CVII Tech Stack

Child pages:

- CVII Technology Stack meeting notes
- CVII Technology Stack -- pre-kickoff ideas page
- CVII Tech Stack: "Deep-dive" topics
- CVII Tech Stack technologies - Protobuf
- CVII Tech Stack Terminology
- CVII-TS: Binary encoding for VSS-based data transfer
- CVII-TS: MQTT
- CVII-TS: VISS v2 server implementation in C++
- IoT-event-analytics and Vehicle-Edge projects
- Workshop and conf planning

The Technology Stack development is one of three main tracks of the Common Vehicle Interface Initiative

blocked URL

Goal of this activity:

Find/Develop/Define the required technology solutions (software, primarily) that are involved in transfer/storage/processing of **the industry-common models for data & services** (as agreed upon by the other CVII tracks).
This includes anything within the "full scope" system (i.e. in-vehicle, edge & cloud), if it is software related to the common data/services model technologies.

Cross-platform support seems to be a general request but Linux is likely the dominant platform for *developing / testing* the code.

The term Technology Stack is used to describe all software that is related to the transfer and use of data and services that adhere to the common model (s).

Examples:

blocked URL

How is the Technology Stack defined and developed?

- Through a combination of formal specification and open-source implementations, as chosen by the participants.

- Of course, by evaluating and selecting existing technologies where they exist, and making the necessary adjustments/bindings/plugins to make the common data/services models fit into existing frameworks.

- ... and developing the documents and software code that is missing.

Technology Stack content analysis, and development state

The following lists show an overview of planned, desired, and existing technologies - including format-converters, code-generators, and bindings to existing technologies.

It should be continuously updated to reflect completion of technology definition (specification) or implementation

For now all known parts are mentioned. Overlapping initiatives ought to be possible to combine, to work towards a single consistent full-stack design.

The purpose of the technology stack is to lead *us* (the automotive industry) towards a reasonable selection of solid core technologies without trying to support everything under the sun. In other words, there might not always be one selected choice in the software solutions, protocols, etc, but there should still be only one choice for the underlying data/services model, according to CVII goals.

Converters and Code Generators

A listing of conversions like this might look like a **anything-to-anything** approach, but it should be clear that the *goal* is not to create that, as explained in the previous section. Instead, it is to make sure there are solutions to hook into technologies that are **strongly desired**, or **unavoidable legacy**.

