VSS to Franca to SOME/IP connections

Prepared by:

Gunnar Andersson,
Development Lead at GENIVI Alliance

10 September 2019

GENIVI®

CREDITS

- For VSS introduction these are reused slides from

  except for the “Thin API” which was added

Bringing the Car to the Internet

October 10, 2018 |
GENIVI and W3C – Enabling the Connected Car through Collaboration

Rudolf J Streif

Conventional Approach – “Fat API”

- An API for every signal or control:

```
var vehicle = navigator.vehicle;
vehicle.vehicleSpeed.get().then(function (vehicleSpeed) {
    console.log("Vehicle speed: " + vehicleSpeed.speed);
}, function (error) {
    console.log("There was an error"); });
var vehicleSpeedSub = vehicle.vehicleSpeed.subscribe(function (vehicleSpeed) {
    console.log("Vehicle speed changed to: " + vehicleSpeed.speed);
    vehicle.vehicleSpeed.unsubscribe(vehicleSpeedSub);
});
```

- Issues with this approach:
  - Addition of new signals and controls requires change of the specification.
  - Challenges maintaining backwards compatibility.
  - Complexity in providing per-API authorization and access control.
  - Single end-point addressing.

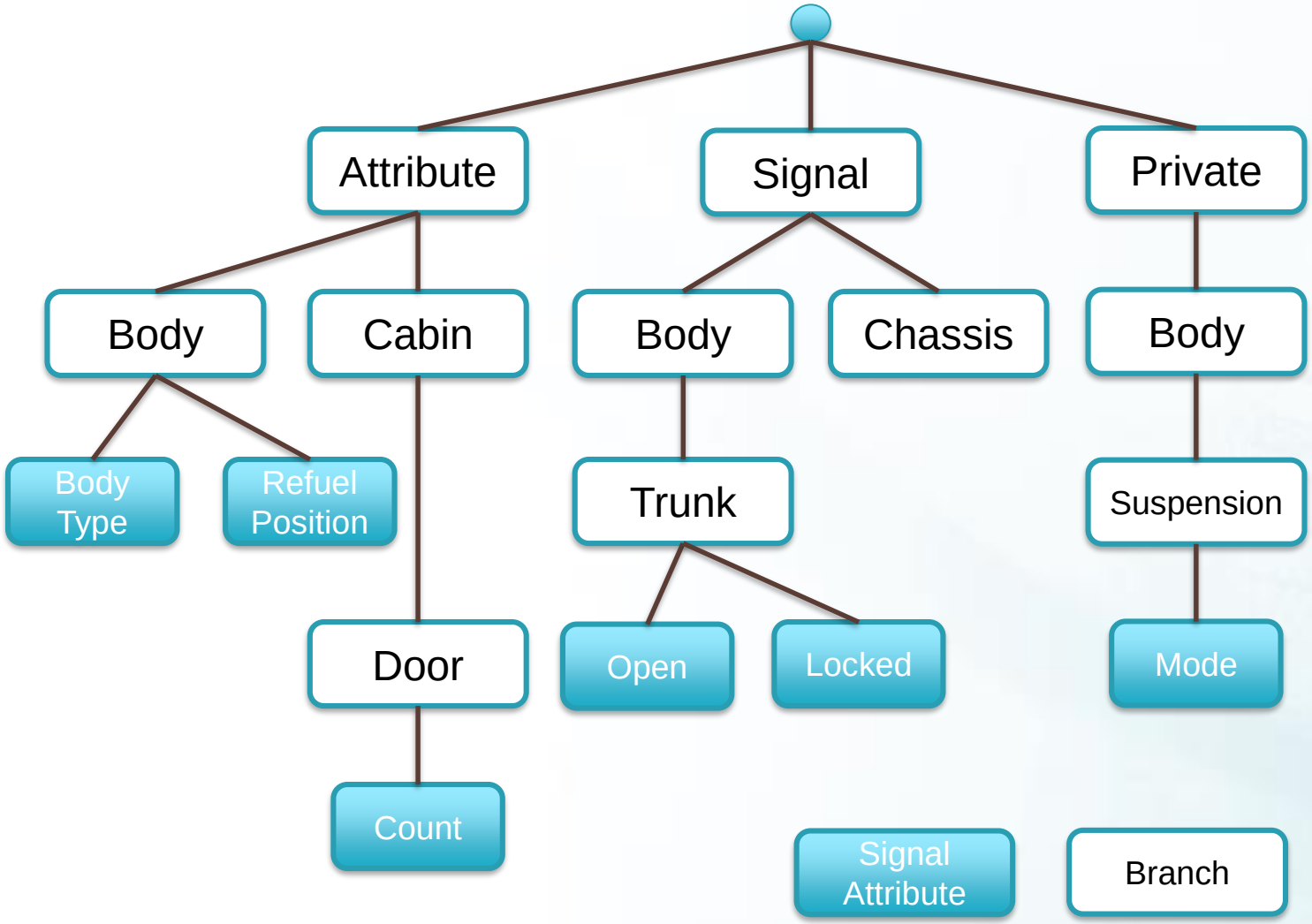
Dynamic “Thin API”

