Evolving Virtual Connected Vehicles - An Experience Report

Sören Frey, Albrecht Messner, Alex Stasewitsch, Jens Nahm
Daimler TSS GmbH, Gropiusplatz 10, 70563 Stuttgart, Germany
{soeren.frey, albrecht.messner, alex.stasewitsch, jens.nahm}@daimler.com

Abstract

We report on our experiences and observations from the automotive industry on a multi-year evolution of a complex telematics system that provides remote access to connected vehicles.

1 Introduction

Since several years, the service portfolios offered by automotive manufacturers are increasingly blended with vehicle telematics solutions [1]. For example, commercial and private customers can utilize connectivity-enabled fleet management and carsharing services, respectively. We give an industrial experience report that provides insights in the evolution history of a complex telematics system that implements a pivotal functionality for those kinds of services: connecting vehicles and backend infrastructure. As the system constitutes an abstract representation of a connected vehicle, it is called a *Virtual Connected Vehicle* or *Virtual Vehicle* [2] for short.

The paper is structured as follows. Section 2 explains the context of the *Virtual Vehicle* system. Section 3 reflects upon the system's evolution by describing its four major architecture versions and the corresponding transitions. Section 4 draws the conclusions.

2 Background

Connected vehicles exchange data with backend systems with the help of telematic control units (TCUs). A TCU is an in-vehicle embedded system that sends and receives data via a mobile communication interface. A TCU may transmit data it receives from directly connected or internal sensors, e.g. from a GPS device, or exchange data with other in-vehicle embedded systems via the common controller area network (CAN) bus standard. This can encompass reading vehicle data (e.g. mileage) and sending them to the backend. The TCU may also receive vehicle commands from the backend (e.g. a *door unlock* command) and write corresponding signals to the CAN bus.

The *Virtual Vehicle* system itself runs in the backend and constitutes the logical endpoint for communicating with connected vehicles. It also provides a high-level API for 3rd party systems (e.g. fleet management portals, call centers, and consumer mobile applications) for interacting with the vehicles.

For exchanging messages with a vehicle, the *Virtual Vehicle* system has to hold and take into account the vehicle's state. Depending on the data format used for message transmission, the system has to translate between low-level CAN messages and the high-level API, too. As various model series and 3rd party systems with a varying number of users have to be supported in a secure and safe manner, extensibility, performance, scalability, security, and maintainability constitute particularly important quality attributes.

3 Major Architecture Versions

The four major architecture versions (AV1-AV4) of the *Virtual Vehicle* are outlined in Figure 1 and described in the following. The system was developed over several years with varying team compositions. We also highlight the seven specific main challenges (CH1-CH7) of AV1-AV3 that drove the evolution.

3.1 AV1: Initial Monolith

Overview The `TCU` exchanges raw *CAN Data* messages with the *Virtual Vehicle* via a `Message Queue`. A `Router` component compresses and decompresses the messages and routes incoming messages to different instances of the *Virtual Vehicle Service* (`VVS`) component. This allows, for example, to address different staging environments and to separate commercial from private customer vehicle data. The `VVS` stores a vehicle's state and its master data via the *Master Data Management* (`MDM`) component. The interpretation of CAN messages (e.g. deriving the vehicle's mileage from a CAN frame's bit sequence) is accomplished by the *Static CAN Interpreter Service* (`Stat. CIS`).

Challenges (CH1) Originating from a proof of concept project, `Stat. CIS` includes the idiosyncrasies of specific vehicle types and model series as hard-wired, static constituents. Corresponding changes are tedious and require a new deployment of the monolithic `VVS` component. (CH2) Furthermore, a `3rd Party` can not only receive actual *Vehicle Data* points (e.g. mileage) that were already interpreted, but it can also receive raw *CAN Data* via the `VVS` or directly from the `Message Queue`. This poses a security risk.

3.2 AV2: Improved Modularity

Overview CH1 is addressed by improving the modularity and extensibility of the *Virtual Vehicle* sys-

