

# Applying VSS in-Vehicle

Patterns and OSS building blocks

September 18<sup>th</sup>

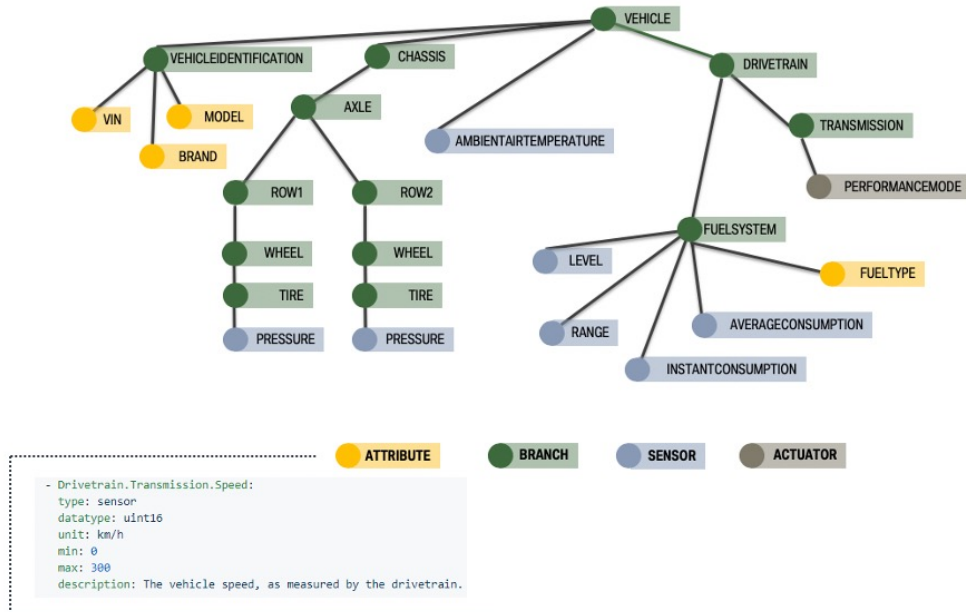


# COVESA

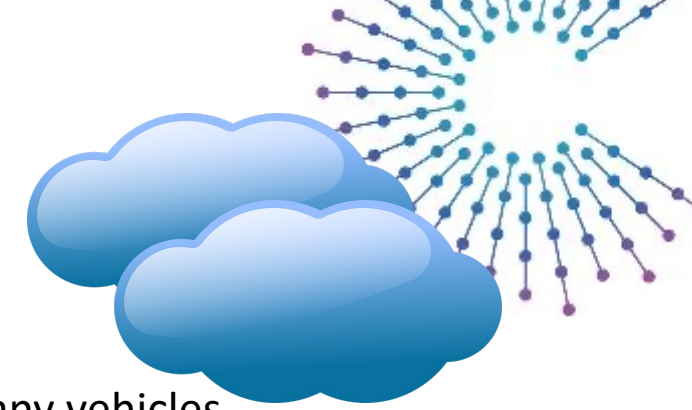
Accelerating the future of connected vehicles

# VSS Recap...

- Common approach for describing vehicle data for machines & humans, and a bit more.
- Provides a common understanding across the value chain of the Connected Vehicle.



