Draft 4 – For circulation and feedback.

Common Vehicle Interface Initiative (CVII)

Introduction

GENIVI and W3C have enjoyed a productive collaboration that has resulted in a series of industry-relevant standards such as Vehicle Service Specification (VSS) and Vehicle Interface Service Specification (VISS). These are both expanding into new initiatives including a standard vehicle service catalog (VSC) and related web-vehicle remote procedure calls. The combination of these existing and new activities form a strong foundation for CVII.

As envisioned today, CVII is an industry-wide, OEM-led dialog on the potential of joint development and adoption of common vehicle data models, access protocols and standard interfaces in the entire scope of the vehicle plus the cloud. This, of course, touches standards and outputs of many existing organizations and thus, requires an inclusive approach, well-defined scope and work breakdown as well as a coordinated effort to deliver adoptable solutions.

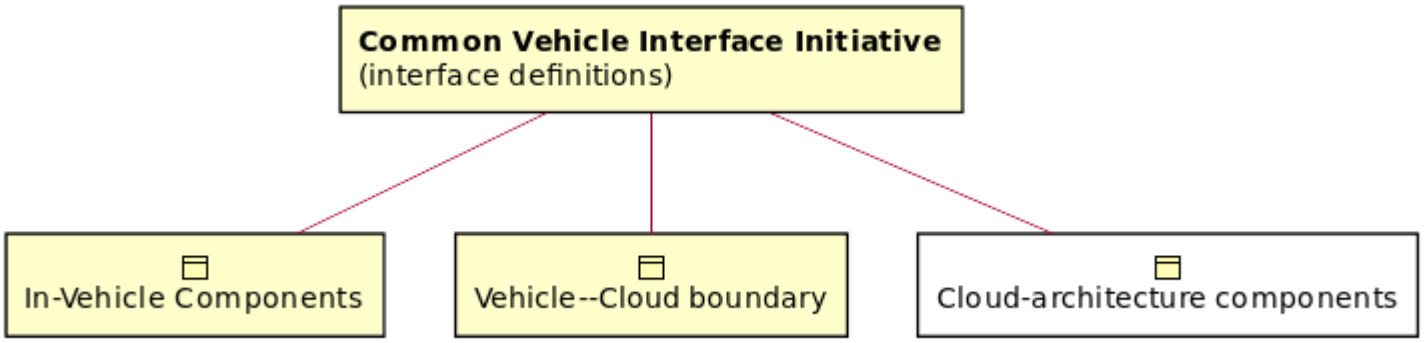
This document *does not identify or describe in detail the business value and new opportunities presented by a common vehicle interface*. Value and opportunities will be described differently by various OEMs and industry stakeholders and thus, are outside of the scope of this document. This document provides a series of illustrations and questions that lead OEMs and other stakeholders to a *more precise definition of the scope of CVII* so that this important work can begin. GENIVI/W3C invites readers of this document to provide feedback to Gunnar Andersson, Technical Lead at GENIVI (gandersson@genivi.org) or to Ted Guild, Connected Vehicle Lead at W3C (ted@w3c.org).

CVII scopes

CVII originally highlighted two main areas of need for “common interfaces”

- The vehicle-cloud interface (remotely accessible functions on the vehicle)
- The in-vehicle decomposition of the system (interfaces between software components, or other subsystem boundaries).

However, looking at the entire architecture, it should be recognized that the cloud-architecture is naturally also decomposed into parts, and the *common interface* aspect is highly relevant also there, for example in a *web app developer* context. The developer needs to access vehicle related data and that is already stored in the cloud and needs to have a single programming interface that gives access to data from many car brands.



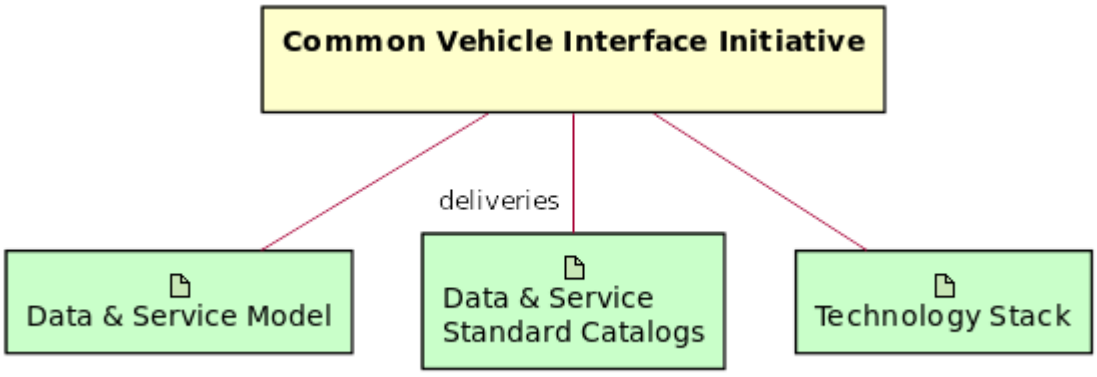
Q: Do the CVII participants agree that all these aspects are in scope and part of the "CVII project"?

