Going from an Electronic Unit Centric Development to Application Software Centric Requires a Different Architecture Mindset in Automotive

Anders Magnuson, Volvo Group Truck Technology, Gothenburg;

Abstract

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 fossil fuel usage by introducing electrified torque generation, which could be distributed over several vehicle units in a vehicle combination. Electronics and especially software play a fundamental role for commercial vehicles in order to achieve energy/power balancing, assist a driver to manually operate vehicles effectively in combination with various degrees of automation and doing that dependable 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 (ECUs) – classical 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 four horizontal and two vertical layering of functionality.

Introduction

One of the cornerstones in the automotive industry has been and still is to achieve large scale reuse of manufactured entities in order to provide the market with mass-produced cost efficient vehicles. Thus the overall design thinking, at least within AB Volvo, is characterized by modularization of the products viewing them from a geometric perspective and thus geometric modules and how geometric modules are wired together are in the forefront. All these “modules“ are formed in a “platform“ which can be looked upon as a gigantic shopping bag full of pieces. At Volvo Group Trucks Technology (VGTT) this is known as the Vehicle Module Structure (VMS) and Common Architecture & Shared Technologies (CAST) highlighting generic (geometric) vehicle modules as in Fig. 1 and geometric interfaces. Transferring this perspective into the software landscape; this is similar to a physical view [1], which nowadays is commonly called deployment view.

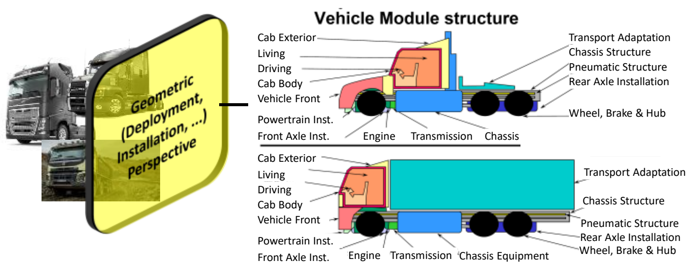


Fig. 1 Main perspective and mindset is geometry-oriented.

This physical thinking has been transferred into the area of electronics where Electric and Electronic Engineering has been focus and still is on where to place mechanic boxes enclosing some electronics and then wire these mechanic enclosures together. These mechanic enclosures are commonly known as Electronic Control Units (ECUs). In early days this wiring was about having dedicated wires going back and forth between ECUs. When these dedicated wires were replaced by communication technologies the focus has continued on wiring the ECUs together, now via Controller Area Network (CAN) and Local Interconnect Network (LIN) links. If you ask for the EE Architecture an automotive company most engineers would provide you with a PowerPoint showing how the ECUs are connected via physical data links as in Fig. 2, but few would present anything similar for the software.

However, already long time before electronics were introduced, when monitoring and control logic was achieved through relays, there has been a structural element known as Electrical Distribution System (EDS). EDS had a focus on how these relays were wired together and the wires carries signals. What has been experienced during the years is that an EDS nor the network topologies fit properly anywhere in a geometric-centric modularization. They so to say give another kind of perspective! Furthermore, the geometric and ECU centric way of looking at the solution has led to that when we are talking about software the focus has been on the ECUs and the binaries that are flashed into their memory.

Also, traditional focus on wires has led to that automotive electric system engineering as a discipline focus on wiring these binaries together via a Signal

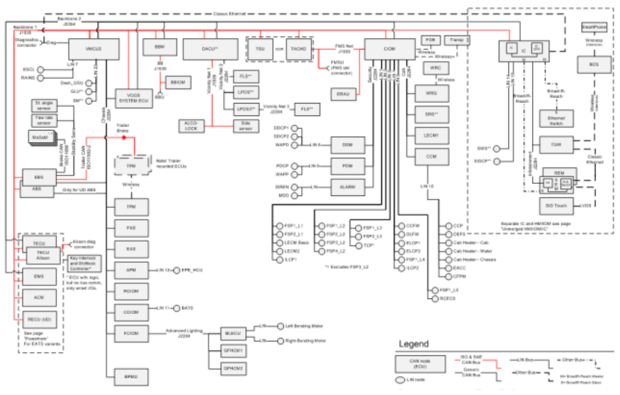


Fig. 2. Network topology is what most look upon as THE architecture in the automotive.