Terminology

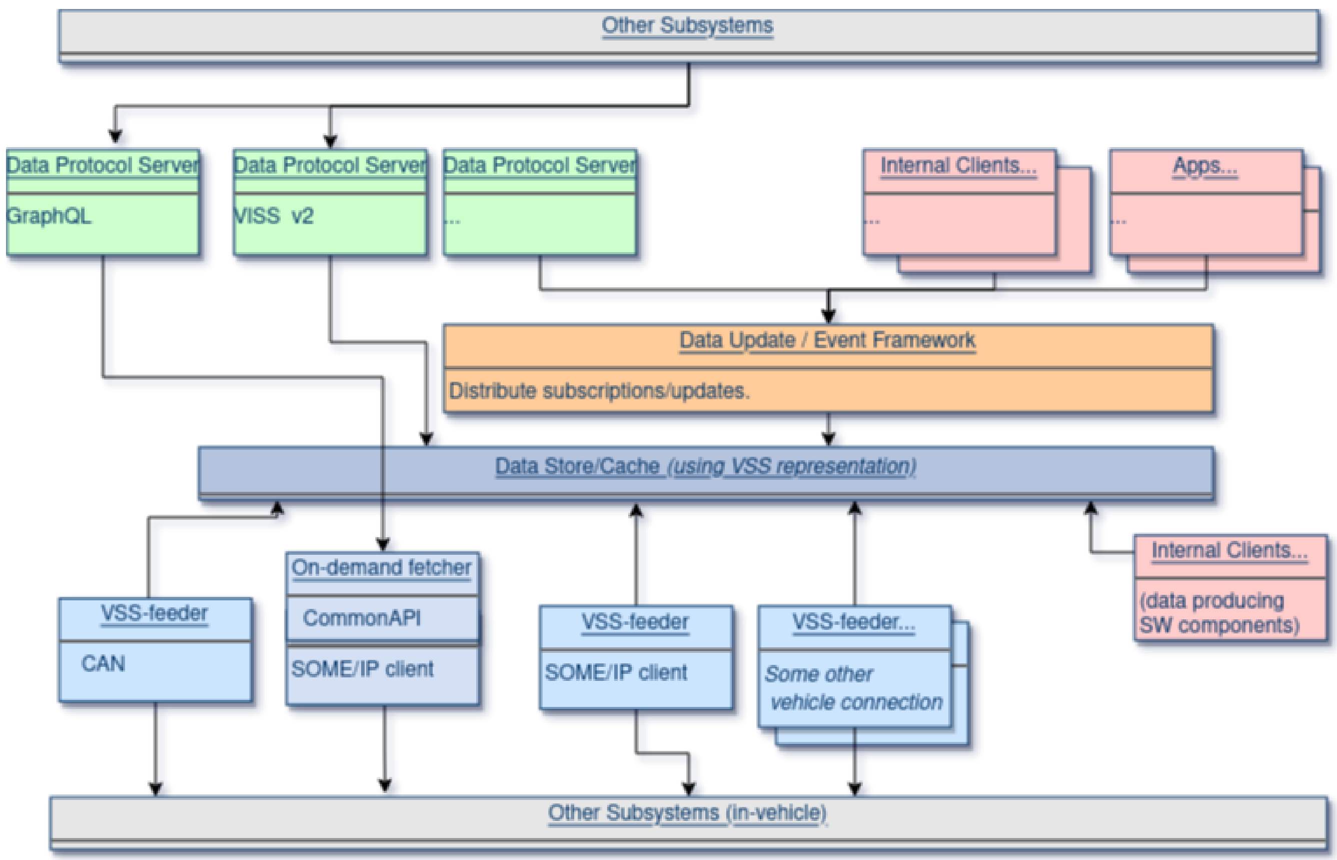
It can be useful to refer to a shared understanding CVII Tech Stack Terminology. Make sure you provide **your** input to these terms.

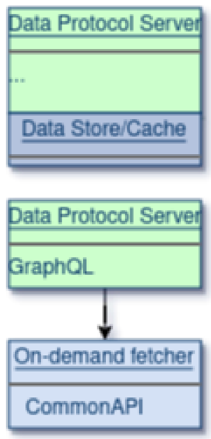
Technology Stack components

A) Architecture(s) and terminology

1. To get consensus around terminology and parts to develop we need to revise this in diagram and text form.
2. This page outlines the goal, and challenges to reach pluggable software components, to be used in a shared reference architecture but also possible to use in different variants that companies may come up with.
3. Agree on and then use the CVII Tech Stack Terminology to understand these designs. Of course, these terms are open to additional input.

A1) Partial system diagram (primarily representing an in-vehicle, possibly a single ECU, but terms are generic and components could be reused in a different setup)





- Some Data servers are closely attached (or include) a data store

- Some Data servers implement connection directly to their data source (skipping a specific data-store component)
- Some fetch data on-demand, when the request comes in and/or through subscription.

Source file for this diagram is here: It can be imported into https://app.diagrams.net/ and edited again.

A2) Vehicle-to-cloud full architecture

The best representation at this time is the Covesa CCS Project Reference Architecture. **NOTE**: Be aware that the technologies mentioned therein are not set in stone. Rather, this page represents a broader view of the variation of software components/protocols/etc. that may apply.

B) Development Plan for needed technology (definitions, implementations, tools)

This is intended to *become* a relatively complete list of needed deliverables.
Add to it, if some technology you need is missing.

Conversion from VSSdefinitions (data catalogs) to alternative file format:

Note: Conversions to other formats are done to *extend VSS with additional features* , or to hook into useful existing technology that happens to consume another input format.