Q: Which is the primary interest (if any) for your company?

Q: The data/functions used in the cloud connect to in-vehicle components eventually, so alignment is very beneficial, but for the technology stack definition is it worthwhile to explore these aspects separately?

Comment: In the full organization of CVII there will be *different consortia* interested in each aspect mentioned here.

CVII potential project deliveries



Technology stack

This chapter defines the term Technology Stack in the context of CVII

To complementing the data/service model, there are the bindings and protocols that define the technical details for how software parts actually communicate to transfer the data items defined in a catalog and to invoke the defined service functions.

In addition there are tools that process or convert the data model description to other formats, generate executable code, or perform other analysis of the information.

Finally, there are fundamental implementations of the core technology, for example in the form of code libraries that implement “binding” to existing communication technologies.

We will refer to this collection of *primarily software* implementations that run a functioning system as the *Technology Stack*.

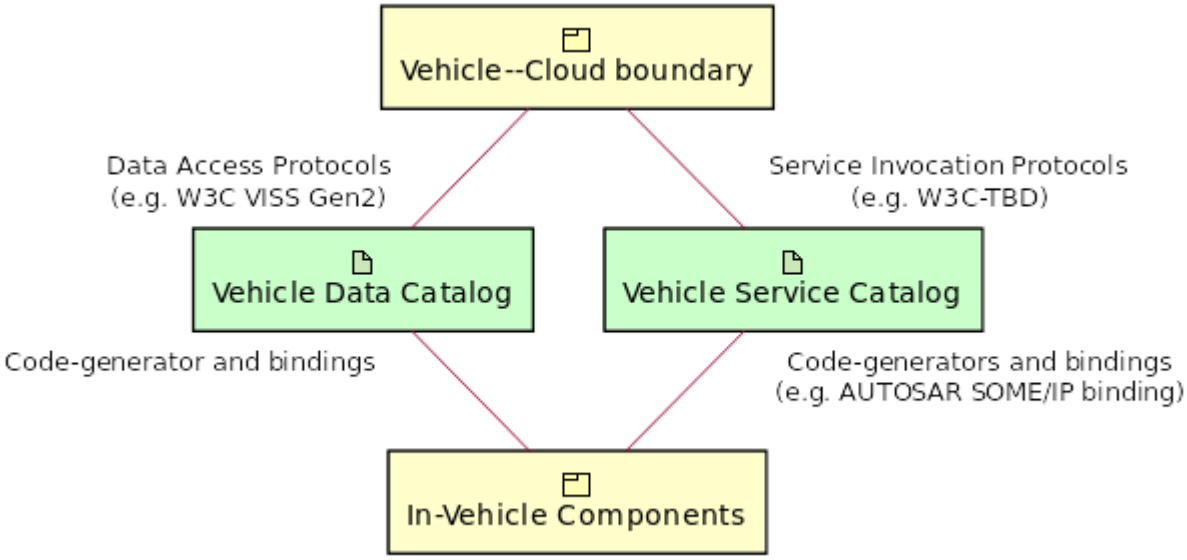
The CVII project can define the technology stack in two major ways

1. Find and identify, or develop, specific *implementations* (typically open-source licensed for maximum reach and availability)
   or
2. Write strict behavior *specifications* that may be used by any party to implement a fully compatible instance of the software part that it describes.

In the case of the latter, it is particularly important for the project to discuss the availability (conditions, price, available competition, future proofing, etc.) of implementations of that specification, so that the identified choice does not create significant obstacles for successful implementation of the whole technology stack.

Technology stack usage in the system

The technologies mentioned on each association line processes the catalog and creates actual executable software (complete, or a framework to be filled in by human programmers) that match the catalog definitions:



Agreement on the scope of the Technology Stack definition

Whereas the definition of the data catalog and data model should be quite unambiguous (see previous chapters), ideas about the connecting technologies and implementations are often more varied.

Note that the data/service models, the standard catalogs, and some implementations of protocols and bindings already exist. This in itself provides tremendous value, and may even provide value *even without* the technology stack being defined in the project scope. A technology stack is of course required for product development, but it *may* be possible for the industry to fill that gap outside of the CVII collaboration project.