Data Base (SDB). The SDB and EDS are rather similar, where EDS focus on packaging hard wires (signals) into harnesses and SDB focus on packaging soft wires (signals) into frames. Also, the SDB is hard to find in a geometric-centric structure. The software elements that really does something, in Classic AUTOSAR - the Software Components [8], are not really visible and treated, but it is primarily the binaries that are installed in the assembly line that are counted and registered in our product data management tools. All other kind of "systems" are too a large invisible in our product life cycle management tool, e.g. a "system" as in Fig. 2 is not officially released, the description is, but the system with all ECUs is not released as a "system".

System complexity in automotive is increasing

Complexity of a system is rather subjective and is impacted by many things. Already 1986 Fredrik Brooks reasoned about essential and accidental complexity [2] and such as people skill, organization, tools and processes and the system itself are all contributing to the overall complexity. Essential system complexity is a result of "Complex behavior that arises from the inter-relationship, interaction, and interconnectivity of elements within a system and between a system and its environment" [3]. Looking at Fig. 2 the complexity looks quite moderate, but what is missing in this picture is the application software. So when the network topology is supplemented with application software structure, as in Fig. 3, the complexity increases. Thus the traditional view on EE Architecture Fig. 2 gives a rather false picture of the truth! System wide complexity has moved from electronics and wires into the software! What we have experienced is that we can no longer proceed with just the ECU-thinking as system behavior is not clear from the properties of its individual parts.

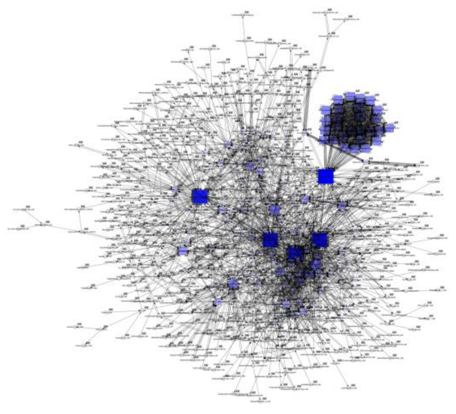


Fig. 3. The complexity increase that comes with interacting software elements.

Continuous software increase to facilitate automation, electrification and connectivity

Nowadays, it is when the application software comes into focus the complexity shows up. That brings us into the topic of where the changes will comes. For example, new legislations on NO_x and CO_2, zero emission zones and noise zones will be enabled through electrification of "powertrain devices" but we must also monitor and control the operation these together with various other high power auxiliaries on trucks such as air fans and air compressors, which thus also needs to be electrified. Thus for example “brake blending” suddenly becomes an energy balancing topic rather than a plain braking topic reducing brake pad wear. 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. This overall coordination only be handled through application software!

Software has become 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 the geometric modularization as in Fig. 1. Cars and trucks have continuously evolved with software enabled features in a relatively moderate pace over the last three decades. 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 [4]. It is important to highlight that not even old software entities such as a Cruise Speed Controller or Cruise Distance Controller are linked to geometric modules as in Fig. 1 like a combustion engine and its associated Engine Control Module (ECM) which today host the Cruise Speed Controller according to SAE J1939/71 [5], [6] (there is an implicit deployment built into this standard). The trend in the Volvo Group is that the pace of innovations through software is accelerating. Also, it will be hard for standards such as SAE J1939 to keep up with this acceleration as it restricts the solution space.

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 a huge amount of application software. As commercial vehicles are used in B2B operations, the interest in automation is perhaps of higher incentive 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. 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. It will be the application software that drives the need for powerful electronics, i.e. flexible and reconfigurable computers! [7]

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. The traditional geometry and deployment perspective is not feasible any longer!