- **VSSo** ✓ vspec2ttl in vss-tools. See VSSo specification here.
- **JSON** ✓ vss2json in vss-tools,
- **GraphQL** ✓ vspec2graphql in vss-tools and BMW implementation vss2graphql_schema
- **ARA:COM,** AUTOSAR-XML format. (⭐ Sort of available indirectly, via Franca IDL conversion and then FARACON)
- **Protobuf** - (✓) vspec2protobuf in vss-tools (⚠️ Used as a definition of the VSS catalog. Not sure exactly how that is to be used - instead we should consider Protobuf under the serialization topic below)
- **Franca IDL** - see below for links. Similar comment as for the vspec2protobuf defining the "VSS" or the prerequisite for transferring values of the VSS?
- **DDS** – Evaluating interest

Serialization / value-message formats

Note 1: This is different from the VSS catalog in that it does not define the signals, it defines how to transfer (or store) measured values of those signals.

Note 2: Several of these are generic technologies that could be use any number of ways to transfer data (e.g. JSON). The purpose of this work is to define a single canonical definition for each such format

- **JSON** ⭐ N.B. lot can be defined by reusing the formats of the VISS specification, but putting it in independent terms in a separate specification would be useful.
- **Protobuf**
- **AVRO** ⭐ Under development in vss-tools
- **Franca IDL** ✓ Conversions exist in vss-tools alt1, alt 2, but likely still needs an update. Plenty to improve/redefine also as part of how VSS shall integrate with VSC
- **gRPC** (Being investigated/added in KUKSA.VAL. Is it for data/notifications or other APIs? How does it differ from protobuf definition. Status Sebastian Schildt ?)

Data transfer protocols

Note: Some of these will require using a serialization format to define the payload. Others may bring/define their own formats.

- **VISS** ✓ – Specification
  WAII implementation (Go lang),
  also (partially) implemented in KUKSA.VAL, and other?...
- **SOME/IP** (✓) – (⚠️ via Franca IDL and CommonAPI). Likely additional ways to implement a more direct VSS SOME/IP connection.
  - (Also **WAMP** exists, also via Franca+CommonAPI)
- **MQTT** – ⭐ There exists *VISS over MQTT* defined in VISS specification, but not a plain usage where you use simple MQTT-subscription (with topics = VSS signals).
  see separate analysis page

Runtimes and Frameworks

This is a list of software projects that do somewhat more than just a "data protocol" but are still involved in defining data exchange.
Note: Some of these define their own protocols, or the protocol is hidden (abstract) behind the framework APIs.

- **Android Automotive** (Vehicle-Properties in VHAL) ⭐ Under development: Code generator for VSSAndroid Automotive Vehicle Properties (See this vss-tools fork and/or code_generation branch on vss-tools).   Once merged, see main branch in vss-tools.
- **Apache Kafka, and NiFi** – This appears to requires some simple plugins/definitions of how to transfer VSS data to leverage the whole framework
- **DAPR** – "Distributed Application Runtime" – Data and RPC framework

Databases / Storage

Note: There are many choices here. Some might simply implement a simple in-memory table. Others define themselves as "databases" or "key-value stores".

In-vehicle

- Apache IoTDB – ⭐ under development, StephenL is working on it.
- Redis – popular choice for in-memory key/value store (and more features)

OEM/Neutral Cloud

A number of choices are possible here and there are no plans yet to prioritize a particular one at this point.
Rather whichever existing implementations become known / implemented first, ought to be listed here.

(only examples)

- InfluxDB
- Postgres Time-Series
- MongoDB
- cloud-providers data store services (AWS, Azure, Google)
- and likely many other options...

Data processing

- Apache Spark

Other software frameworks
These typically have a scope beyond data/RPC only but support the common data/services model in their implementation

- AOS – "Kubernetes for the vehicle"

Components for services / remote-procedure-call (VSC)

Conversion from VSC *catalogs* to alternative file format

Note: Conversions to other formats is done to extend it with additional features or hooking into existing technology.

- ARA:COM (AUTOSAR-XML format). Defining the data items is a step towards then using SOME/IP or other protocol AUTOSAR side, leveraging AUTOSAR-defined code generators.
- "gRPC IDL" (i.e. protobuf extension)
- Franca IDL - ⭐ VSC project has Franca import/export as a core focus. Occasionally VSC is considered almost a new version/extension of Franca. Better definitions will follow, but the direction to stay close to it is clear.

