Rethink EE Architecture in Automotive to facilitate Automation, Connectivity, and Electro Mobility

Anders Magnusson
Volvo Group Truck Technology
40508 Gothenburg
Sweden
anders.magnusson.3@volvo.com

Leo Laine
Volvo Group Truck Technology
40508 Gothenburg
Sweden
leo.laine@volvo.com

Johan Lindberg
Volvo Group Truck Technology
40508 Gothenburg
Sweden
johan.jl.lindberg@volvo.com

ABSTRACT

Nowadays software and electronics play a fundamental role for commercial vehicles in order for a driver to manually operate them effectively and safely in different transport applications. Although the overall design thinking in the commercial vehicle industry is still very much oriented towards a geometric perspective and thus physical modules, which for software means binaries related to physical electronic boxes. Furthermore, there are many incentives for a higher degree of automation for commercial vehicles to gain productivity, while at the same time facing very different demands on final transport applications. In addition, the environmental impact drives the need to reduce the fossil fuel usage by introducing electrified propulsion torque, which could be distributed over several vehicle units. In order to manage this variety of final applications a product line oriented approach is used that will also be challenged by the need to support a feature range from manual to fully automated vehicles and alternative powertrains, possibly distributed torque supply and electrification of many things. In order to deal with different transport applications; wide feature range; and a transition from traditionally closed embedded systems towards interconnected machines and systems there is a need to shift the traditional ECU-oriented mindset. In this paper a supplementary perspective is added to the traditional geometry-oriented perspective – a functionality perspective, which facilitates reasoning about functionality and thus application software. The paper proposes a reference architecture that is based on horizontal and vertical layering of functionality.

KEYWORDS

automotive, reference architecture, application software, automation, electro mobility, platforms, product lines, horizontal layering, vertical layering

ICSE-SEIP '18, May 27-June 3, 2018, Gothenburg, Sweden © 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5659-6/18/05…$15.00
https://doi.org/10.1145/3183519.3183526

1 INTRODUCTION

One of the cornerstones in the automotive industry is to achieve large scale reuse of manufactured and assembled entities in order to provide the market with mass-produced cost efficient products where resulting elements are formed in a "product platform". As a result of this the modularization focus is very much on a geometric installation perspective and geometric modules as shown in Figure 1.

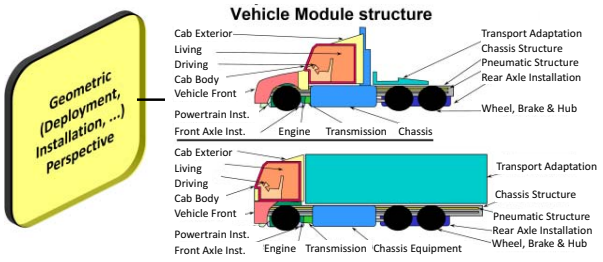

Figure 1. The main perspective and mindset in automotive is geometry/deployment oriented dealing with large installable monolithic binaries in the reasoning about software.

In the Volvo Group this is known as the Vehicle Module Structure (VMS) highlighting generic (geometric) vehicle modules. Transferring this perspective into the software landscape; this is similar to a deployment viewpoint [1] – where things are geometrically deployed.
As a result the general software focus at Volvo Group and in the automotive in general is on modules of executables, binaries etc. that are flashed into the memory of the electronics enclosed by a physical box – known as an Electronic Control Unit (ECU). Not long time ago the ECU and the binary were not even treated as individual entities but as single part numbers, i.e. you could not replace the binary without changing the hardware. These ECUs are combined in an in-vehicle network topology as shows in

Figure 2. Most engineers working in the automotive would probably state that this IS THE architecture.

The primary driver for this focus has been and still is the bill of materials (BOM) and thus lower the material cost. A lot of development money and resources are therefore invested to lower the footprint of the software in the memory to save some cents on the memory size. On the contrary, this BOM optimization focus tends to drive complexity of the software, which now drives excessive increase in software development cost.

1.1 Software in automation, electro-mobility and connectivity

Cars and trucks have continuously evolved with software enabled features in a relatively moderate pace over the last three decades. Software is today a major enabler for improving old features as well as providing new features in the automotive industry many of these features are not directly linked to this geometric modularization. It is not only high end vehicles but also low end vehicles that have a quite impressive amount of software in order to manage things like a vehicle's perimeter, seat adjustments, acceleration, braking, etc. It has been estimated that more than 80 percent of new vehicle innovations are enabled through software [18] and by adding sensors. The trend in the Volvo Group is that the pace of innovations into customer accessible products is accelerating. Not even old software entities such as a Cruise Speed Controller are linked to geometric modules like a combustion engine and its associated EMS ECU which today host the Cruise Speed Controller according to SAE J1939 (there is an implicit deployment built into this standard).

These large executable monolith entities or ECUs as in Figure 2 are not really supporting an overall reasoning about features and application software specifically. It worked some years back when the ECUs and their binaries were purpose made for a particular "function" such the engine ECU controlling injection and timing, the gearbox ECU controlling clutching and gear shifting mechanisms, and the brake ECU controlling electronically the brake request and anti-lock braking or a trip computer. It will also be the application software that drives the need for powerful electronics, i.e. flexible and reconfigurable computers. [22] The way of working with commercial vehicles is far from adapted to looking upon them as software intensive products or service providers, which vehicle automation, connectivity and also electro mobility is about.

As in enterprises, software is a major contributor to automation – replacement of human performed activities. This of course also goes for various levels of vehicle automation, where vehicle automation is synonymously with huge amount of application software. As commercial vehicles are used in B2B operations, the interest in automation is perhaps of higher incentive for trucks than for cars as it contributes to operational margins. Roughly 1/3 of operation cost is related to having a human behind a steering wheel. Only comparable, B2B operation in cars would be taxi business and small package deliveries. The automation will also go hand in hand with an increased level of connectivity in order to operate logistics of unmanned vehicles, maintenance and in some case remotely drive a malfunctioning vehicle to get it into the roadside.