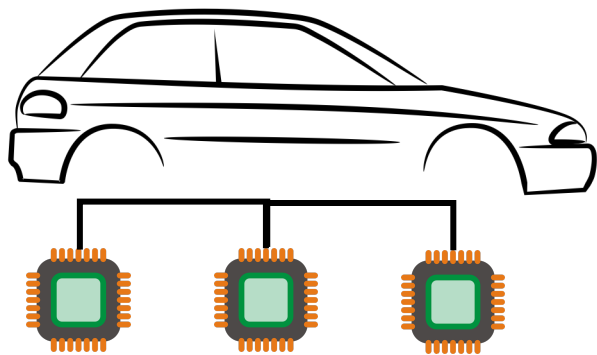
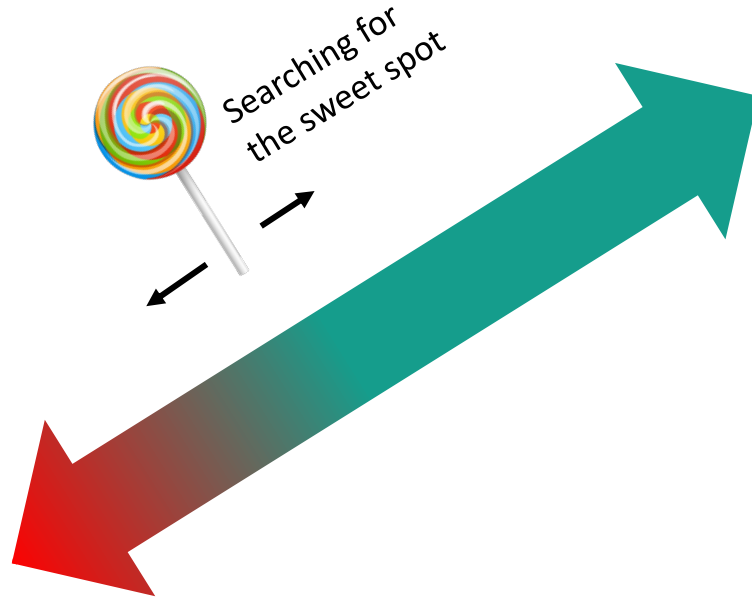
# Where to best leverage VSS?



## Backend

- The cloud
- Aggregating data of many vehicles
- Link data to other domains

You want common data models: VSS



## Deeply-Embedded Layer

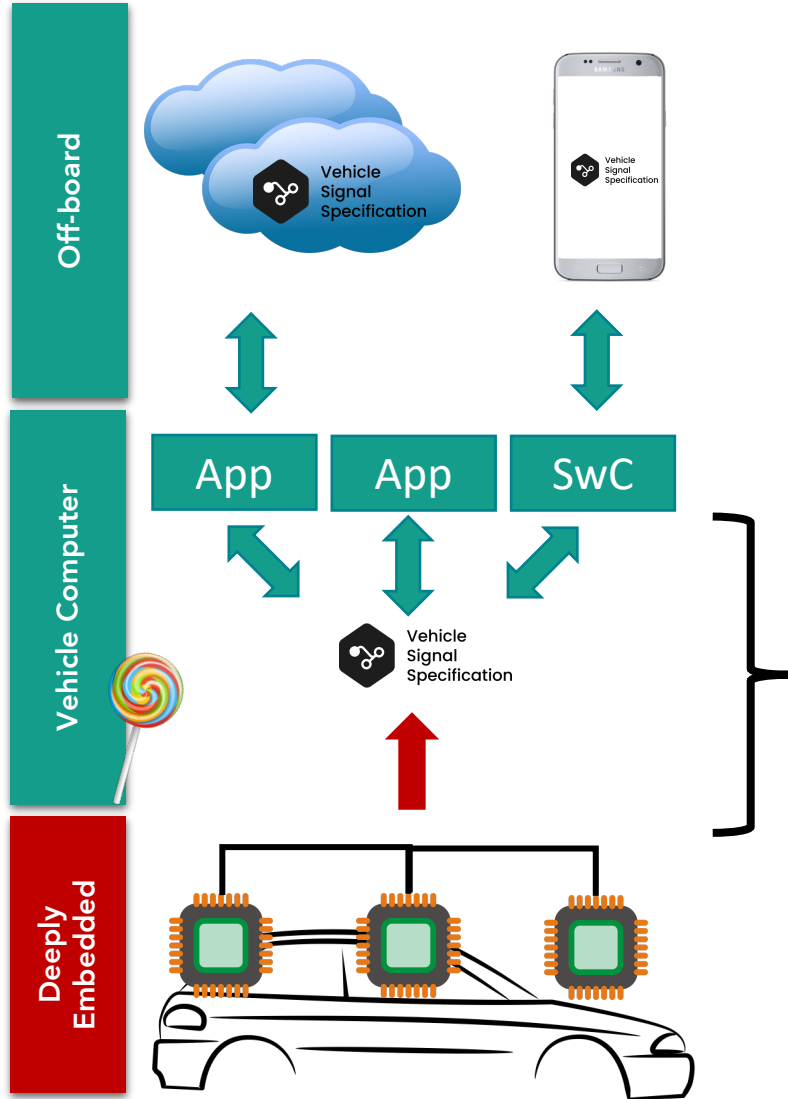
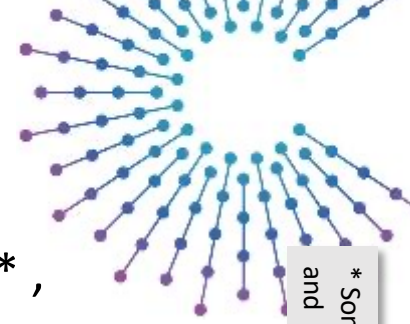
- Small  $\mu$ Cs
- CAN/LIN
- Very proprietary