- One common access function for all data.
- For example a request by name or ID.
  Typically a string but a more efficient Identifier is theoretically possible
  getSignal(string signalname)
  - +Create/Update/Modify, similarly
- **Advantages**:
  - Flexible – named data item can exist or not exist.
    Add data dynamically if needed, add new services as needed.
  - Reasonable to divide services up dynamically
    - e.g. some service provides a subset of the whole data tree
  - More WWW-like, Web protocols are often more like this.
- **Disadvantages**:
  - No compile-time checking that signal/API actually exists
  - Potentially less efficient to send the signal name (string) in, and on the service side “look it up”

New Approach – Services with Signal Tree

- The core services *get*, *set*, *subscribe*, *unsubscribe*, *getVSS* and *authorize* are provided by a network server.
  - The services *get*, *set*, *subscribe* and *unsubscribe* provide access to vehicle signals and controls.
  - The service *getVSS* allows clients to query the server for available signals.
  - Using the *authorize* service, the client presents a security token to the server for authentication and authorization.
- Vehicle Signals and Controls are identified as nodes of a vehicle signal tree.
  - A fully qualified signal name addresses a single signal node.
  - Wildcards for branches and node names provide for addressing of signal groups.

Vehicle Signal Tree



- Tree structure provides for hierarchical access to signals and attributes.
- Branches group signals and attributes into entities that logically belong together.
- Wildcards allow access to entire sets of signals.

Addressing

```
Signal.Chassis.Brake.FluidLevel
Signal.Drivetrain.FuelSystem.Level
Attribute.Cabin.Door.Count
Attribute.Engine.Displacement
```

- Dot-notation for name path.
- Last path component, called node, represents the signal or attribute.
- Leading path components represent the branches.
- Wildcards can be used to address multiple signals and/or branches.