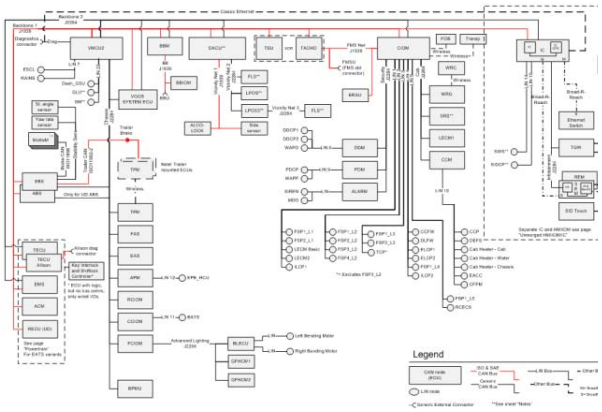

Figure 2. Network topology is what most look upon as THE architecture in the automotive.

Furthermore, electrification of powertrains is a necessary part to comply with zero emission zones as well as to reduce fossil usage. High efficient engines and alternative fuels are also measures for CO2 reduction. But also other kinds of electrification of various auxiliary loads such as oil pumps, air fans will happen. From a software perspective these technologies drive the need to support several different and complex powertrain topologies compared to the traditionally used combustion engine powertrain and thus such as to enable monitor and control power consumption. To handle this complexity with feature and energy operation the software must prevent that different powertrain topologies alters the complete set of software but only the powertrain related parts. From an overall energy management perspective, usage of different energy sources and buffers requires monitoring as well as prediction to minimize the energy consumption for a certain transport mission. This is related to the overall vehicle mission as a whole rather than to a particular "mechanic device" device and thus such functionality should not be part of the device.

What we more and more experience in hour truck development in the Volvo Group is that for the ever increasing amount of software, the traditional geometry and deployment perspective is not feasible any longer. This continuous increase of software requires that application software is carefully designed with respect to extensibility in order to support forthcoming truck features and to support large variances in the different transport applications (variability) where trucks are used. However, designing for extensibility and variability is almost in the opposite direction compared to the current mindset which is working towards software optimization for the electronics which makes it hard to reallocate and in some cases to port software to new electronics.

1.2 Several layers of suppliers primarily focusing on delivering ECUs

With its roots in the mechanical world, the automotive industry has a long tradition of making use of many different suppliers of the diverse set of manufactured parts used in the assembly line to produce a vehicle. Some of these suppliers are even larger than OEMs and they make use of other suppliers – a whole chain of suppliers. This of course also goes for electronics and software, which has resulted in that many suppliers' focus, is on delivering an ECU with a binary as installation entities, which is a result from binding software functionality to dedicated electronic hardware early. However, continuing on the path with a new ECU for each new feature is not feasible any longer because ECUs cannot be geometrically distributed or packaged as it is commonly known as in automotive. There is simply no space left although trucks are big the space is preferably used for carrying payload instead of ECUs and wires. Many geometrically distributed ECUs will also result in many wires as well as increased communication over relatively slow CAN and LIN busses, which are today's dominating network communication paradigms in automotive.
In contrast to the car industry the truck industry has cemented this way of looking at software through the SAE 1939 CAN industry standardization where the information elements, known as signals, are standardized to/from certain ECUs and thus signals are packaged into frames. The intention of this recommended practice is to allow electronic devices to communicate with each other by providing a standard architecture [12]. An implication of this particular standardization is that it prescribes at which ECU a certain piece of software runs. This is completely in the opposite direction compared to other application domains that have been and still are heading towards a Service Oriented Architecture (SOA) *architectural style* in [17]. In SOA, the hardware and run-time platforms, at which the software is running, are almost neglected. The focus is more on services which are units of software with distinctive encapsulated functionality – Lego Bricks - that run in a network of computers.
Since 2006 the AUTOSAR [6], which is a component-based framework focusing on an a common execution environment, has slightly changed the traditional ECU-oriented focus within the Volvo Group and many other automotive OEMs by putting non-allocated software components in the forefront- Although we are still purchasing electronics and software running on that ECU. With this mindset the focus turns into logical but static interaction between software components, which does not really go hand in hand with the tradition of SAE J1939 as signals are not predefined to come from a certain source and packaged into a certain frame. Also, AUTOSAR has been leveraged as a key mechanism to achieve software reuse, scalability to different vehicle and platform variants and transferability. A component-based approach requires existence of components which can be integrated. However the experiences from the recent developed Truck Electronic Platform within the Volvo Group is that there are no AUTOSAR-based off-the-shelf software components available on the market that can be compiled and linked together into a binary with either in-house developed or other 3rd party software components. The authors do not see any signs for that to come in near time.

1.3 Product lines supporting multiple product families and brands

In order to provide a mass market with a large number of products with freedom to enable selections among various available features to "customize" the products to each brand's personal interest or as for trucks for their final transport application, all vehicle OEMs have for many years worked with product platforms. This is done to scale in economy such as reuse of development effort, reuse of manufacturing capabilities, warehouse handling of spare parts, etc. In the software community this is known as product lines [11].
In the commercial truck field trucks are frequently divided into categories based on their mechanical characteristics such as heavy and medium duty trucks, but also according to long haul distribution, construction, and city distribution. Sometimes also buses are included into this context as in Figure 3. The Volvo Group also includes construction equipment vehicles which also are working towards a similar architecture. On top of this many OEMs have put several brands with their market/product niche characteristics under the same umbrella, e.g. the Volvo Group has the following truck brands in their product/brand portfolio: Volvo, Renault, Mack, Eicher, UD and a joint venture with Dongfeng and the Volkswagen group has MAN and Scania. When it comes to features heavily supported by software and electronics many of these are common among the product families and brands. Hence the Volvo Group is looking upon software and electronics as a separate product line with variability built in to achieve for example uniqueness between brands as in Figure 3.