RPC protocols

Note 1: In contrast to VSS, for VSC/services we did not separate the serialization definition from the protocol. It is simply understood that each RPC protocol will define the wire-format of its method-invocation request and responses as part of its specification.

1) W3C-chosen official protocol for automotive services "RPC"

Defining such a protocol is being discussed, to mirror the definition of VISS for data. A definition project has not yet been launched since it relies on the VSC-language and semantics being solidified first. (It may of course decide to reuse existing definitions, including some of the ones listed below)

2) Options

- **gRPC**
- **GraphQL** - through its mutation feature GraphQL can theoretically support RPCs. Is there interest in doing that?
- **SOME/IP (**Possible if converted to Franca IDL, via the CommonAPI framework **(capicxx-someip)).** Just like for VSS there are likely additional ways to implement a more direct VSC SOME/IP connection
- **DDS** ? – Evaluating interest
- **DAPR** – "Distributed Application Runtime" – Data and RPC framework

Runtimes and Frameworks

- **Android Automotive** - likely less applicability than for VSS, since Android has its own preferred HAL. Describing Android-specific interfaces with the generic common IDL might have less value therefore, but it is of course a theoretical possibility if this is a benefit to someone.
- **...**

Noteworthy software frameworks (typically with scope beyond data only), but to include VSC

- AOS – "Kubernetes for the vehicle" VSC connection not defined yet

Other things mentioned for completeness

- Is JOYNR a candidate worth including and studying here? (It is Franca IDL based, and also supports overlapping technologies like MQTT)

Shortlist, prioritized projects
(2021, not clear if it is still representative)

1. "Second" VISS v2 implementation (in addition to WAII implementation)

Background

- C++ seems to be requested.   There is also some mild interest in Rust in the industry, but so far it seems not strong enough to supplant C/C++ as the main choice.
- KUKSA has both done VISS implementations in C++, based on VISS v1 - needs to be updated to v2
- AOS includes a VIS(S) implementation in Go, based on VISS v1

Details:

- VISS v2 server Planning / Work Breakdown page

2. Demonstration of VSC-based development

Background:

- Bosch eager to get started. Discussions about VSC (latest intro material here) may need to be progressed first (or in parallel).
- Getting started on a demonstration case will naturally drive the development of "key components" that are missing.

3. Define "efficient binary encoding" of VSS payloads, as a reusable implementation

A) For individual message updates through protocols (VISS? MQTT, etc.)
B) For in-memory storage of multiple data points and possibly subsequent batch (image) transfer.

- Volunteers needed – see Planning / Work Breakdown page

Project Backlog – development project candidates

- **VSS to Android Properties** implementation (Code Generator).   It should likely introduce jinja2 template based generation, similar to the vsc-tools .   Work breakdown page.
- **VSS over MQTT** (Starting point exists in IoTea impl. Develop full implementation, include access-control/security, and as a reusable component separated out from IoTea framework) Work breakdown page
- **VSS to/from Autosar ARA:COM XML** format
- **VSC to/from Autosar ARA:COM XML** format
- Proper **VSS to Franca IDL** converter. Status and remaining steps are in vss-tools/PR#6.

for further, potential examples, refer to the table based overview below.

Historical / preparation information.

Initial Brainstorm, implementation ideas

Which technologies come immediately to mind?

- (MQTT moved to separate page)
- Investigation point: VISSv2 with MQTT as "transport" - is there a possible compatibility?
    - This has now been showed as an idea in W3C.   link?

- General (MQTT and other): Binary data representation to optimize compared to VISS-JSON transport.
    - ...use also as variant encoding in VISS?
    - ...use as independent protocol (not part of VISS spec)
- AUTOSAR Adaptive compatibility
    - SOME/IP is important so we cannot ignore it.
    - DDS also.
    - (However, ARA:COM abstracts the technolgies. For AUTOSAR systems, generating ARA:COM XML format is enough, the rest follows. For non-AUTOSAR systems that are going to communicate with AUTOSAR systems, however, something that implements the concrete chosen protocol is needed.
        - Previous work: Franca IDL Common API for SOME/IP and vSomeIP, and Franca IDL AUTOSAR XML converter)
    - Opinion: SOME/IP is used in a very static way – an alternative might be desirable.
    - (AUTOSAR) RESTful services (not HTTP, rather stateless principle, get, set etc.) Only discussions currently, no drive?
        - How is this concrete protocol defined?
        - Bosch might be willing to work on this.
- Protobuf conversion Main page
  (because it is used and preferred by SENSORiS, but is also generally popular, and there was a recent vspec2proto converter proposal in vss-tools)