**Not a happy place for VSS**





# Our Answer



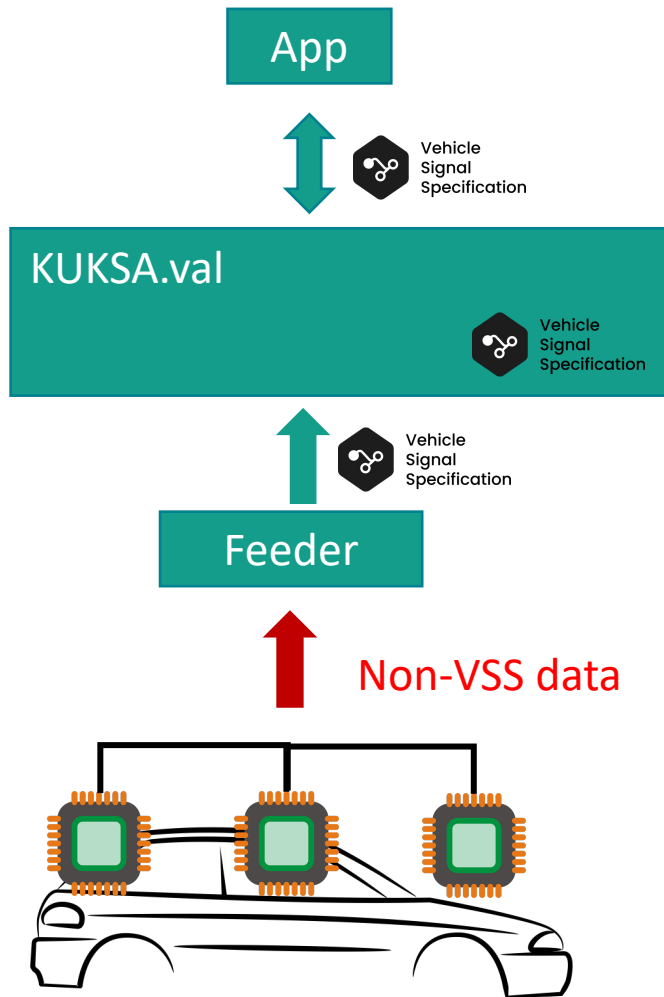
- Start converting to the VSS world in a Vehicle computer\* , because
  - This is the place the industry is working on decoupling hard- from software
  - Here you save money & effort with more generic/portable software
  - Here you can afford the costs of abstraction



- Has you covered transforming signals from different parts of your E/E architecture to VSS.
- Provides secure access to VSS signals using simple to use interfaces