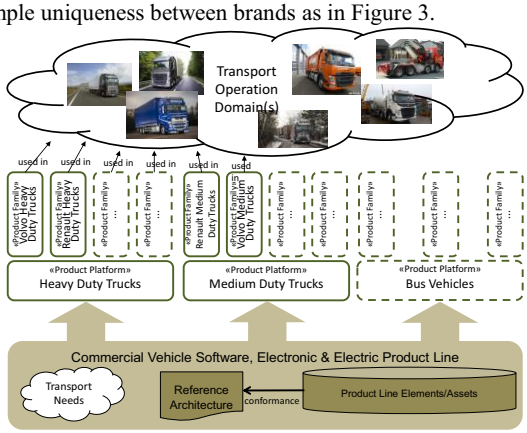

Figure 3. Product lines supporting multiple product families and brands within the Volvo Group.

1.4 Hard to achieve large scale reuse without a shared architecture

It has been recognized that an architecture has major impact on the easiness to cooperate among teams. In 1997 it became clear that Microsoft divide projects in a way that mirrors the structure of its products, which helps teams, create products with logical and efficient design and with efficient groupings of people [9]. In

this case all development is inside the same enterprise, which is of today not the case for commercial vehicles. There is no need to debate that development of software functionality for commercial vehicles would be vastly different. In order to achieve large scale reuse of application software entities rather than the binary monoliths, a shared architecture thinking is a necessity, not only within a company, but also among OEMs and different tiers of suppliers. But, it has also been recognized that the design of any system is significantly affected by the communications structure of the organization that develops it, i.e. any organization that designs a system will produce a design whose structure is a copy of the organization's communication structure [10] such internal line organization structure, structure of agile release trains, and various tiers of suppliers.

The situation right now for commercial vehicles is that *there is no such common architecture view*, especially not to manage cost efficient solutions where the products offered by OEMs like Volvo Group ranges from manually to fully automated operation of trucks together with the energy operation variability. The AUTOSAR component framework does not help either.

Industry Challenge: Establishment of a shared architecture for the commercial automotive application software domain in order to reach large scale reuse of software implemented services both within an OEM and between OEM and suppliers in order to facilitate Automation, Connectivity, and Electro mobility.

In this paper we will present the results from the internal R&D conducted within the Volvo Group on vehicle automation and electro mobility and the architecture that guides us in our software development and by that contribute to the industry challenge. We perceive that the situation in the Volvo Group is similar for many OEMs and for most 1st and 2nd tier suppliers:

- a necessity to support a portfolio of truck and brand families
- a necessity to support for various feature levels for these truck and brand families
- the transition from manually operated trucks to fully automated trucks will not happen overnight, it will for a quite long period be so that application software must be designed in a way to support this evolution of transport automation which can include automated or manual driving of the vehicle system

The way of looking at architecture in this paper is based on the definition given by [15]: *fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution*, where the system is a product line and the environment is commercial vehicles, mainly trucks. The paper only covers a fraction of the fundamental concepts of the reference architecture we are targeting towards. The conference paper format is too small for that.

2 Apply a Functionality Viewpoint

To be able to achieve large scale software reuse it is a necessity to apply a product line engineering approach. And to provide solutions for a large range of final applications of the trucks there is a need to build in many and different kinds of variation points, i.e. reconfigurability and tailored vehicle features, in such a product line. To manage the transition that software is mainly about mechanical component control, e.g. engine, transmission, braking, to actual vehicle feature control, it is a must to not just talk about the physical modules but rather more abstract entities – some kind of entities of functionality. There is a need for a similar picture as the network topology in Figure 2 but for application software.

Within the Volvo Group we do that by adding an additional perspective on the vehicles – here defined as the functionality viewpoint, as in Figure 4, and not function viewpoint in order to avoid the mental mapping of y=f(x) as structural entities. The need for such a viewpoint is also identified by [5] as a link between overall customer demand and physical structure. This will make a shift to focus on reasoning about and reuse of modules of functionality rather than "physical" modules – in this sense a product line is a set of modules of functionality shared across multiple end-user products [13]. The intention is that we look upon structural entities showing up in this perspective as "products" in a similar way as structural entities in a geometric viewpoint are treated as "products" with part numbers etc. Currently our product life cycle management applications are not supporting this as they are primarily designed to assemble physical things in production.

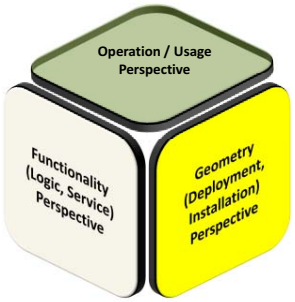

Figure 4. Multiple viewpoints to deal with separated and unrelated concerns.

Furthermore, in order to deal with higher and higher demands on dependable vehicle operation, driven by more advanced features but also by the functional safety standard ISO 26262 [16], there is a necessity to more clearly structure and through that separate different concerns such as different levels of criticality of functionality from each other in order to guarantee that the lower criticality elements cannot interfere with the functioning of the higher criticality elements. The ISO 26262 explicitly acknowledges a functionality perspective through its focus on functional (safety) concepts in the standard.