VSC to code generation

- Code generation - general stubs/proxies (C++ and other)

A lot of communication related technologies were investigated in the Generic Protocol Evaluation project during 2019.
A set of reference links are here : List of relevant technologies

AUTOSAR

Current tool chain

(RED is not existing or not yet clearly defined). (The rest exists already)

blocked URL

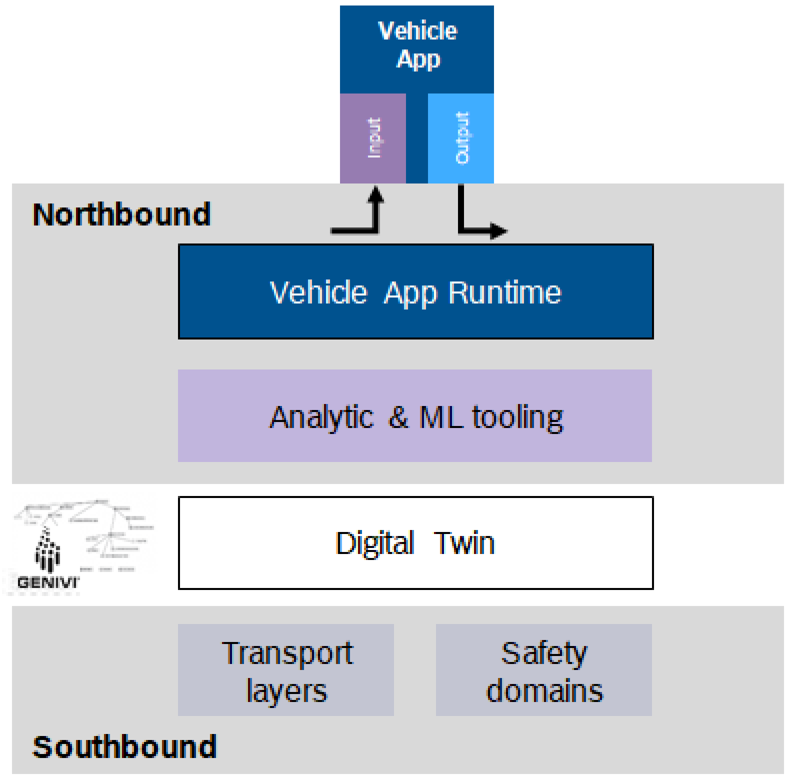
Notes

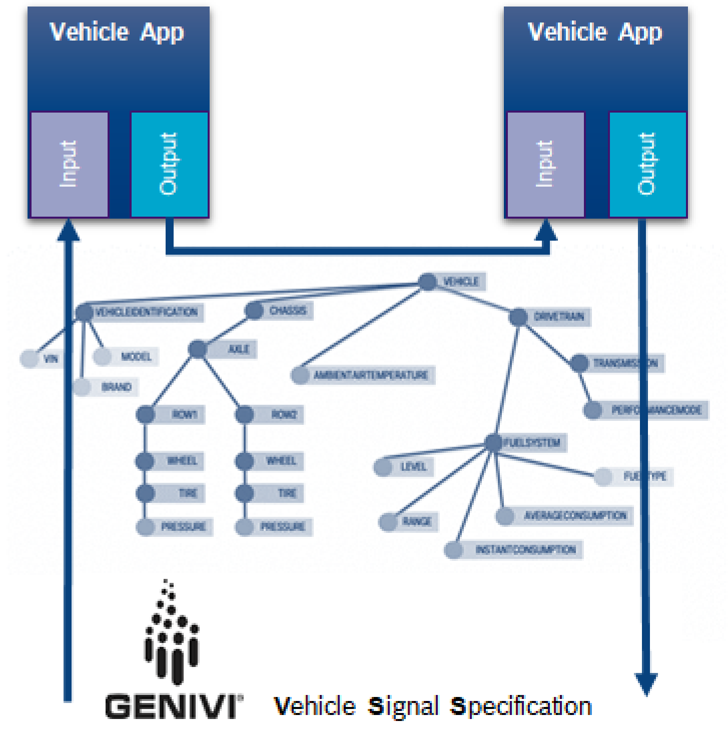
Going via Franca is complicated...   VSS/VSC to AUTOSAR directly makes sense