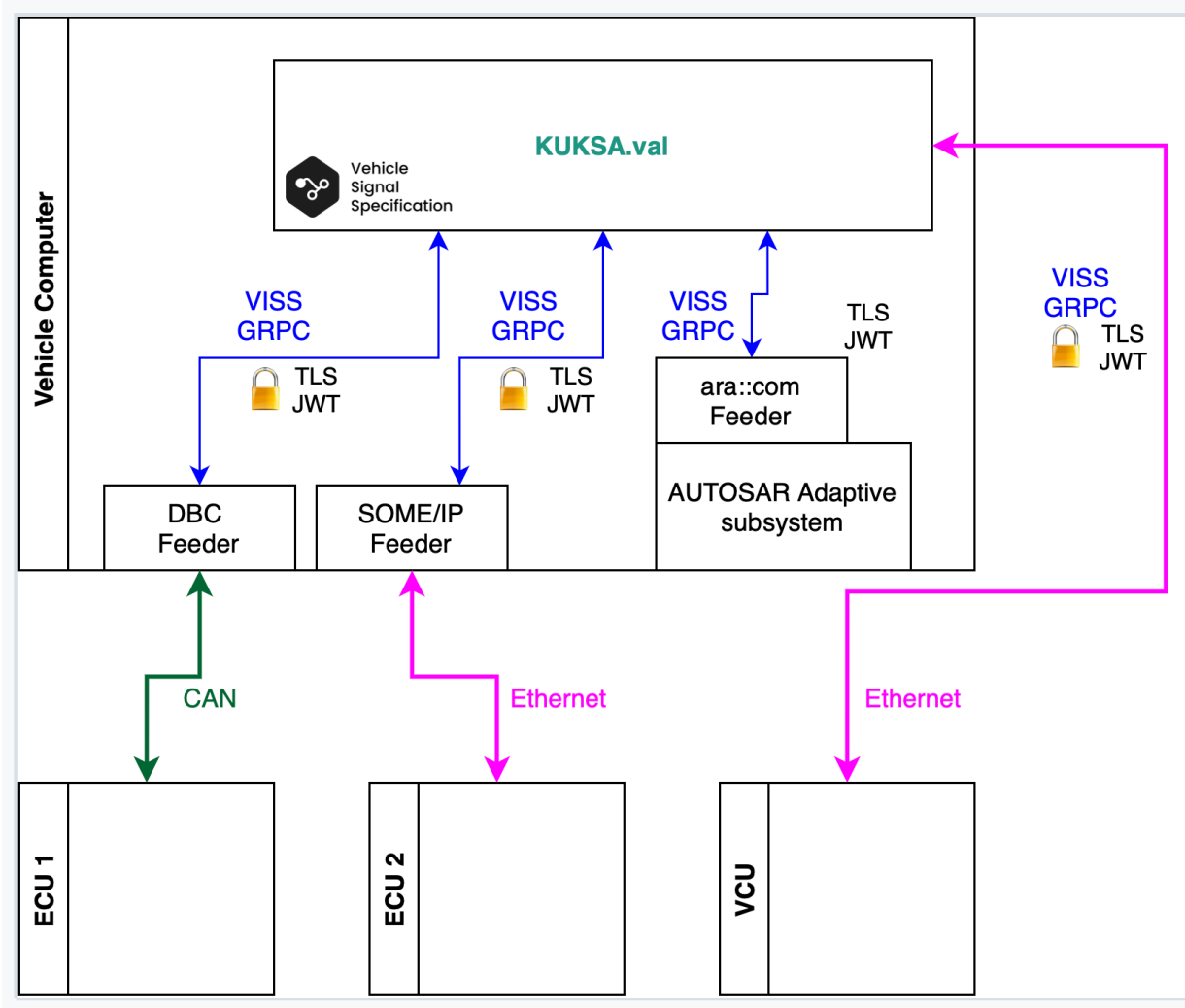
\* Something with a processor and a full blown (POSIX) OS