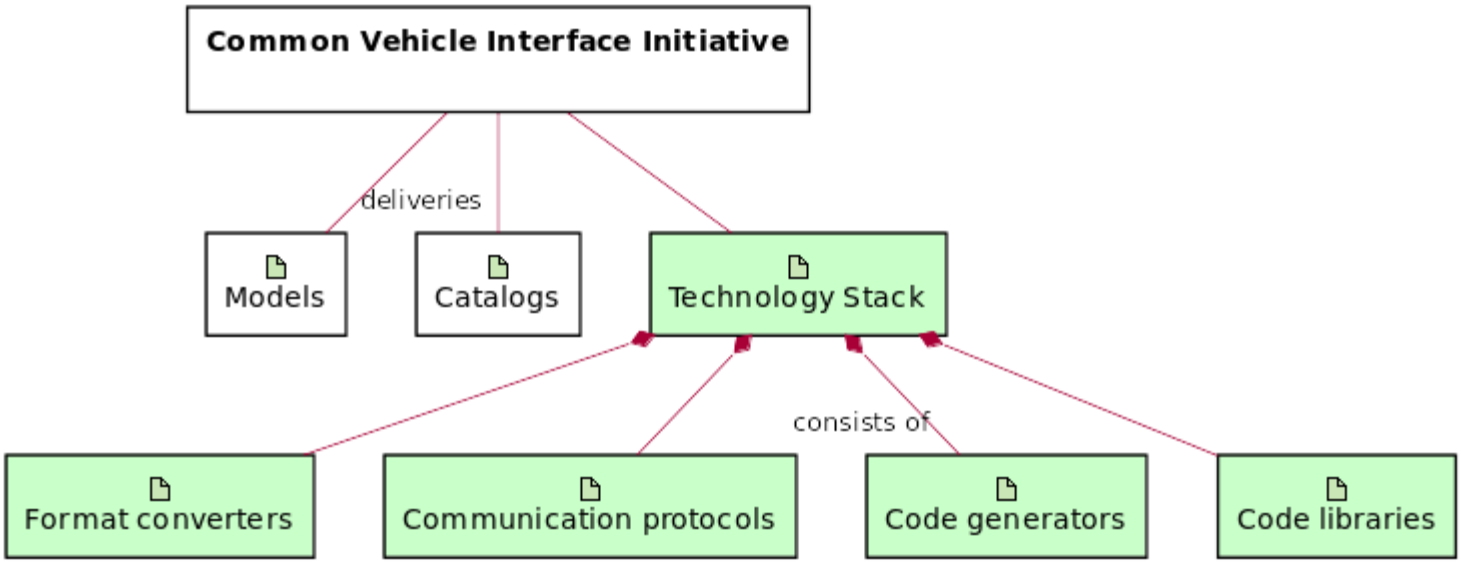
In particular, adherence to underlying model and data/function catalog will act as a common compatible *thread* throughout the transfers across a (partially even proprietary) technology stack. As long as the technology stack aims to maintain the characteristics that the model has defined (data type, semantic meaning, and other behavior), then it is likely that whatever comes out the other end is understandable by looking back at the model and catalog. Semantic compatibility can laregly achieve this *even after* it has been transferred through multiple layers of technology stack parts. These parts might include varied network protocols, technology bindings and code libraries, automated code generation and format translations. The data/service *model* and *catalog* is the key that ties all of it together.

Thus the development of data/interface model and shared catalog may provide value even if some of the technology stack implementation were to be left open for production projects to sort out.

Obviously, a lot of advantages would however be lost by leaving the technology stack definition out of scope for the joint project! The CVII participants have the *opportunity* to actively agree on and define a comprehensive and robust technology stack in a full-featured manner. This includes *deciding* on the protocols and bindings and *ensuring* that (at least one) fully robust implementation is made available.

Our first order of business should be to agree on how well defined or diverse the industry would like the technology stack to be (as an output of CVII), and how the common parts shall be developed.

Technology Stack contents



Q: To what extent do CVII participants wish to *define* the required/recommended technology stack as part of the collaborative project?