Supplement the Deployment Centric Perspective with a Functionality Perspective

To achieve strategic large scale software reuse it is a necessity to apply a product line engineering approach [9], which sometimes is referred to as “platform development“. As commercial vehicles operates in a diverse set of transportation applications there is a need to build in many and different kinds of variation points (variability) to enable vehicle feature tailoring, while at the same time aim for reusability and changeability in such a product line. One can think of it as a “one software code branch only“ [10] with built in variability. 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. Therefore we are adding an additional perspective on the vehicles – here defined as the functionality viewpoint, Fig. 4 [1], also identified by [11] 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 [12]. 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”.

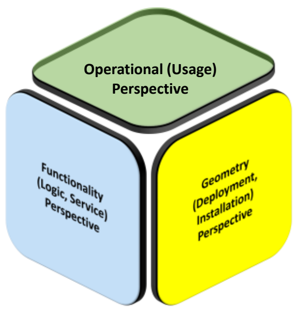
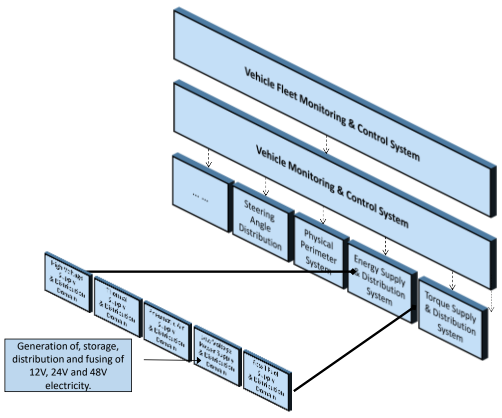

Operational (Usage) Perspective
Functionality (Logic, Service) Perspective
Geometry (Deployment, Installation) Perspective

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

With higher and higher demands on dependable vehicle operation, driven by more advanced features but also by the functional safety standard ISO 26262 [13], 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.

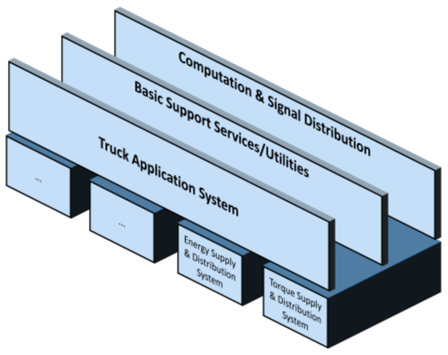
Organizing the Functionality Perspective

So, *how to think when looking upon commercial vehicles from a functionality perspective?* 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 made here 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 Fig. 5. For example the combustion engine from a functionality perspective would then reside in the Torque Supply & Distribution System but when it is installed (deployed, packaged) it can be under the cab at the truck, in front of the cab in a conventional truck, in rear or at the center in a bus. So this thinking does not just support software. We thus at the highest level apply a layered approach. As can be seen in Fig. 5 there are dependencies from the Vehicle Monitoring & Control System (VMCS) to various “systems”, which will mainly be handled by the Device Abstraction Layer introduced in Fig. 7. Furthermore, as the nature of the functionality dealt with in the VMCS 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 “system” entities as in Fig. 6, which is a kind of layered architecture style [21]. As these are separated from each other the reasoning and focus in these also varies a lot. Another thing worth to highlight is that VMCS focus on single vehicles. However, to gather statistics of wear or energy consumption trends of for example all Volvo, Renault and Mack trucks, there is a need to add another layer on top as in Fig. 5, here called Vehicle Fleet Monitoring & Control System (not in the scope of this paper).



Vehicle Fleet Monitoring & Control System
Vehicle Monitoring & Control System
... ...
Steering Angle Distribution
Physical Perimeter System
Energy Supply & Distribution System
Torque Supply & Distribution System
High Voltage Supply & Distribution Domain
Thermal Supply & Distribution Domain
Pneumatic Air Supply & Distribution Domain
Low Voltage Power Supply & Distribution Domain
Fossil Fuel Supply & Distribution Domain
Generation of, storage, distribution and fusing of 12V, 24V and 48V electricity.