# KUKSA.val Scope and Design Choices



- 100% Open Source Eclipse Project (Apache 2.0 license)
- "In-vehicle digital twin" based on VSS
- Lightweight (core written in C++/RUST)
- Only providing "current" view (no historic data)
- No access without authorisation
- Easy to use language-agnostic interfaces (VISS/GRPC)
- Data Feeders to transform data to VSS
- Support for simple VSS actors

# Feeder Concept



Doing the heavy lifting of transforming custom signals to VSS format and pushing them to KUKSA.val

Using the standard KUKSA.val VISS or GRPC interface

Fully configurable CAN feeder available as OSS

SOME/IP feeder blueprint available as OSS (soon)

# KUKSA Protocol Options

## KUKSA W3C VISS dialect

- WebSocket + JSON
- Support get/set subscribe (no complex filters, but basic calls and replies should be compliant with W3C VISS V2)
- Extension: KUKSA authorisation
- Extension: Extend/modify VSS tree during runtime
- Extension: You can make a difference between dealing with current values and target values (relevant for actuators)
- Supported by the KUKSA.val server (C++)

## KUKSA GRPC interface

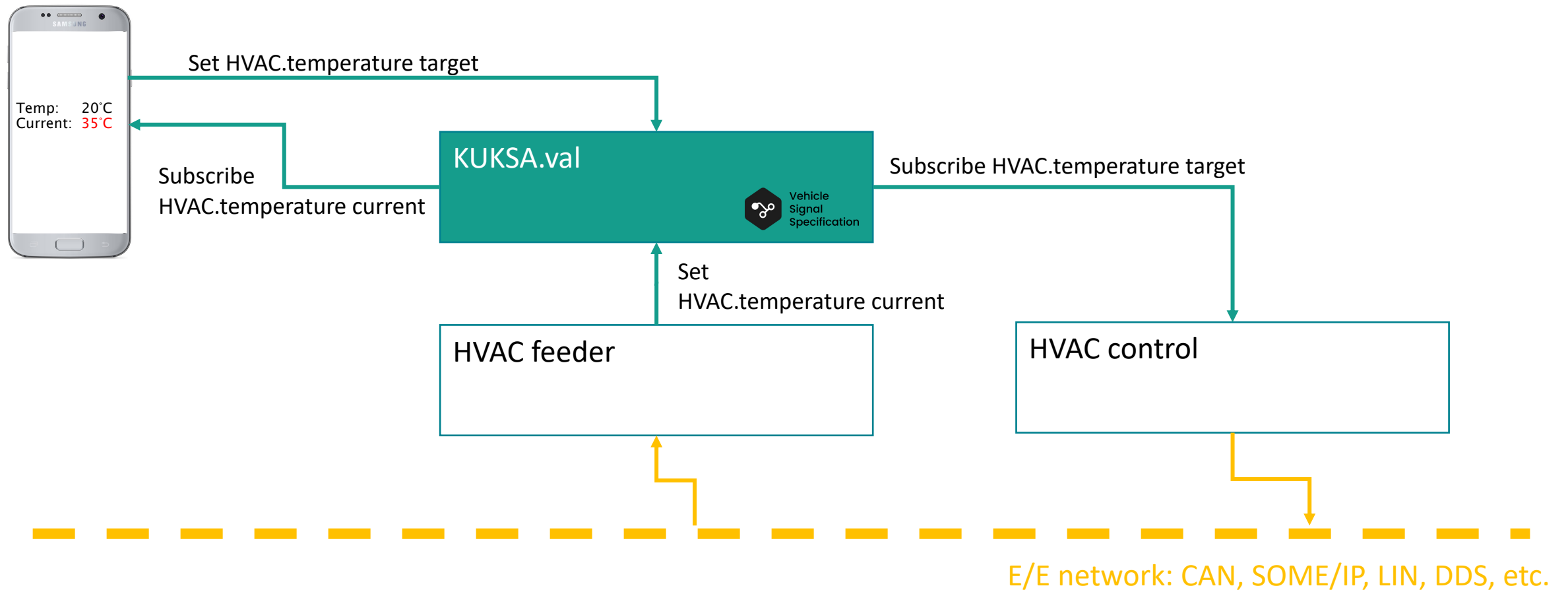
- Binary efficient rpc protocol realised with [grpc.io](https://grpc.io)
- Supports almost any programming language
- Supports get/set subscribe for sensors and actuators
- More fine grained control over which fields/metadata are to be read or set
- (Soon) supported by the KUKSA.val databroker (RUST) and kuksa.val server (C++)