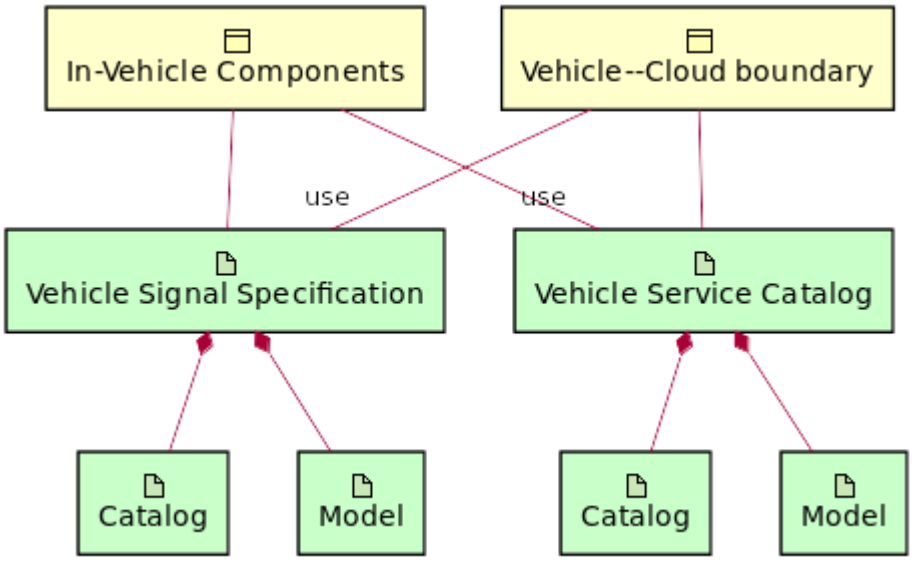
Q: To what extent do CVII participants wish to *implement* the software in the technology stack in collaboration (compared to only writing specifications and leaving it up to the technology-supplier network, or in-house development, to provide implementation?)

Summary

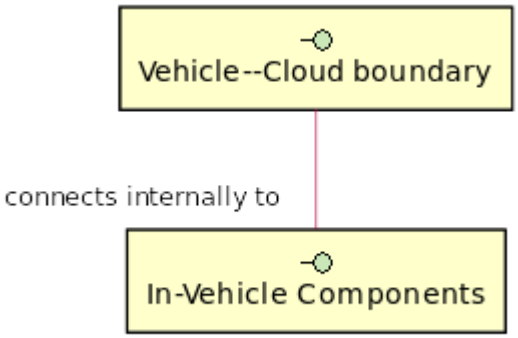
Parts of the technology stack already exists, and the ongoing project(s) will without question identify and provide some parts of it. It's a different thing to actively agree among all CVII participants to agree upon a full-fledged production-quality technology stack in collaboration.

VSS/VSC usage inside and outside vehicle

For maximum benefit make use of the common standards in the vehicle-cloud boundary **and** inside vehicle (as far as possible):



Relationship between internal/external track



This reflection of APIs throughout a system tend to extends through all of the individual components inside the vehicle and in the cloud architecture as well.x

We can immediately see that if the same interface definitions can be used, there is minimal “translation code” that needs to be written in each boundary.

Since everything is connected through a vast technology stack from the point of measurement or calculation to the point of use, we promote “end-to-end” thinking and propose that the common data/interface model *should* be applied as widely as possible in the entire inside and outside vehicle.

Limitations to end-to-end applicability

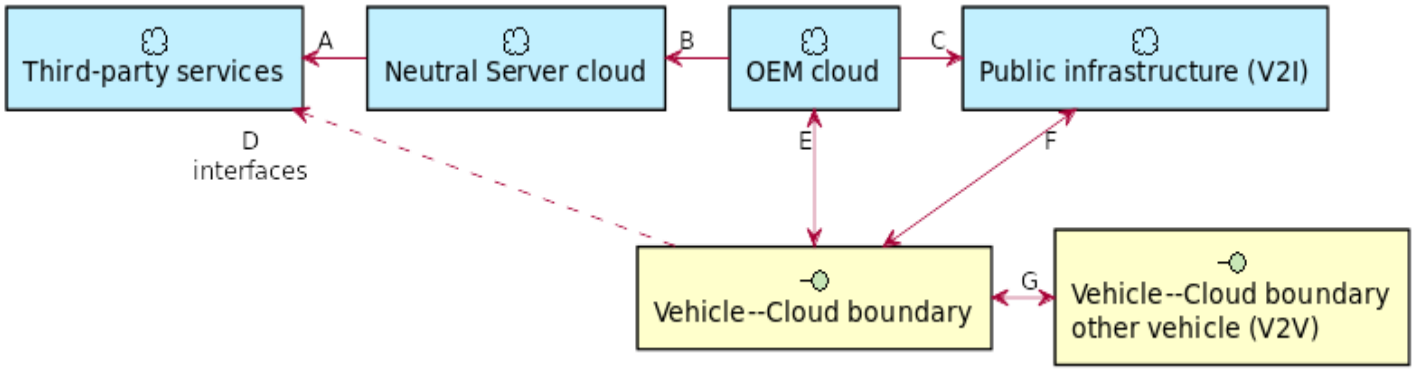
It is of course understood that for various legacy reasons, data and interface standards might not be feasible to apply exactly everywhere. The most common objection that comes up when discussing signal standards is for example that the signal encoding on traditional CAN buses are highly OEM-specific, occasionally secretive, and evolved using very established OEM-internal processes. It is a perfect example that there is likely to be parts of a system that need a translation from a proprietary world to the standardized one, and this is 100% expected.