Another perspective that has been added in this transition is that of an operation / usage perspective (Figure 4), where of course Use Cases or End User Functions are excellent mechanisms to cover and describe operational issues, needs etc. For example, *steer* or rather *keep a path*, *control speed* (brake + propulsion), *illuminate road ahead*, etc. are just some customer activities commonly known as functions, y=f(x), in automotive enabled by the architecture and these entities are used to reason about the operation/usage perspective. However, this perspective focuses on the transport operation domain and really not on the solution domain as the functionality and deployment (geometric) perspectives do and is not part of the architecture thinking presented in this paper. Also, it is hard to take these end user functions one by one and define an architecture for each function but they have to be looked upon several at the same time.
In addressing the industrial challenge, the question then comes to: *How to think when looking upon the commercial vehicles from a functionality perspective?*

3 Layered Organization and Distribution of Functionality

In principle one has to take a full vehicle perspective on this and also include functionality handled through mechanics, pneumatics, electronics and not just software. Based on the thinking of separation of concern the architecture approach is based on that functionality dealing with monitoring and control things is separated from functionality handled through electricity, diesel, mechanics, pneumatics, hydraulics, etc. as in Figure 5, where the latter things that are to be monitored and controlled. Also, the low layer in Figure 5 can be expanded into "systems" but that is not in the scope of this paper. As can be seen in Figure 5, there are dependencies from the Truck Monitoring & Control System to various "systems", which will mainly be handled by the device abstraction layer introduced in Figure 8.

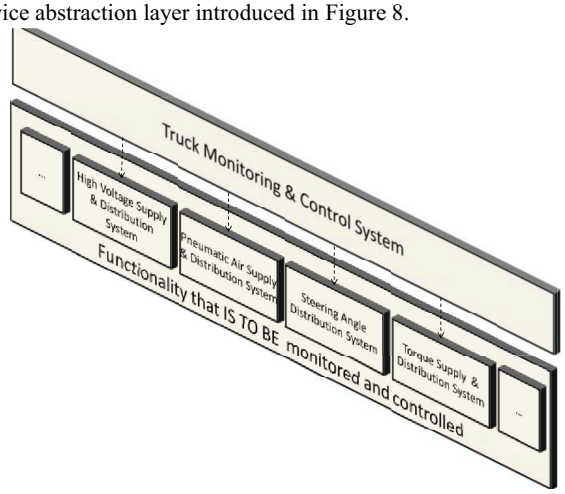

Figure 5. Separate the things that are to be monitored and controlled from the one actually monitoring and controlling.

Furthermore, as the nature of the functionality dealt with in the Truck Monitoring & Control System is very different, ranging from converting an analogue value to a digital value, forwarding this digital value from a converter circuit to an application software entity where it might become a temperature, leads to that this is structured into three major product line entities as in Figure 6 which is a kind of layered architecture style [7]. As these are separated from each other the reasoning and focus in these also varies a lot.
In the Computation & Signal Distribution the focus is on electronic functionality interfacing various actuators and sensors that are part of the device functionality such as interfacing a Fan Motor, an Inlet Air Pressure Sensor, Brake Pad Wear Sensor etc. It also focuses on processing and memory capacity as well as network structures its performance. It is worth to mention that much of its necessary content depends on the content and structures of the Truck Application System, although part of the history its content and structure has been defined in advance of understanding the content of Truck Application System and hence very often soon run out of resources. In the Truck Application System the focus is on managing information entities and units that are relevant to talk about in the application domain such as the *inlet air* and its temperature, humidity and mass flow (speed) or a *climate comfort* service. The world in-between these is some kind of middleware functionality that bridges these two worlds, where AUTOSAR [6] is just one such middleware framework. The three dimensional way of describing Figure 6 is made intentionally, which will be revisited later in the paper.

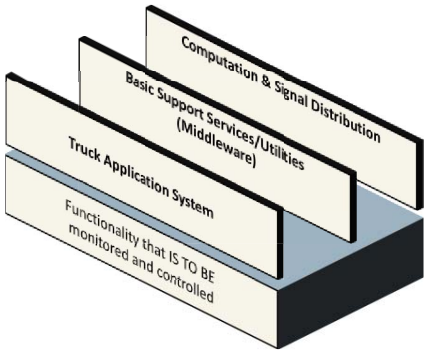

Figure 6. The Truck Monitoring and Control System is divided into three major "subsystems".

Except for the nature of the functionality another really important reasoning for this separation is that cycle-time for developing functionality in the Computation & Signal Distribution is very different from the one in the Truck Application System. Furthermore, as the hype around agile/lean based development process frameworks like SAFe [22] also has reached the automotive domain with hope for more software-based features both faster and more continuously developed, integrated and

distributed to customers, there is really a necessity to even more strongly make this separation happen at the top level in a product line.

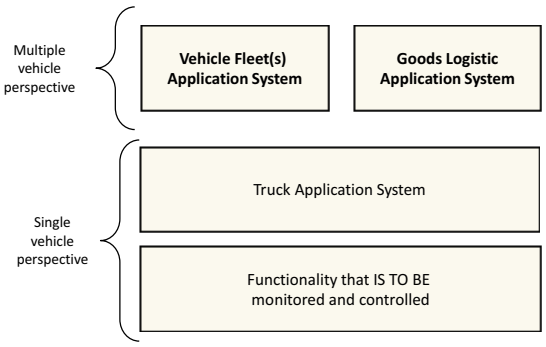

Figure 7 Extend the thinking manage multiple vehicles rather than single vehicles

In the defined approach it is within the Truck Application System the structure of various application software entities will reside that have different pace compared to low level functionality in Computation & Signal Distribution and Basic Support Services/Utilities. The separation into Computation & Signal Distribution and Truck Application System also simplifies the development of technical safety requirements "allocated" either to software or hardware, part 4 in [16].
Another thing worth to highlight here is that Truck Application System focus on single vehicles. However, to gather statistics of wear or energy consumption trends of for example all Volvo, Renault and Mack trucks - a fleet of trucks – there is a need to add another layer on top of Truck Application System as in Figure 7. These are not in the scope of this paper.

3.1 Layer application functionality horizontally

By work and experiments conducted during 2009-2016 it has been concluded that there is a need for a slightly different layering than in [4], among other things the architecture presented here has added clear separation of HMI and the ego vehicle's environment and operational functionality has been divided into two layers. The reference architecture presented in this paper proposes five horizontal layers as in Figure 8 where two layers address operational functionality; two layers deals with tactic functionality; and one layer deals with strategic functionality but also two vertical layers are added.