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



Computation & Signal Distribution
Basic Support Services/Utilities
Truck Application System
...
...
Energy Supply & Distribution System
Torque Supply & Distribution System

Fig. 6. The Vehicle Monitoring and Control System is divided into three major blocks of functionality.

<u>Computation & Signal Distribution System</u>: the focus is on electronic functionality <u>interfacing</u> various actuators and sensors that are part of the device functionality such as interfacing a Fan Motor, an Inlet Air Pressure Sensor, and Brake Pad Wear Sensor etc. It also focuses on processing and memory capacity as well as network structures its performance. We can say that this system is the classical way of looking at an “EE system“ – the ECUs and their network connections as in Fig. 2. That is the diagram in Fig. 2 is a “document” that only describe the internal design of this system and that released when the system and all its content is released.

Vehicle Application System: the focus is on managing entities that owns and deals with information about the vehicle and thus used to monitor and control the operation of the vehicle. These entities 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. It is this system that holds all application software functionality all the way down to source code modules.

Basic Support Services/Utilities System: The world in-between these is some kind of middleware functionality that bridges these two worlds, where AUTOSAR Classic Platform [8] is just one such middleware framework, Linux another one. Graphic Engines, Voice Interprets also belongs to this system.

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 Vehicle Application System. Furthermore, as the hype around agile/lean based development process frameworks like SAFe [14] 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.

Vehicle Application System - layer application functionality

By work and experiments conducted during 2009-2018 at Volvo GTT it has been concluded that there is a need for a slightly different layering than in [15], among other things the architecture presented herein has added clear separation of HMI and the ego vehicle’s environment and operational functionality has been divided into two

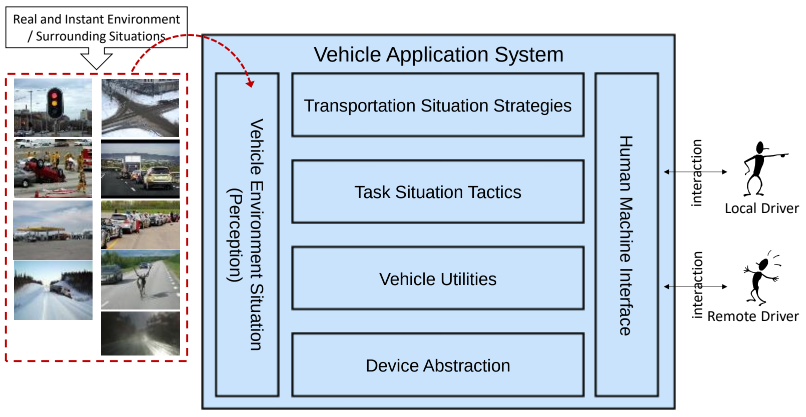

Real and Instant Environment / Surrounding Situations
Vehicle Application System
Vehicle Environment Situation (Perception)
Transportation Situation Strategies
Task Situation Tactics
Vehicle Utilities
Device Abstraction
Human Machine Interface
interaction
Local Driver
interaction
Remote Driver

Fig. 7. Basic layering of the application software inside Vehicle Application System.

layers. The reference architecture presented in this paper proposes four horizontal layers as in Fig. 7 where two lower layers address “operational” functionality; one layer deals with tactic functionality; and one layer deals with strategic functionality, but also two vertical layers are added. Each horizontal layer raises the abstraction level from the “physical” functionality that is to be monitored and controlled. All with the perspective of a single vehicle, but the vehicle may at the vehicle utility layer be looked upon as two track model, as single track model in the task situation tactics layer and as a particle in the transportation strategies layer!