For example, it is perfectly reasonable to adopt the strategy to translate from the proprietary CAN data encoding to a VSS-model defined encoding, in some ECUs that consume (and in some that produce) CAN signals. It is then possible to implement the rest of the software interaction, inside vehicle and outside, on a well understood common basis.

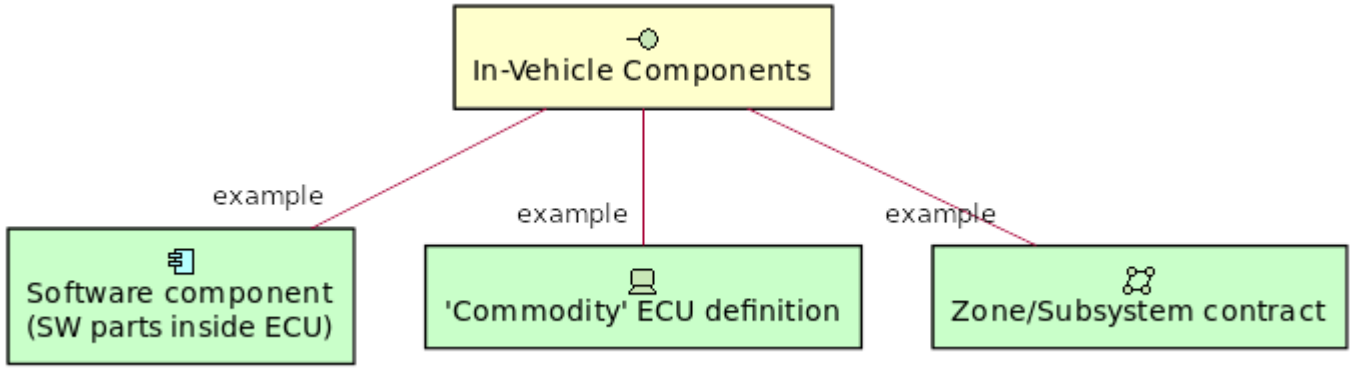
It is however inevitable that translation of data, as well as function calls, happens *somewhere* and should not be seen as an obstacle to the advantages that can still be gained in other areas. It is inevitable at least if some of the promises of standardization are to be realized (either those that relate to more effective development or to connecting to outside defined interfaces such as shared public infrastructure, or to the creation of a developer ecosystem through a common API for multiple car brands, at *some* location in the vehicle/cloud whole architecture). Therefore, there should be a healthy discussion among all CVII participants about where those translations will be necessary (perhaps different per OEM, and that’s fine too), and in which subsystem boundary to focus the API standardization effort.

System decomposition and identified interfaces

Vehicle-to-cloud candidates for data/interface standards



In-vehicle candidates for data/interface standards



As before, we would like to argue that using the *same* data/interfaces and principles for defining them in *all* of the possible system boundary decompositions will yield the maximum value. This is due to that the direct advantages (having a clear standard, having interface description languages and tools) is multiplied in as many places as possible, but likely also significant *compounded synergy effects* from doing the same thing across the system.

With this said, the question still ought to be asked:

Q: Is (standardization of) any particular system boundary more attractive for the CVII participants?

- …for the standard catalog?
- …for the standard model and tools?

(read next chapter which highlights the differences)

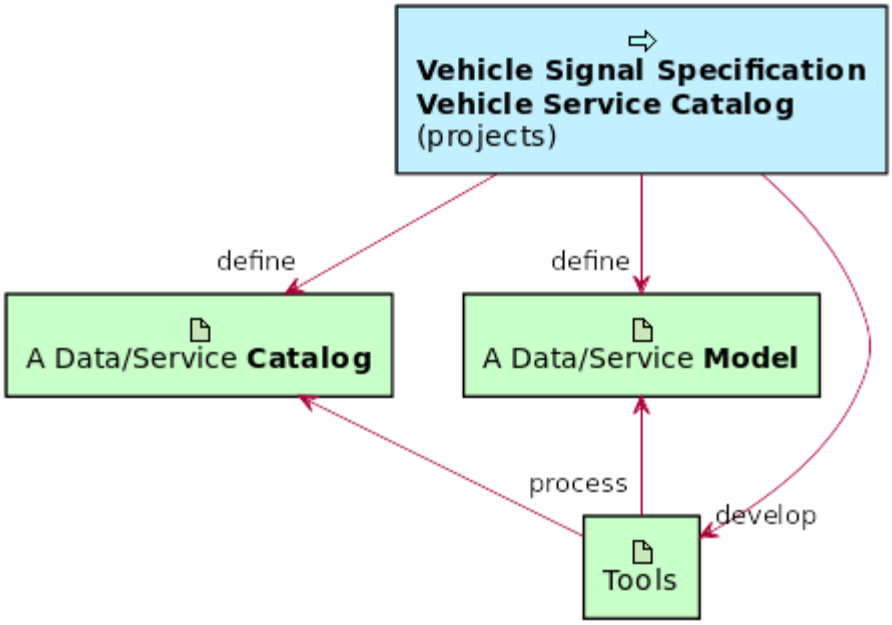
VSS/VSC projects – model vs. catalog

It is very important to note that VSS (and VSC which mirrors the VSS approach) is already delivering two distinct aspects that are complementary and independently useful, although certain effects may require both to be realized:

1. The *underlying data/service* **model**, which is independent from its contents (the catalog). This includes all the modeling rules for how to describe data (/functions) and the intended behavior of all items that participating in the overall model. [1]
2. An industry-wide *standard common* **catalog** of named items. (Data and services/functions items).

- As a complement, common tooling and other technology stack components is tied to the model, and is also able to process *any* items (catalog) regardless of its contents.

VSS and VSC projects deliver 2-3 separate things:



[1] For example the model rules include:

- The allowed fundamental data types (what types of values can be used, such as integers, real values, strings, arrays, etc.)
- General rules for naming items
- General rules for how the hierarchical tree is built
- The concrete syntax of the *definition file* of the data catalog, such that tools can process these files, and humans can write the content in the right format

The second item (standard catalog) is critical to provide a common interface to all vehicle brands. This type of common programming interface is required to build lively third-party development ecosystems since they depend on avoiding fragmentation. It allows applications to be written once but be deployed with maximum reach (compare Android), which is what developers expect.

It is however understood that parts of the industry will want to innovate on unique functions and will want to **add certain proprietary extensions** to the standard. These proprietary extensions have the following expectations:

1. We have an expectation of some private extensions to the standard data/function catalog. [1]
2. It is however highly **discouraged** to have incompatible extensions or changes to the underlying data/function model, beyond that which is *already anticipated and designed for* (such as usage of VSS layers and the principle of deployment models). [2]
3. There *may* be some expectation of extension to the technology stack. [3]

[1] It is unavoidable (and perhaps even desirable from innovation perspective) that some companies will want to make some part of the final product data/function catalog proprietary. While we would encourage putting as much as possible in the common standard, there would be a major risk in not recognizing this fact! The industry might then lose the benefit of the common model if it is too strictly tied to wholesale acceptance of the entire common catalog, or *only* the common catalog. We must therefore recognize and plan for a judicious proprietary extension of the catalogs.

[2] It is **critical** that the model itself is **not fragmented**, but rather kept well defined and consistent across the industry through mutual alignment among stakeholders to define its required features. *Some* extension capability, such as layering, is however already designed into VSS/VSC (to support items 1 and 3).

[3] Technology stack extension may include adding converters to connect to new and yet unknown technology, which happens continuously also in the shared collaboration project. Proprietary (or openly developed, but yet unexpected) extension of the technology stack is likely to be driven by adapting to existing company strategies and methods. These may be legacy methods that already have heavy investment, or it may be aligning to technology used in other verticals (e.g. an existing IoT-focused technology stack).
Beyond adapting to such existing constraints, extensions and incompatible changes to the technology stack are unlikely to be well warranted for the purpose of new and differentiating innovation. It is rather in the proprietary extension to data and function catalogs that such innovation may show itself, because the very basic operations (transfer of data, invocation of functions) we speak of really ought considered a commodity feature that all CVII participants can agree upon, and should therefore be provided to the common project. The development strategy for the technology stack however remains area to be discussed upon among CVII participants (see separate chapter for this)

Differentiation between the value of the Data model and the catalog

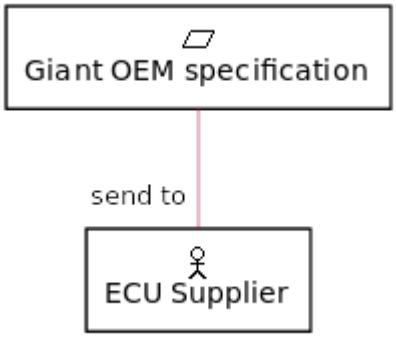
Given the above, we highlight the value in the second item (data/service *model*) independently from the first. The usage of the same model, tools, and methodology for **all** parts of the system, including any proprietary extensions to the data/function catalog, is what creates the great scaling opportunities. When proprietary extensions to the data/function catalog is shared among collaborators (e.g. from Car maker to supplier, possibly even under an NDA), the implementation can *still* use the exact *same methods as used for the standard catalog*. This is in contrast with proprietary and fragmented technology stacks being *introduced under the guise of* it being necessary to support proprietary data and functions. That is completely unnecessary. The common model shall enable innovation and extension, while still avoiding the headaches of conflicting technology for such basic things like data/service definition, the technology stack for data exchange and invocation of functions.