All with the perspective of a single vehicle, but the vehicle may at the vehicle utility layer be looked upon as two track model but just as a particle in the route situation layer! Each layer raises the abstraction level from the "physical" functionality that is to be monitored and controlled. The architecture approach defined here combines a strict hierarchical style as in [3] with a heterarchical style [14] where all modules communicate with each other – it becomes a layered style. The idea of a layered style is to deal with the disadvantage of a strict hierarchy as it introduces inflexibility and long response chains but also the problems associated with a strict heterarchical style as it introduces many problematic couplings all over and lowers the possibilities to reuse and by that achieve a scalable product line of application software.

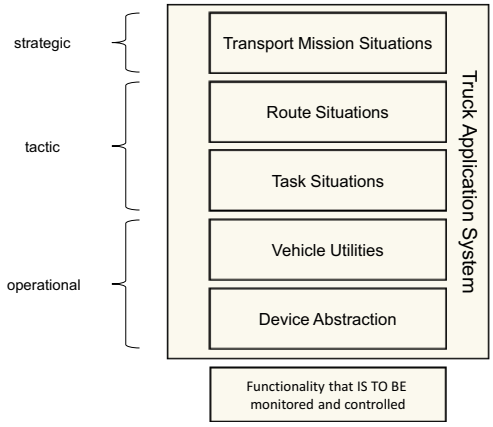

Figure 8. Five horizontal layers of the application software.

A similar layered architecture for platooning feature of commercial heavy vehicles has been presented in [19], which also contains strategic, tactical and operational layers. There are differences in names and it can be seen that the focus has been solely on platooning. In the presented approach here, the platooning service is managed in Route Situations and when vehicles joins the lead vehicle the adaptive cruise control is changed to platoon cruise control within the task situations layer. Now the follow vehicle is just a slave and listens directly to the requests of lead truck. However, in [19] it is not revealed how full automation is going to be approached nor how transition of transport automation which can include manual and automated driving of the vehicle system.

3.2 Device Abstraction Layer

The overall thinking in this layer is that application software entities shall be representations of mechatronic devices such as a *Fuel Tank*, *Wheel*, *Clutch*, *Wheel Brake*, *Windshield Wiper*, etc. to *localize the knowledge about the characteristics of a particular device* as shown in Figure 9.
This will facilitate changes in the devices without having that rippled all over the product line. Device abstraction entities are carriers of information elements that represent various properties of an element that is to be monitored or controlled. E.g. *current tire pressure*, *current wheel speed*, and *current tire temperature*, and *nominal tire dimension* are organized into a software entity representing the *wheel*. And data such as *current wiper position*, *current wiper speed*, *wiper operation*, etc. is data that together forms a software entity representing a *windshield wiper*. The important here is that this is to be looked upon as service provider

and does not necessarily run on its own ECU as would have the traditional way in automotive. We are not there yet, but having such entities looked upon as individual deployable entities would ease a continuous evolution of our product line as well as ease upgrades in the fields.

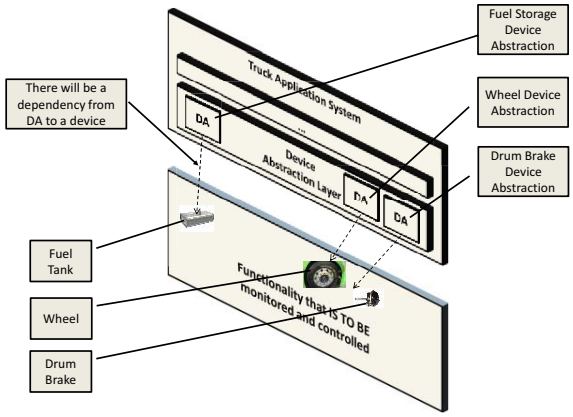

Figure 9. Device Abstractions are supposed to be representations of od modules of the functionality that is either monitored or actuated.

As the devices the device abstraction will be abstractions of, this layer will also hold many application software entities representing electro-mechanical sensors and actuators such as strain gauge for measuring the temperature in a friction brake or a strain gauge measuring the tire pressure in a wheel. In the AUTOSAR context this is known as a sensor/actuator component and these are having tighter connection to the electronic I/O than the previous mentioned device abstractions and also less changes in an operational vehicle. These are candidates for more I/O-oriented computers.

The idea with these device abstractions is that they hold statically configured information such as wheel size that part of the wheel device abstraction and nominal tank volume that is part of the fuel tank device abstraction. Instead of having such configuration information spread over many places, e.g. a GUI element that shows current volume for a driver, shall ask for the nominal tank volume and another entity dealing with trip information such as distance to empty also asks for the same nominal tank volume instead of having that parameterized at the usage places. This will slight increase bandwidth when distributing these entities but that is outweighing by the application software components being more self-contained.

Furthermore, this thinking is also related to that there might be different variants of these devices such as Disc Brake and a Drum Brake and as these share many properties it is good if they are defined in a single point e.g. as a Wheel Brake as in Figure 10. Unfortunately many design tools popular together with AUTOSAR such as Matlab/Simulink and DaVinci Developer do not support this kind of design. Instead it would be beneficial to apply the object-oriented inheritance mechanism for this. Also, in this layer there will be many software entities that deal with the characteristics of actuators and sensors directly linked to these devices.

A key principle in this layer is that these Device Abstractions shall not have direct interaction with other Device Abstractions. Instead, coordination between device abstractions is made at a higher layer so in this case the architecture goes for a hierarchic style.

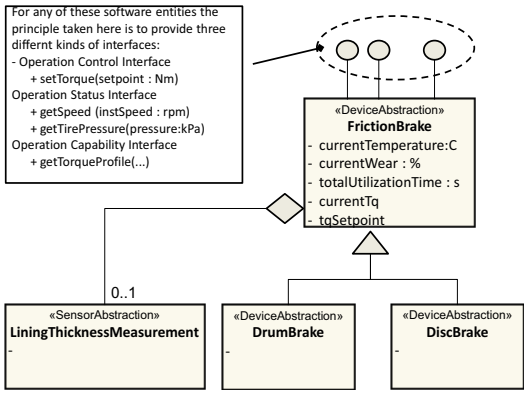

Figure 10. An example of a Device Abstraction with two variants and three different types of interfaces are provided.