The architecture approach defined here combines a strict hierarchical style [16] with a heterarchical style [17] 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 variable product line. A similar layered architecture for platooning feature of commercial heavy vehicles has been presented in [18], which also contains strategic, tactical and operational layers and it can be seen that the focus has been solely on platooning. In the presented approach here, the platooning planning is managed in Transportation Situation Strategies and the actual joining and leaving is taken care of in Task Situation Tactics, e.g. the “adaptive cruise controller“ is becoming a “platoon cruise controller“. In [18] 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.

Device Abstraction Layer: The overall thinking in this layer is that application software entities shall be representations, device abstractions (DA), 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 Fig. 8.

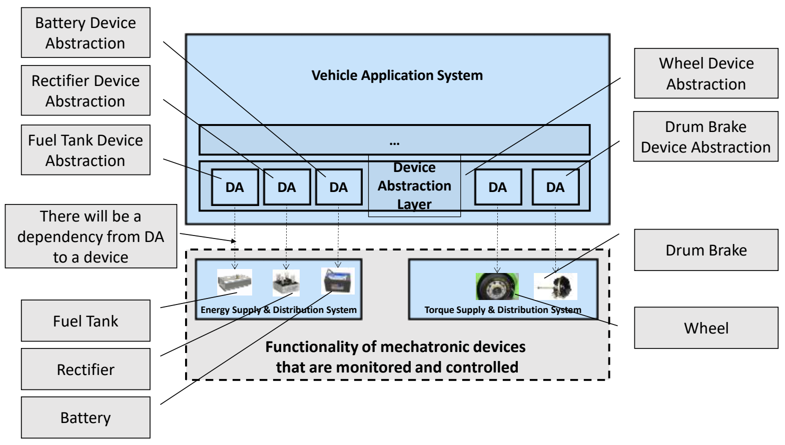

Battery Device Abstraction
Rectifier Device Abstraction
Fuel Tank Device Abstraction
There will be a dependency from DA to a device
Fuel Tank
Rectifier
Battery
Vehicle Application System
...
DA
DA
DA
Device Abstraction Layer
DA
DA
Energy Supply & Distribution System
Torque Supply & Distribution System
Functionality of mechatronic devices that are monitored and controlled
Wheel Device Abstraction
Drum Brake Device Abstraction
Drum Brake
Wheel

Fig. 8. Device abstractions are supposed to be representations of modules of the functionality that is monitored and/or actuated.

DAs are carriers of information elements that represent various properties of a mechatronic element that is to be monitored or controlled. For example *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 Das are looked upon as service providers and does not necessarily run on its own ECU. 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. Furthermore, this thinking is also related to that there might be different variants of these devices such as a Disc Brake and a Drum Brake and as these share many properties it is good if they are defined in a single point, Friction Brake, as in Fig. 9. 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.

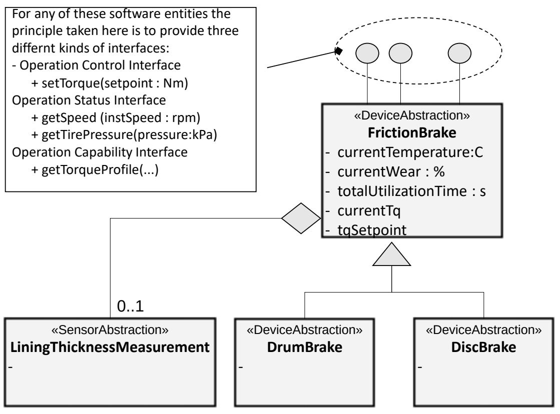

For any of these software entities the principle taken here is to provide three differnt kinds of interfaces:
- Operation Control Interface
+ setTorque(setpoint : Nm)
Operation Status Interface
+ getSpeed (instSpeed : rpm)
+ getTirePressure(pressure:kPa)
Operation Capability Interface
+ getTorqueProfile(...)
«DeviceAbstraction» FrictionBrake
- currentTemperature:C
- currentWear : %
- totalUtilizationTime : s
- currentTq
- tqSetpoint
0..1
«SensorAbstraction» LiningThicknessMeasurement
«DeviceAbstraction» DrumBrake
«DeviceAbstraction» DiscBrake

Fig. 9. A DA with two variants and three different types of interfaces.