Specification Format

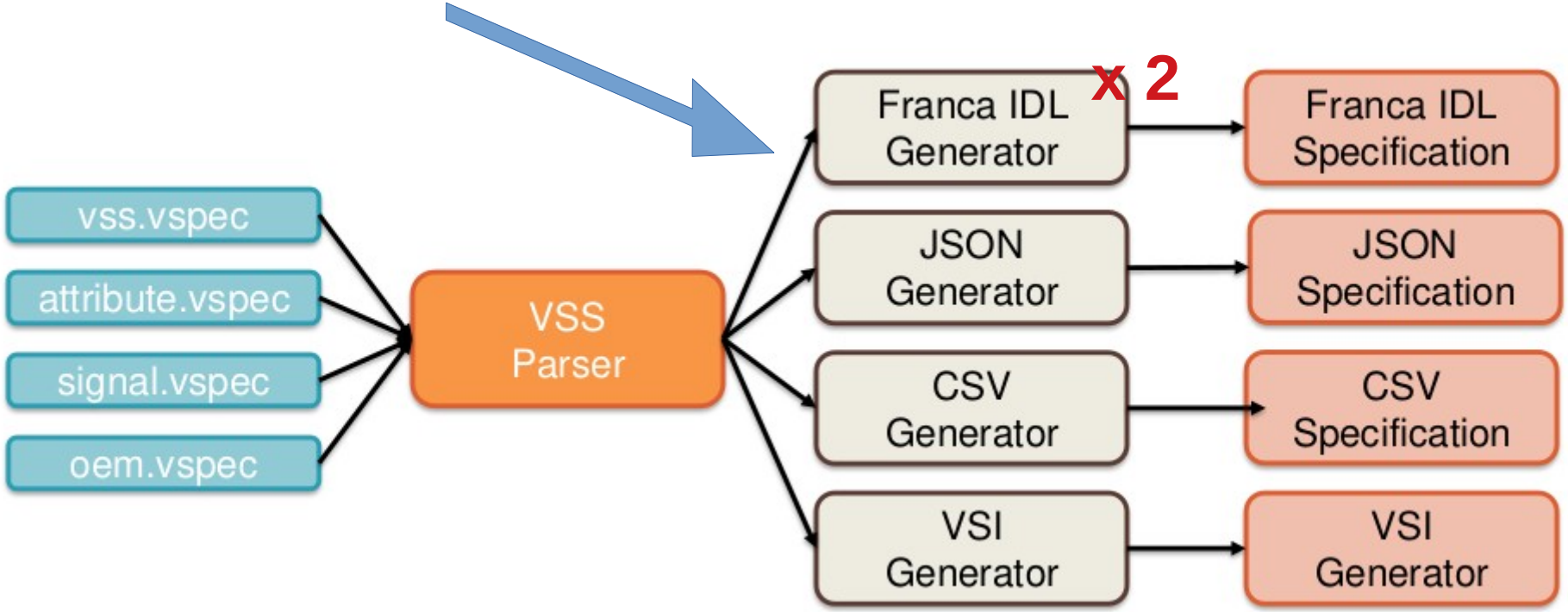
```
- Signal.Drivetrain.Transmission:
  type: branch
  description: Transmission-specific data
- Signal.Drivetrain.Transmission.Speed:
  type: Int32
  min: -250
  max: 250
  unit: m/s
  description: Current vehicle speed, sensed by gearbox
```

- Formatted as YAML lists
- Simple conversion into other formats such as JSON, France IDL, CSV, and more
- # denotes a comment or a directive

Creating signal access with VSS + SOME/IP

- **SOME/IP** (and other communication methods) are available with CommonAPI framework & tooling
- **capicxx-someip-runtime** connects the CommonAPI high-level C++ interface with the **vSOMEIP** implementation
- Observable “properties” are named Attributes in Franca IDL
- Franca **Attributes** are supported by CommonAPI and in this case they become SOME/IP properties
- *A “Thin API” could be created. Thee usage of an Interface Description Language has a smaller impact for such a small number of functions.. However, we get a useful high-level C++ programming API from the CommonAPI tooling. In this case it creates a SOME/IP remote procedure call capable interface automatically, if we use Franca + CommonAPI.*
- Note however, the following example shows converting each VSS signal to an attribute. Thus it is an example of a “FAT API” according to previous introduction.

- Previous Franca generators created a datastructure holding definitions from the VSS
- This calls for an access API instead (using Attributes/Properties)
- Add new Franca IDL Attribute generator!



Running vspec2franca_attributes 1

./tools/vspec2franca_attributes.py
-p Vehicle.Drivetrain
-m Vehicle.Drivetrain.FuelSystem
-n FuelSystem
-v $(cat VERSION)
-i:spec/VehicleSignalSpecification.id
-I ./spec ./spec/VehicleSignalSpecification.vspec
vss_rel_$(cat VERSION)_attributes.fidl

We want to generate the interface named **FuelSystem**,
from all signals matching the VSS sub-tree **Vehicle.Drivetrain.FuelSystem** (only)
and put it in a package named **Vehicle.Drivetrain**