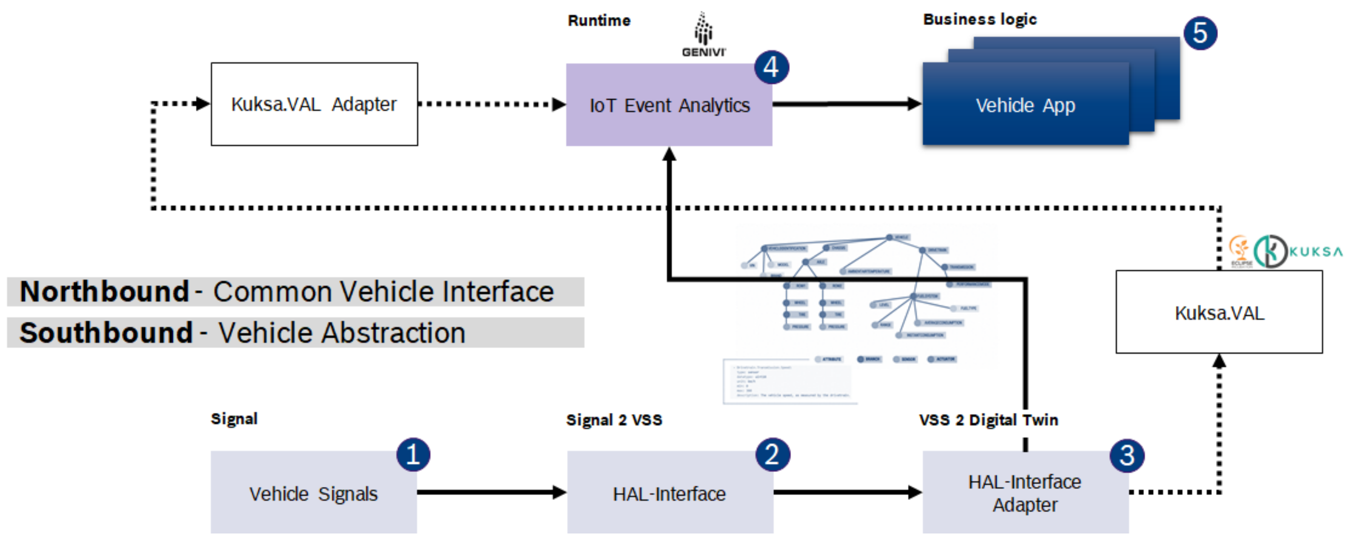
Direct approach for AUTOSAR

blocked URLblocked URL

Vehicle Edge & IoT Event Analytics




Vehicle App
Input
Output
Vehicle App
Input
Output
VEHICLE
VEHICLEIDENTIFICATION
CHASSIS
DRIVETRAIN
VIN
MODEL
BRAND
AXLE
AMBIENTAIRTEMPERATURE
TRANSMISSION
PERFORMANCEMODE
ROW1
ROW2
WHEEL
TIRE
PRESSURE
WHEEL
TIRE
PRESSURE
FUELSYSTEM
LEVEL
FUELTYPE
RANGE
AVERAGECONSUMPTION
INSTANTCONSUMPTION
GENIVI
Vehicle Signal Specification


Runtime
GENIVI
Business logic
Kuksa.VAL Adapter
IoT Event Analytics
4
Vehicle App
5
KUKSA
Kuksa.VAL
Northbound - Common Vehicle Interface
Southbound - Vehicle Abstraction
Signal
Signal 2 VSS
VSS 2 Digital Twin
Vehicle Signals
1
HAL-Interface
2
HAL-Interface Adapter
3