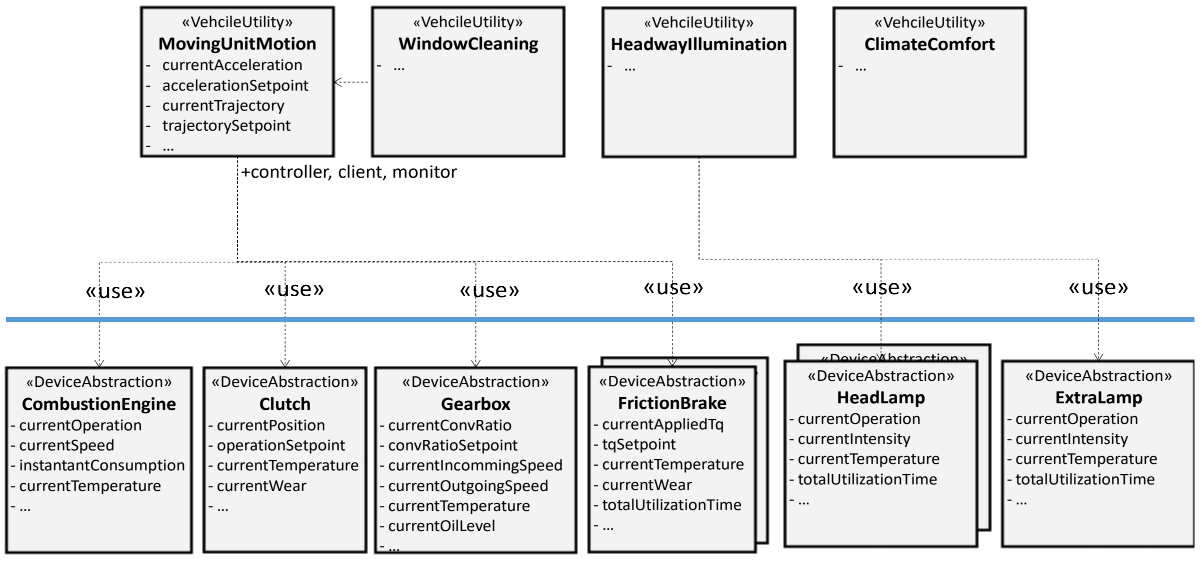


Fig. 10. Example of Vehicle Utilities that are acting as controller, clients and encapsulate coordination between different device abstractions.

Vehicle Utilities Layer: The purpose for this layer is first of all to raise the abstraction level from individual device abstractions into so something we call vehicle utilities (services). These shield away individual devices. A vehicle utility is defined as some kind of useful vehicle wide service that enables a client to perform one or many of its activities via an HMI in a manually operated truck or when automated by elements in the Task Situation Tactics Layer. Hence *a Vehicle Utility represents a goal experienced* by the consumer e.g. a Moving Unit Motion (for trucks there can be many units that form a vehicle combination), Power Situation, Headway Illumination, Window Cleaning, Climate Comfort etc. as in Fig. 10. When a Vehicle Utility is operational it is utilizing lower level DAs to achieve its desired goal and thus it will take on many different roles towards DAs such as controller, client, monitor, and coordinator and towards elements in the HMI or Task Situation Tactics layer Vehicle Utilities will works as servers.
What we have done is that this layer will in practice be replaced through a number of domain packages organizing a set of vehicle utilizes as seen in Fig. 11. We intend to look upon these