3.3 Vehicle Utilities Layer

The purpose of this layer is first of all to *raise the abstraction level from individual device abstractions to some kind of modularization of useful application functionality into vehicle utilities*. A Vehicle Utility is defined as some kind of useful vehicle wide service that enables a user to perform one or many of its activities via an HMI in a manually operated truck. Hence a Vehicle Utility represents a goal experienced by the consumer. So for example a human driver performing the driving task, the task is enabled by utilizing a Moving Unit Motion, a Window Cleaning and a Climate Comfort utility as in Figure 11.

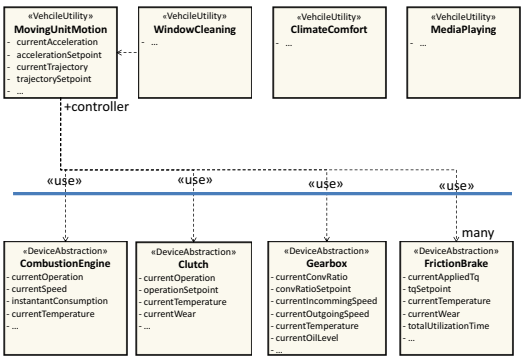

Figure 11. An example of a Vehicle Utility module that is acting as controller and encapsulate coordination between different device abstractions.

When a Vehicle Utility is operational it is utilizing lower level device abstraction functionality and hence hides mechanisms (devices) to achieve its desired goal. Looking a little bit more specifically into this layer one can see that vehicle utilities are quite different from each other and some are closer and more coupled to each other. Therefore this layer will in practice consist of a number of domain packages organizing a set of vehicle utilizes as seen in Figure 12, where the key domain for providing the services related to a vehicle's six degrees of motions, such as longitudinal and lateral motion, doing motion estimations and device coordination is within Vehicle Motion & Power Utilities. It is these domains that facilitate our organization to talk about the application software as the network topology and its ECUs has supported to automotive organization in the past.

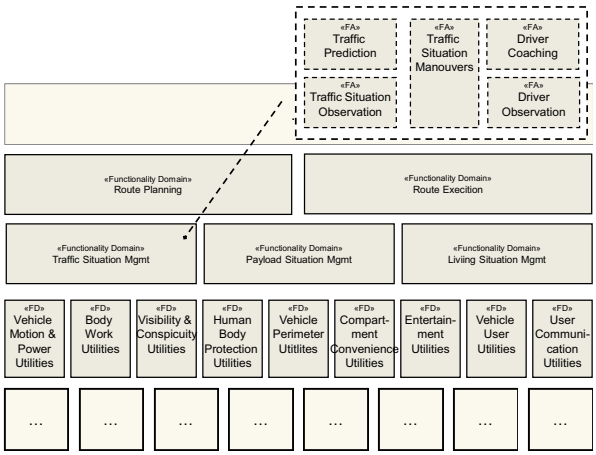

Figure 12 In practice, a layer will be replaced with a set of application domains.

3.4 Task Situation Layer

When a human user performs a use case such as "transport payload from A to B" or "load/unload payload" a use case is actually the performance of a coordinated set of tasks in a given situation. In a situation where a physical user performs a task (a non-automated situation), the human user is acting as a controller. The purpose of the task situation layer is to enable introduction of more and more automation of these tasks and coordination of tasks traditionally performed by drivers and other local operators. In order to achieve this, it "consumes" various vehicle level utilities. So *basic entities that show up inside this layer are entities that represent tasks a user performs (today but not tomorrow)*. It is not foreseen that these tasks will disappear, but they are automated and hence they shall also be represented as such in the application software. For example, in the case of a driver in the control loop, the driver is actually acting as a *speed controller*, a *trajectory controller* (steering), *headway illumination controller*, *forward sight controller*, etc. and therefore such tasks shall also be represented as application software entities. Further breakdown of this layer into domains can be made such as *traffic situation mgmt*. and *payload situation mgmt.* as in Figure 12. In [20] and [21] one of these domains is broken down into even smaller functionality areas (FA) such as *traffic observation mgmt.*, *traffic prediction mgmt*., *maneuver mgmt*., *driver coaching mgmt*., and *driver observation mgmt*.

3.6 Transport Mission Situation Layer

This layer raises the abstraction one level further, and instead of reasoning about it as a "physical" truck or user activities, it instead focuses on what kind of different transport missions that can be offered (still in the single vehicle perspective) such as: How many containers and sizes are expected to be transported and assigned by a vehicle or gravel transport assignments, soil transport assignments, sand transport assignment, or frozen food transport assignment. This is about dealing with strategic decisions such as "optimize" for operation cost or delivery speed among assignments.

3.7 Ego vehicle's environment situation

Another kind of functionality that is different compared to the concerns addressed by the previously defined horizontal layers is the functionality that deals with creating a picture of an ego vehicle's environment situation. In this architecture this is orthogonal to the knowledge in the horizontal layers as seen in Figure 13.

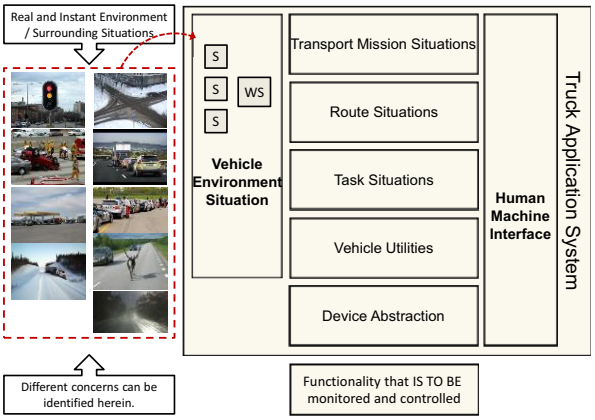

Figure 13. Vertical layers dealing with creating an instantaneous picture of an ego-vehicle environmental situation and services that let a human to interact with all functionality.