Q: Is the above argumentation making sense to you and what are your comments?

Q: What might be obstacles that you see now and how can they be removed?

Using CVII models to define ECUs

The dominating model of ECU development is something like this:



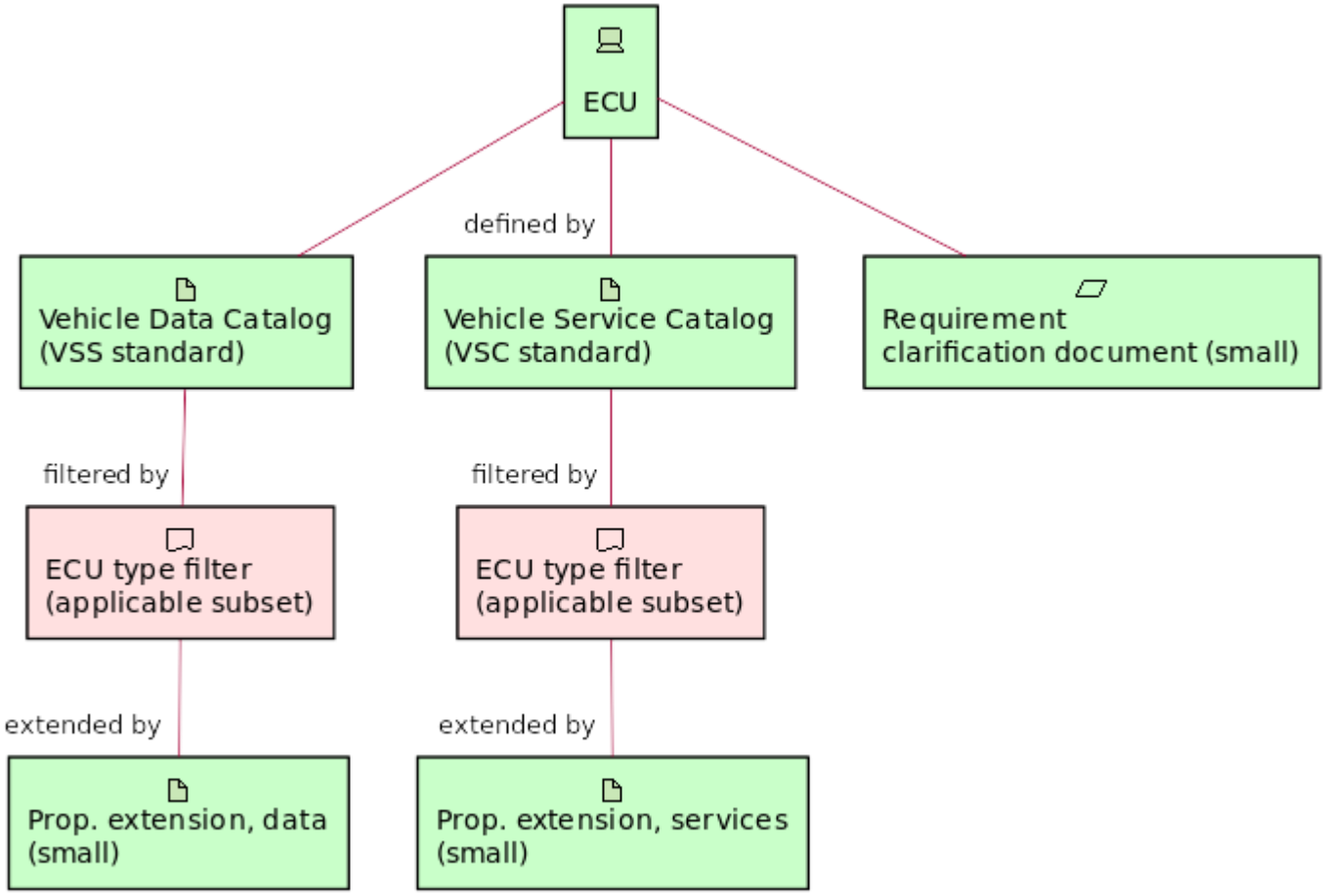
The OEM set of specifications is often very large, complex, incomplete, too much unique to each OEM customer, and not enough based on standards. This leads to all sorts of problems, not the least of which is a long, imprecise, and non-optimized pricing and quotation activity, frequently followed by an arduous, long, development process to bring the system into production.