cat vss_rel_2.0.0-alpha+006_attributes.fidl

Example results

```
// Copyright (C) 2019
// Contributors to Vehicle Signal Specification
// (https://gitub.com/GENIVI/vehicle_signal_specification)
//
// This program is licensed under the terms and conditions of the
// Mozilla Public License, version 2.0.  The full text of the
// Mozilla Public License is at https://www.mozilla.org/MPL/2.0/

const UTF8String VSS_VERSION = "2.0.0-alpha+006"

// Vehicle signal attributes generated from VSS specification version ...

package Vehicle.Drivetrain {

  interface FuelSystem {

    attribute string HybridType                  /* Vehicle.Drivetrain.FuelSystem.HybridType */
    attribute uint16 TankCapacity                /* Vehicle.Drivetrain.FuelSystem.TankCapacity */
    attribute float ConsumptionSinceStart       /* Vehicle.Drivetrain.FuelSystem.ConsumptionSinceStart */
    attribute boolean EngineStopStartEnabled      /* Vehicle.Drivetrain.FuelSystem.EngineStopStartEnabled */
    attribute uint8 Level                       /* Vehicle.Drivetrain.FuelSystem.Level */
    attribute float InstantConsumption          /* Vehicle.Drivetrain.FuelSystem.InstantConsumption */
    attribute uint32 TimeSinceStart              /* Vehicle.Drivetrain.FuelSystem.TimeSinceStart */
    attribute boolean LowFuelLevel                /* Vehicle.Drivetrain.FuelSystem.LowFuelLevel */
    attribute uint32 Range                       /* Vehicle.Drivetrain.FuelSystem.Range */
    attribute float AverageConsumption          /* Vehicle.Drivetrain.FuelSystem.AverageConsumption */
    attribute string FuelType                    /* Vehicle.Drivetrain.FuelSystem.FuelType */
  }
}
```

Running vspec2franca_attributes 2

./tools/vspec2franca_attributes.py
-p Vehicle
-m Vehicle.Drivetrain.FuelSystem
-n Drivetrain
-v $(cat VERSION)
-i:spec/VehicleSignalSpecification.id
-I ./spec ./spec/VehicleSignalSpecification.vspec
vss_rel_$(cat VERSION)_attributes.fidl

We want to generate the interface named **Drivetrain**,
from all signals matching the VSS sub-tree **Vehicle.Drivetrain.FuelSystem** (only)
and put it in a package named **Vehicle**

cat vss_rel_2.0.0-alpha+006_attributes.fidl

Example results

```
// Copyright (C) 2019
// Contributors to Vehicle Signal Specification
// (https://gitub.com/GENIVI/vehicle_signal_specification)
//
// This program is licensed under the terms and conditions of the
// Mozilla Public License, version 2.0.  The full text of the
// Mozilla Public License is at https://www.mozilla.org/MPL/2.0/
```

```
const UTF8String VSS_VERSION = "2.0.0-alpha+006"

// Vehicle signal attributes generated from VSS specification version ...

package Vehicle {

  interface Drivetrain {

   attribute string FuelSystem.HybridType          /* Vehicle.Drivetrain.FuelSystem.HybridType */
   attribute uint16 FuelSystem.TankCapacity        /* Vehicle.Drivetrain.FuelSystem.TankCapacity */
   attribute float FuelSystem.ConsumptionSinceStart /* Vehicle.Drivetrain.FuelSystem.ConsumptionSinceStart */
   attribute boolean FuelSystem.EngineStopStartEnabled /* Vehicle.Drivetrain.FuelSystem.EngineStopStartEnabled */
   attribute uint8 FuelSystem.Level                /* Vehicle.Drivetrain.FuelSystem.Level */
   attribute float FuelSystem.InstantConsumption  /* Vehicle.Drivetrain.FuelSystem.InstantConsumption */
   attribute uint32 FuelSystem.TimeSinceStart      /* Vehicle.Drivetrain.FuelSystem.TimeSinceStart */
   attribute boolean FuelSystem.LowFuelLevel        /* Vehicle.Drivetrain.FuelSystem.LowFuelLevel */
   attribute uint32 FuelSystem.Range               /* Vehicle.Drivetrain.FuelSystem.Range */
   attribute float FuelSystem.AverageConsumption  /* Vehicle.Drivetrain.FuelSystem.AverageConsumption */
   attribute string FuelSystem.FuelType            /* Vehicle.Drivetrain.FuelSystem.FuelType */
  }
```

Conclusions

- CommonAPI + Franca IDL is an already existing way to get a high-level C++ programming interface that connects to SOME/IP communication.
- Converting VSS data to some kind of Franca interface enables leveraging this path to SOME/IP
- The generated code could be used to implement a vehicle data server, but it's also possible that some vehicle data servers will be provided by AUTOSAR systems and use other software bindings to define the SOME/IP service. Since CommonAPI generates both client and server APIs, the client API might still be useful to connect to the service, if the AUTOSAR vehicle signalling system is based on a VSS-style description of the data, and compatible SOME/IP usage

(Also, the Franca2ARA translation tool might be useful)

- Proof-of-concept implementation of VSS-to-Franca Attributes is available (open for changes)