KUKSA Python library for even easier access to basic functionality with just a few lines of code

# Sensors & Actuators in KUKSA.val

- We do not want this to replace any complex RPC/SoA middleware with KUKSA.val, but we do want to be able to “actuate” simple things





# Security Model

## What you get

- TLS for Websocket and GRPC
- No access without JWT based authorisation
- Flat permission model in KUKSA.val

```
{
  "sub": "kuksa.val",
  "iss": "Eclipse KUKSA Dev",
  "admin": false,
  "iat": 1516239022,
  "exp": 1767225599,
  "kuksa-vss": {
    "Vehicle.Speed": "r",
    "Vehicle.Powertrain": "rw"
  }
}
```

## Bring your own

- Key & Certificate Management (Provisioning, Expiration)
- Complex Role based access models
  - Bring you own IDM/IAM!
- Deployment/Granularity:
  - Individual Authorisation for each App/function
  - Authorisation per environment (Runtime/ECU): “All IVI components have access to these signals”

# Beyond KUKSA.val: Eclipse Velocitas

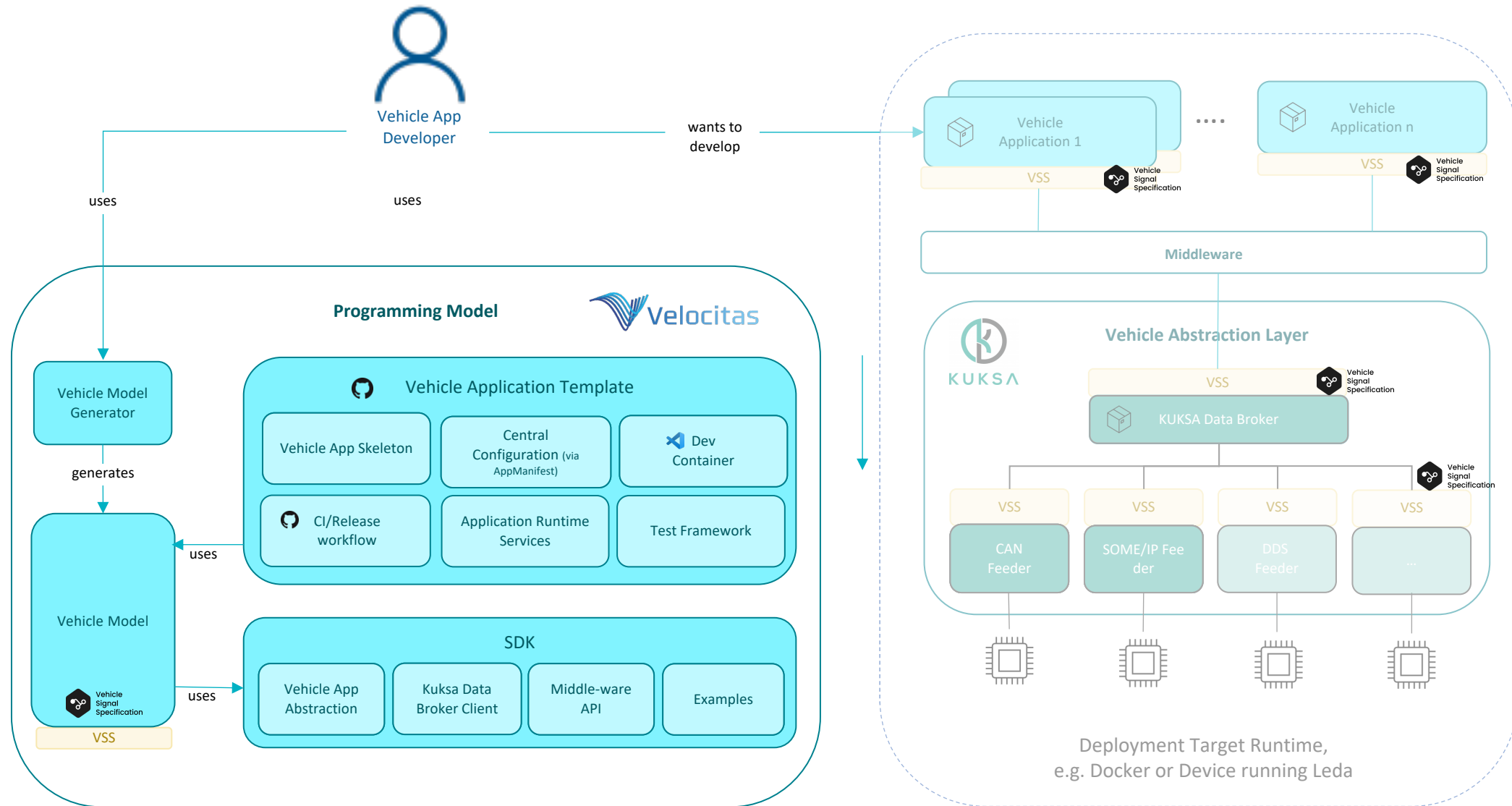
- KUKSA.val has you covered transforming signals to VSS and having them accessible inside a vehicle
- In case you are also searching for solutions
  - How to structure your Vehicle App
  - How to structure your development and delivery workflow
  - How to test, package and deploy in-vehicle Applications