For example, the functionality that represents *weather*, *traffic jam* or *road characteristics situations* are located here; this software entity can make use of other software entities such as detections from a camera, radar or temperature sensor as well as communication with another vehicle to get a proper picture of the weather situation in the ego vehicle's environment. This implies that there is an internal layering herein that distinguishes between

raw measurements of the environment coming from sensor functionality and a more fused picture of the situation based on several sources of information.
If one wants to keep track of and monitor for example the environmental situations for all Volvo, Mack and Renault trucks - to build up some kind of larger situation picture of their surroundings, i.e. applying the thinking in Figure 7, there are possibilities to collect and manage for example a *weather situation* from several trucks to build up a *world weather situation* or *road friction/characteristics situation* as in Figure 14. It can also be so that this world situation pushes situations from one vehicle to another if they are close.

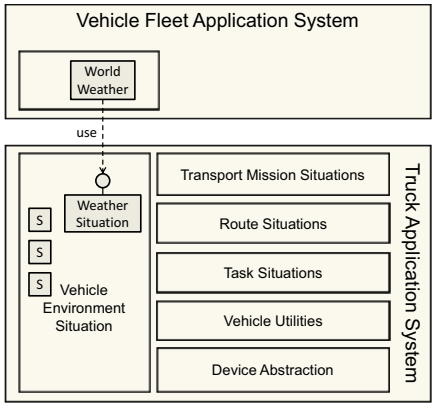

Figure 14. Extend the single vehicle perspective of its environment to a fleet of vehicles.

3.8 Think Model-View-Controller to deal with human interaction

Although automation is a hot topic for commercial vehicles the transition will take time and a human will need to interact with various kinds of functionality, especially to support a product line, that enable trucks within the range from manual to full automation. Thus it is natural to think of a commercial vehicle as a human interactive intensive application. The approach taken here is to clearly separate Human Machine Interface (HMI) as in Figure 13 from the functionality that represents the core knowledge as in the model element in the Model-View-Controller architecture pattern [8]. The model elements represent core (vehicle) knowledge, where a model element could be a single object or it could be some structure of objects, and these model elements belong to the various horizontal layers in Figure 8.
As the Volvo Group is dealing with multiple brands in its product portfolio this separation enables HMIs to look very different between the brands while maintaining stable core knowledge. This separation enables deployment of HMI functionality to thin GUI clients in dedicated HMI ECUs or smart phones having core elements in more generic and common ECUs.
Furthermore, it opens up for a possibility to localize what kind of control a user can do and when that can be done. It also addresses the issues of managing multiple users to control from multiple places simultaneously and not having that issue rippled down into the core knowledge, e.g. having direct access buttons mounted e.g. in a door panel and in a handheld device as well as via soft buttons visible in an thin GUI app running on a smart phone. In order to deal with this a generic architecture component known as a User Input Controller (UIC) has been defined which is responsible for WHAT control is offered to a human user and WHEN that control is possible. All these UICs belong to the vertical layer Human Machine Interface and work as a link between a user and the core functionality.
The architecture also has a similar User Output Controller that provides feedbacks valid for a human user. This entity works as an abstraction layer between view entities and core elements and can organize feedback different than how the core elements as structured.

3.8 Think Object-/Component-/Service-Oriented instead of Function-Oriented

It is important to understand that any kind of software system such as a gearbox control system, navigation system, telecom base station or an order management system all deal with a massive amount of information and most of the times in real-time. They are just different categories of information systems. Having this in mind the ownership of information that represents various kinds of states such as a *speed limit*, *max vehicle speed*, *current acceleration*, *current curvature*, *instantaneous fuel consumption*, *average power consumption*, etc. is essential. Also, *current vehicle acceleration* is completely different than *instantaneous fuel consumption per time unit* which is an issue for a combustion engine device abstraction and *instantaneous fuel consumption per travelling unit* (km or m) is different and the scope of a vehicle utility. Therefore the layered architecture is supplemented with the mindset of an object/component-oriented architecture style, as already highlighted in Figure 9 and Figure 10, at the lowest level in software modularization within the layers. These objects are acting as service providers as well as service consumers and the will also be possible deployable entities in a SOA-environment. This will in turn facilitate a scalable product line but also ease agile self-going development teams having smaller deployable entities than deploying a big binary monolith as of today.
One fundamental principle in the architecture is that of letting each software element provide three different kinds of interfaces, as visualized in Figure 10: operation control, operation capability (instantaneous capabilities), and operation status. We look upon them as different kinds of service provision interfaces.
Achieving a directed dependency, e.g. as depicted in Figure 11, requires a careful design of the interfaces at the end of the dependency arrow. That is, the dependency is realized by Operation Interfaces of one or more public Service Objects residing in these domains of functionality. The design of this Operation Interface must be done with care otherwise reuse of functionality can get lost as well as changes might ripple all over. Thus the Operation Interface shall be structured into three different kinds of (application) interfaces Operation Control, Operation Status and Operation Capability

This is very well supported by a component-based approach although current version of the component-based AUTOSAR framework does not have ownership of information at its heart. This is a quite big difference from current design paradigms in automotive which often favor the function-oriented paradigm focusing on algorithmic decomposition where data is flowing around among the functions as parameter passing or as global data.

There is a proposal to standardize some of the functionality domains presented in this paper and also the interfaces in-between in AUTOSAR. Currently in AUTOSAR, a proposal is made for standardization of the interfaces for requests, capabilities and status as an extension of the Sensor Actuator interface pattern.