Alternative approach:

Some OEMs wish to see a more industry-wide, clear, and well defined specification for an ECU. While it might not eliminate the OEM specification completely, here is a model to consider:

1. If *all data* that is to be exchanged is defined in a well understood common language (such as a shared data model, *compare VSS*)
2. If *all functions* an ECU is expected to provide and to consume are described in well understood and common language (such as a shared service/interface model, *compare VSC*)
3. If at least the *majority* of the above data and function definitions are taken from an already *predefined catalog* that all actors (buyer and supplier) in the industry are aware of, with only a small part being proprietary extension to the same…

Then the definition of the ECUs to be acquired could look more like this:



Caveat: Of course this chapter is only speaking about the definition of the software/functionality of the ECU. It is understood that ECUs are also constrained by several other aspects such as physical characteristics, hardware, environmental, production and certification requirements.

The idea shown above leverages the standard VSS/VSC catalogs that have definitions pertaining to the entire vehicle, and it defines this particular ECU *primarily* by applying a kind of requirements filter. In this case, the filter is straight forward – it simply selects the *subset* of the common catalog that applies to this ECU type. Such a filter can easily be described using very similar data formats as the VSS/VSC specifications themselves. In fact, such formats for extension have already been considered and defined in the VSS/VSC projects (e.g. the concept of “VSS layers”)

Necessary extensions (ideally these are few) could then be added on top, and a much smaller OEM requirements/clarification document can accompany the ECU definition. This layered approach allows adding or modifying items to extend the standard catalog.

Finally, any additional requirements that might be needed could be added. This too could be relatively small according to the argumentation in the chapter “What’s in a System”, that the majority function is defined completely by the data and service/function specification.

Standard/commodity ECUs

For *some* ECUs more than others, a definition like the above could become a common industry artifact in itself. This should minimize the proprietary extension and *some* ECUs may become **commodity items**, with a wide and lively market competition with many vendors that can deliver these as essentially a **COTS** (Commercial Off-The-Shelf) item. Acquiring COTS where applicable leaves more opportunity for development of innovative functions in the more innovative areas.

Q: Would the above modeling of ECUs work in your organization?

Q: What might be obstacles that you see now and how can they be removed?

Q: What are your thoughts about defining standard/commodity ECUs?

Using CVII models to define subsystems / zone-contracts and more:

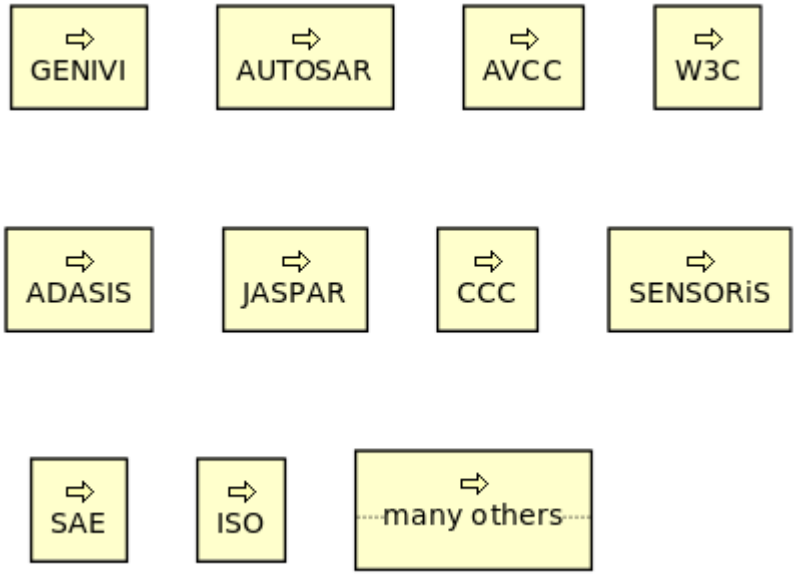
Within OEMs it is common to find company organization mirroring the responsibility of different part of the (electrical) system. There are specifications exchanged for subsystems so that departments know what to deliver, and these are even referred to as *technical contracts.*

Mirroring the above model for ECUs, a similar setup could be made to subdivide the in-car electrical system into subsystems, or currently popular “Zones”. Just like in the ECU example, a common data model combined with a model of available operations can make up the majority of the functional behavior description, and it can be modeled similarly, by applying filters to a common catalog.

Additional modeling of the interconnection of subsystems, individual *testing methods* of the same, and many other development aspects can be derived from this fundamental model. All such development aspects benefit from these formal data/function models.

For all involved (departments, suppliers, and even other car companies in so far as there are some joint collaboration projects), speaking an already familiar known *common language* facilitates the process tremendously.

Organization among different consortia



To be discussed…

It is likely that answering the other questions first should clarify the essence of the required work and then eventually be able to relate that goal to these other projects.

Q: In which consortia is your company involved?

Q: How do you envision organizing the work within these consortia so that the goals of the common initiative are met, rather than producing conflicting and fragmented technologies?