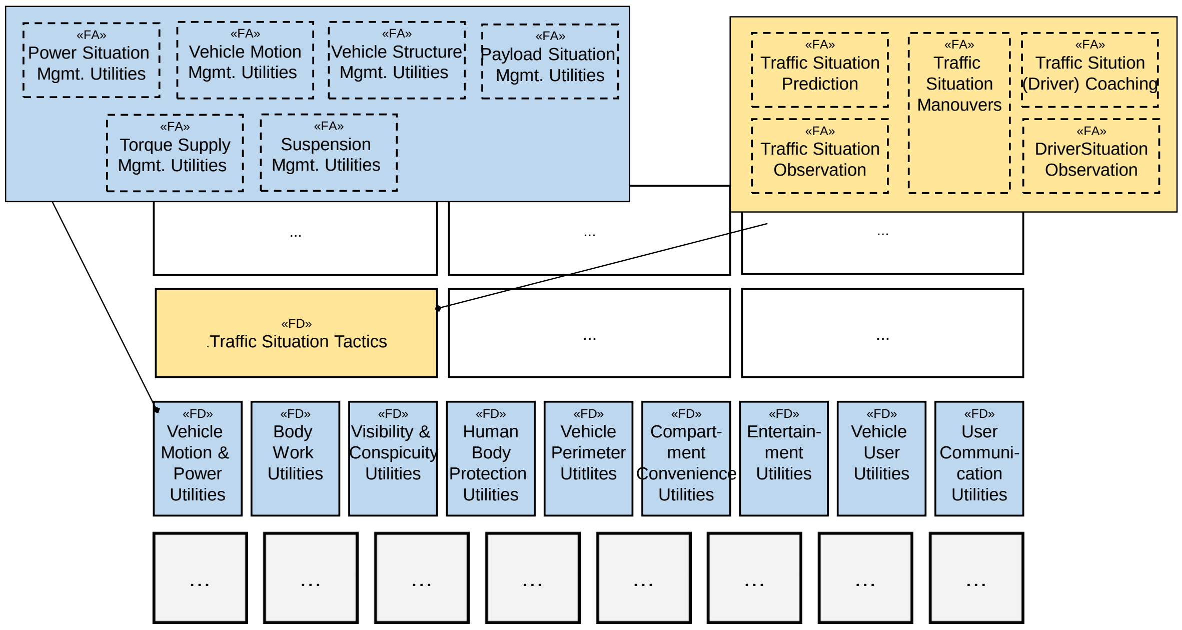

«FA» Power Situation Mgmt. Utilities
«FA» Vehicle Motion Mgmt. Utilities
«FA» Vehicle Structure Mgmt. Utilities
«FA» Payload Situation Mgmt. Utilities
«FA» Torque Supply Mgmt. Utilities
«FA» Suspension Mgmt. Utilities
«FA» Traffic Situation Prediction
«FA» Traffic Situation Manouvers
«FA» Traffic Sitution (Driver) Coaching
«FA» Traffic Situation Observation
«FA» DriverSituation Observation
«FD» Traffic Situation Tactics
«FD» Vehicle Motion & Power Utilities
«FD» Body Work Utilities
«FD» Visibility & Conspicuity Utilities
«FD» Human Body Protection Utilities
«FD» Vehicle Perimeter Utitlites
«FD» Compart-ment Convenience Utilities
«FD» Entertain-ment Utilities
«FD» Vehicle User Utilities
«FD» User Communi-cation Utilities

Fig. 11. In practice, a layer will be a set of application domains looked upon as “products”.

as “products“ that are released, maintained, evolved, etc. and thus they enable us to point out “product manager/owners“, although its internal elements might be distributed and executed in a number of ECUs. Furthermore, it is these domains that facilitate our organization to talk about the application software as the network topology and its ECUs has supported automotive organizations in the past. As there can be quite many software elements in these, these are further decomposed into some intermediate products which we denote areas as seen to upper left in Fig. 11.