4 CONCLUSIONS

The economic value of software will continue to grow for commercial vehicles. This leads us to the conclusion that efficient development and large scale reuse of application software for commercial vehicles, regardless if the execution environment frameworks are AUTOSAR, Adaptive AUTOSAR, Linux, etc., cannot be achieved if there is no mutual agreed architecture from a functionality perspective since deployment will vary. This paper defined an approach for working with large scale reuse of application software for commercial vehicles as it gives guidance on the nature of what kind of application software components there shall be in order to support multi-product families and multi-brand environments. The experiments and prototype trucks developed since 2009, with various degrees of automation levels working together with existing more manually operated features show that this way of structuring application software works. But it also shows that it is a quite different approach compared to current design paradigms explicitly for commercial vehicles and therefore has impact on many existing solutions.

Although many of automotive suppliers share the situations with multiple brands and product families it is also a conclusion that it will be extremely hard to get this through the commercial vehicle community as it will have large impact on many suppliers' business models and their current intellectual property investments. What has been experienced during the years is that intellectual property very often lies in the application software rather than in the electronics. This is especially true for new innovations that are put into market as new features for customer. However, there is a need to standardize a reference architecture and its main domains and horizontal interfaces within the automation community to easier split responsibility between OEMs and various suppliers.

As a result OEMs probably have to change their purchasing principles heavily from purchasing hardware to purchasing intellectual properties, which is completely different than negotiating on material cost. It might be so that this transition cannot be achieved if the OEMs do not perform insourcing of certain application domains at least of for levels above the device abstraction layer.

It is the author's conclusion that this way of looking at application software will have impact on computer and network structure Figure 2. What has been experienced is that this architecture thinking will enable the introduction of more generic computers close to desktop computers were most of the intelligence will be executed and lower intelligent I/O computers sometimes generic and sometimes device-oriented towards for example a Rear Lamp I/O Node or a Gearbox I/O Node where for example functionality related to the device abstraction layer will execute.

REFERENCES

[1] P B Kruchten. The 4+1 View Model of Architecture. IEEE Software November 1995

[2] Frank van der Linden. Software Product Families in Europe: The Esaps & Café Projects. IEEE Software July/August 2002

[3] 4D/RCS: A Reference Model Architecture For Unmanned Vehicle Systems. Version 2.0. The Army Research Laboratory Demo III Program. Aug. 2002, National Institute of Standards and Technology, Gaithersburg, Maryland 20899

[4] Sagar Behre and Martin Törngren. A Functional Reference Architecture for Autonomous Driving, Information and Software Technology, Vol 73. May, 2016 p136-p150

[5] Kevin Baughey. Functional and Logical Structure: A System engineering Approach, SAE International 2011-01-0517

[6] AUTOSAR (AUTomotive Open System ARchitecture). https://www.autosar.org/

[7] Frank Buschmann, Regine Meuiner, Hans Rohnert, Peter Sommerlad, and Michael Stal. Pattern-Oriented Software Architecture (POSA) - A Systems of Patterns, p31-p51, ISBN 0-471-95869-7

[8] Frank Buschmann, Regine Meuiner, Hans Rohnert, Peter Sommerlad, and Michael Stal. Pattern-Oriented Software Architecture (POSA) - A Systems of Patterns, p125-p143, ISBN 0-471-95869-7

[9] Michael A. Cusmano. How Microsoft Makes Large Team Work Like Small Teams. Sloan Management Review. 1997 Fall. p9-p20

[10] Melvin Conway. How Do Committees Invent. Datamation Magazine. 1968 April

[11] John D. McGregor, Linda M. Northrop, Salah Jarrad, and Klaus Pohl. Initiating Software Product Lines. IEEE Software 2002 July/August

[12] SAE 1939/1 Recommended Practice for Control and Communications Network for On-Highway Equipment (and subsequent documents on http://www.sae.org/standardsdev/groundvehicle/j1939a.htm)

[13] Agus Sudjianto and Kevin Otto. Modularization to support multiple brand platforms. Proceedings of the DETC: ASME Design Engineering Technical Confeence September 2001. DECT2001/DTM-21695

[14] Kimon P. Valavanis, Denis Gracanin, Maja Matijasevic, Ramesh Kolluru, and Georgios A. Demetriou. Control Architecture for Autonomous Undererwater Vehicles. IEEE Control System. Vol. 17, Issue: 6, 1997, p48-p64

[15] Systems and software engineering — Architecture description, ISO/IEC/IEEE 42010

[16] Road Vehicles – Fuctional Safety. ISO 26262 http://www.iso.org/iso/catalogue_detail?csnumber=43464

[17] Phil Bianco, Rick Kotermanski and Paulo Merson. Evaluating a Service-Oriented Architecture, TECHNICAL REPORT, CMU/SEI-2007-TR-015

[18 Manfred Broy, Ingolf H. Krüger, Alexander Pretchner, and Christina Salzman. Egineering Automotive Software. Proceedings of the IEEE. Vol. 95. Issue. 2. Feb. 2007.

[19] Magnus Adolfson. ON-BOARD SYSTEM FOR TRUCK PLATOONS - Cooperative mobility solutions for supervised platooning (Companion), Final project conference, Applus IDIADA Technical Centre, Sep. 2016.

[20] Peter Nilsson, Leo Laine, Bengt Jacobson, and Niels van Duijkeren. Driver Model Based Automated Driving of Long Vehicle Combinations in Emulated Highway Traffic, *Proceedings of the IEEE 18th International Conference on Intelligent Transportation Systems*, Spain, 2015.

[21] Sachin Janardhanan, Mansour Keshavarz Bahaghighat, and Leo Laine. Introduction of Traffic Situation Management for a rigid truck, tests conducted on object avoidance by steering within ego lane. Proceedings of IEEE 18th International Conference on Intelligent Transportation Systems. Spain. 2015.

[22] Scaled Agile Framework (SAFe). http://www.scaledagileframework.com/

[23] EEA for the CONNECTED AUTONOMOUS FUTURE. January 2017