Check out Eclipse Velocitas

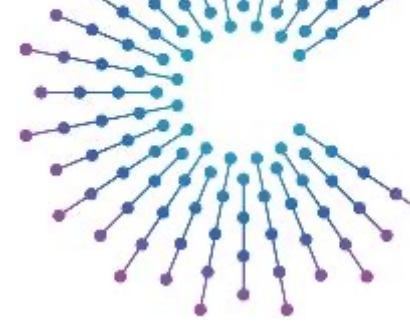
<https://eclipse-velocitas.github.io/velocitas-docs/>





# Eclipse Velocitas Toolchain & SDK



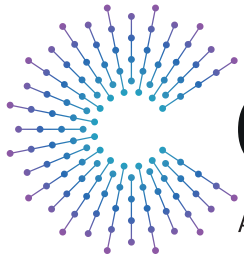
# Takeaway



- The most promising location to **adopt VSS** is in a **Vehicle Computer**
  - It is very feasible cost/effort-wise
  - Here you can start profiting from the benefits of a common data model and “IT-like” technologies
- **You are not alone**, for architecture patterns and Open Source software components check out
  -  <https://www.eclipse.org/kuksa/>
  - <https://github.com/eclipse/kuksa.val>
- For a more complete, opinionated solution how to develop, test and deploy Vehicle Applications, check out Eclipse Velocitas. Batteries (KUKSA.val) already included!
  -  <https://eclipse-velocitas.github.io/velocitas-docs/>



Both projects are proud members of <https://sdv.eclipse.org> and open to any interested party



# COVESA

Accelerating the future of connected vehicles



KUKSA

## Thank you

**Stay in contact**

<https://github.com/eclipse/kuksa.val>

<https://eclipse.org/kuksa>

[sebastian.schildt@de.bosch.com](mailto:sebastian.schildt@de.bosch.com)