br>� CCS-129 - Investigate Apache NiFi/Spark and other technologies for data architecture<br>DONE<br>Ulf | Later. programming language preference could be influenced by the programmer<br>� CCS-127 - Develop proof-of-concept version of OEM cloud VSS2 database<br>� Implemented in in ccs-client repo.<br>� Sanjeev looking at Apache ecosystem and Hortonworks /Cloudera platform capabilities. | -- |
| 9 | OEM Cloud Resource Management (= Data server API) | Implementation of basic API management
• Resource Mgmt is the EvVe 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. |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GraphQL Apache Apollo?</td>
<td>CCS-129 - Develop proof-of-concept version of OEM cloud data API  IN PROGRESS</td>
</tr>
<tr>
<td></td>
<td>Located in vss-graphql repo</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Has one schema.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Does not have tool to regenerate schema.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Has Apollo server in docker</td>
<td>(Alexander Domin)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CCS-130 - Reach out to BMW for the GraphQL example  DONE</td>
</tr>
<tr>
<td></td>
<td>CCS-128 - Develop OEM cloud resource management (= Data server API)  TO DO</td>
<td></td>
</tr>
</tbody>
</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. |
|  | Repo link? | CCS-132 - Develop proof-of-concept version of neutral server marketplace  TO DO |
|  | In vss-graphql-client-swift repository | Kevin |
|  | • Has programming framework/example (in SWIFT) to access data via graphql. | |
|  | • 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 part app access | |
|  | • TODO: Expose a neutral-server API to third party applications. | |
|  | CCS-131 - Develop neutral server marketplace  TO DO | |
|  | CCS-130 - Reach out to BMW for the GraphQL example  DONE | |
| 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 |
|  | Example/standard API work needed
The example apps are using High Mobility’s API. | CCS-134 - Develop proof-of-concept version of 3rd party application  DONE |
|  | 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. The apps could however be used with modification. | Kevin |
|  | CCS-133 - Develop 3rd party application  IN PROGRESS | |