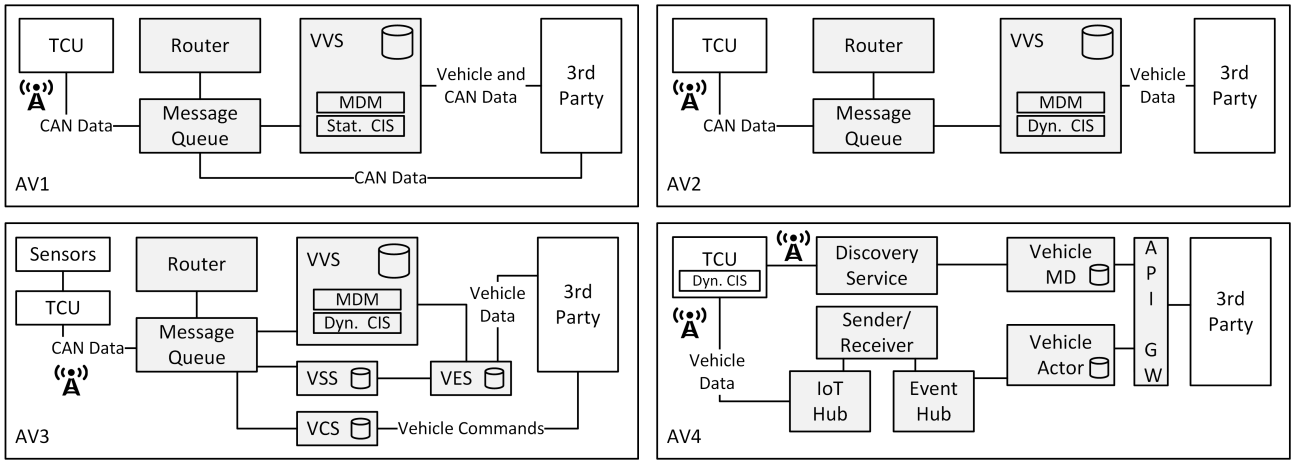

Figure 1: The evolution of the *Virtual Vehicle* (gray) along its four major architecture versions (AV1-AV4)

tem. `Stat. CIS` is replaced by the *Dynamic CAN Interpreter Service* (`Dyn. CIS`) component. Hence, peculiarities of specific vehicle types and model series can now be separated in independent modules. These modules can be modified dynamically and are used in conjunction with `Dyn. CIS` for interpreting *CAN Data*. For tackling CH2, a `3rd Party` can no longer interact with vehicles by exchanging raw *CAN Data*. This and some later changes require those external components to co-evolve with the *Virtual Vehicle*.

Challenges (CH3) The system's scalability has to be improved to serve an increased number of drivers (`TCU`s) and more `3rd Party` stakeholders as the *Virtual Vehicle* system moves into a piloting phase. (CH4) Moreover, the modularity has to be improved further to effectively cope with new requirements. For example, data points from external sensors in vehicles (such as temperature sensors for ensuring a continuous cooling chain), that are not available via the CAN bus, have to be processed.

3.3 AV3: Microservices Architecture

Overview CH3 and CH4 are addressed by restructuring the *Virtual Vehicle* system towards a microservices architecture utilizing, among others, container technology. Each microservice has its own database and scaling out is easier (CH3). Furthermore, the different concerns are now separated more strictly (CH4). For example, `VVS` now only handles vehicle master data and data points. *Vehicle Commands* (that result in write accesses to the CAN bus) are processed by the new *Vehicle Control Service* (`VCS`) that can also serve a `3rd Party`. Sensor data can be interpreted by the new *Vehicle Sensor Service* (`VSS`) and, similarly to other *Vehicle Data*, be provided to a `3rd Party` by the new *Vehicle Event Service* (`VES`).

Challenges (CH5) For worldwide availability, the system shall be migrated to a commercial Platform as a Service (PaaS) offering. (CH6) The performance for transmitting and processing vehicle data and commands should be increased further and (CH7) maintainability should be improved by reducing the usage of low-level *CAN Data* in the *Virtual Vehicle*.

3.4 AV4: PaaS-Based Architecture

Overview The *Virtual Vehicle* system is aligned with the framework of the chosen PaaS (CH5). The `Discovery Service` determines the appropriate backend environment for a given `TCU` per connection establishment. Then, the `IoT Hub` is used for communicating with the vehicle. Tasks like header validation and data compression are delegated to the `Sender/Receiver` component. Event propagation is performed by the `Event Hub` instead of `VES`. `3rd Party` components now use the *Virtual Vehicle* via a PaaS-enabled *API Gateway* (`API GW`). Vehicle master data is handled by the `Vehicle MD` component. The `Vehicle Actor` processes the vehicle data. CH6 and CH7 are addressed by moving the interpretation of CAN messages (`Dyn. CIS`) to the `TCU`. *Virtual Vehicle* now only receives more lightweight *Vehicle Data* instead of *CAN Data*. New vehicle types and model series do not have to be added to the *Virtual Vehicle*, but only to the `Dyn. CIS` of the `TCU`.

4 Conclusions

As confirmed by the development history of the *Virtual Vehicle* system, complex systems need to continuously evolve to mature and retain business value. Moreover, building on feature-rich platforms, e.g. cloud services, is vital for a short time-to-market.

References

[1] Yilin Zhao. Telematics: safe and fun driving. *IEEE Intelligent Systems*, 17(1):10–14, Jan 2002.

[2] S. Frey, L. Charissis, and J. Nahm. How software architects drive connected vehicles. *IEEE Software*, 33(6):41–47, Nov 2016.