Task Situation Tactics Layer: When a human user performs a use case such as "transport payload from A to B" or "load/unload payload" is actually about 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 both a monitor and controller. The purpose of the task situation tactics 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" services offered by 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)* and apply some tactics to perform these. For example, in the case of a driver in the control loop, the driver is actually acting as a *driving controller, speed controller*, a *trajectory controller* (steering), *headway sight 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 Tactics* in Fig. 11. As seen in Fig. 11 and in [19] and [20] the *Traffic Situation Tactics* is broken down into smaller functionality areas (FA) such as *Traffic Situation Observation* (a local world model [16]), *Traffic Situation Prediction*., *Traffic Situation Maneuver (*plan, perform, and perform maneuver's both L2 to L5 automation), Traffic situation (*Driver) Coaching* and *Driver Situation Observation*.

Transport Situation Strategies 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 transportation (mission) 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.

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 an orthogonal layer to the horizontal layers as seen in Fig. 7. For example a *Weather Situation*, *Traffic Jam Situation, Environment Traffic Situation* and a *Road Characteristics Situation* are software entities located herein and provides services to the horizontal layer. Thus they abstract away how that is gained from making of other more internal software entities, e.g. a Weather Situation entity can make use of detections in an image from a camera, clutter from a radar and/or detection from dedicated temperature sensors as well as communication with another vehicle to get a proper picture of the complete weather situation in the ego vehicle’s environment.

Human Machine Interface “layer“ - think Model-View-Controller: 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 and that human, acting as a driver, which can be either locally or remotely as in Fig. 7. This is especially important to consider in product line that intend to enable trucks within the range from manual to full automation. The approach taken here is to clearly separate Human Machine Interface (HMI) as in Fig. 7, where View and Controller according to MVC pattern [21] resides herein. 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 also opens up for a possibility to localize what kind of control a user can do and when that can be done as well as 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. The architecture also has a similar User Output Controller that provides feedbacks valid for a human user.

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. Having this said, 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 highlighted in Fig. 9 and Fig. 10, as the lowest level in software modularization within the layers. These objects are acting as service providers and service consumers. The intention is also that these will be individually 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. Achieving a directed dependency, e.g. as depicted in Fig. 10, *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 Objects residing in these domains, areas, units, etc. (a successive hierarchic structure) Thus we propose that software elements provide three different kinds of interfaces, as visualized in Fig. 10: Operation Control, Operation Capability (instantaneous capabilities), and Operation Status. 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.

Conclusion

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 development work 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 [22] and we conclude the same but all recognize that is really hard to establish such thinking a large global organization. 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 [23] such internal line organization structure, structure of agile release trains, and various tiers of suppliers (Conway Law). Thus, going in this direction putting the application software in the forefront rather than the ECUs it will have large impact on many suppliers' business models and their current intellectual property investments as intellectual property very often lies in the application software rather than in the electronics as well as our internal organization and such transitions are painful. Thus, there is no need to debate that development of software functionality for commercial vehicles would be vastly different.
Furthermore, taking a functionality-oriented perspective is more or less a must to move towards more service-oriented solutions [24] and focusing in “logical structures” and “logical relationships”. Service-oriented solutions will also be an enabler for more self-going agile teams. How to take full advantage of that we also needs to move towards more IP-based solutions instead of CAN/LIN-oriented protocols.

References

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

[2] Fredrick P. Brooks Jr, No Silver Buller – Essence and Accidents in Software Engineering, TBD

[3] Mitleton KE, Land F (2012) Complexity & Information Systems. Blackwell Encyclopaedia of Management.

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

[5] SAE J1939/71 Vehicle Application Layer (The data parameters (SPNs), messages (PGNs) and reference figures and information previously published within this document are now published in SAE J1939DA, Published October 25

[6] SAE J1939 Digital Annex (has been broken out from the SAE J1939/71), Published April 22, 2019

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

[8] Virtual Functional Bus, AUTOSAR Classic platform, Release 4.3.1

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

[10] Hans Aerts and Han Schaminée, How Software Is Changing the Automotive Landscape, IEEE Software November/December 2017

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

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

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

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

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

[16] 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

[17] 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

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

[19] 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.

[20] 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.

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

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

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

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