CHALMERS
UNIVERSITY OF TECHNOLOGY

Virtual vehicle modeling architecture with centralized control

Mario Majdandzic
Pooja Sousthanamath

Department of signals and systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis 2018:NN

Virtual vehicle modeling architecture with centralized control

Mario Majdandzic
Pooja Sousthanamath

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Signals and Systems
Chalmers University of Technology
Gothenburg, Sweden 2018

Virtual vehicle modeling architecture with centralized control

MARIO MAJDANDZIC,POOJA SOUSTHANAMATH

Supervisor: Anne Piegsa, Semcon
Examiner: Kristofer Bengtson, Signals and Systems

Master's Thesis 2018:NN
Department of Signals and Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2018

Virtual vehicle modeling architecture with centralized control

MARIO MAJDANDZIC, POOJA SOUSTHANAMATH
Department of Signals and Systems
Chalmers University of Technology

Abstract

As autonomous driving technology matures toward series production, the amount of electronics in vehicles is growing quickly making the engineering of software intensive components more complex and difficult. It is necessary to take a deeper look at various aspects of vehicle modeling architectures for autonomous driving. This paper describes a functional reference vehicle modeling architecture(VMA), along with various considerations that influence such an architecture. With an aim to evaluate the current state-of-the-art architectures and the new revised functional VMA based on quality attributes (such as reusability, scalability and modularity) to reduce complexity. The functionality is described at the logical level, without dependence on specific implementation technologies. The description of functional VMA includes it's main components and the rationale for how these components should be distributed across the architecture and its layers. A comparison with similar architectures is also provided, in order to highlight the similarities and differences. The comparisons show that in the context of automated driving, the explicit separation of concerns, decoupling of hardware and software components and vehicle platform abstraction are unique to the proposed centralized control architecture These components are not unusual in architectures within the Artificial Intelligence/robotics domains; the proposed architecture shows how they can be applied within the automotive domain. The outcome of the concepts presented in the revised architecture is expected to trigger further research initiatives in this challenging area.

Keywords: VMA, centralized control, quality attributes.

Contents

1
Introduction

The accelerated rise of new technologies, sustainability policies, and changing consumer preferences in the vehicle industry has given rise to a challenge with regards to test and verification [1]. Reduced time and cost frames lead to fewer physical test vehicles. In addition, much of the functionality cannot be tested in real world traffic due to safety reasons. In order to reduce the usage of resources on building different prototype variations for testing of functions, realistic mathematical models of vehicles can be used to simulate the behaviour of a vehicle. This opens up the possibility for analysis and understanding of different parameter interaction and impact this has on the simulation outcome, without the need of a otherwise required physical prototype. Analysis and simulation is specifically important for autonomous vehicles due to rare and potentially dangerous situations and maneuvers, that must be inspected to reach adequate certainty to build the prototype. The current trend is to replace the testing of prototypes with simulations, in order to save time, money and predict its performance in the real world to get a broader understanding of the complete system.

With a growing list of commercial, free or internally-developed OEM proprietary model libraries, the need for a unifying Vehicle Model Architecture (VMA) was quickly realized in the automotive industry [2]. The purpose of a standardized model architecture is to provide consistent interfaces and system decomposition to promote plug-n-play interoperability between model libraries [3].

Electronics in automobiles has come a long way since their introduction in this industry a few decades ago. According to industry experts, 80% to 90% of the innovation within the automotive industry is based on electronics. Automobiles of today rely on electronics more than ever before; to not only provide better implementation of functionalities but also to provide new, paradigm changing functionalities [4]. A big part of electronics is software [5]. Also, with functions such as Lane Assist, Automated Emergency Braking, and Adaptive Cruise Control, the current generation of automobiles has taken the first steps towards automated driving. The automotive industry is heading towards highly and even completely automated vehicles. This is no simple task. Among other things, a scalable systems architecture is needed that covers all current and future functions and integrates them into one fully automated system.

Vehicle system modeling is an important part of optimizing overall vehicle performance among other things. To avoid building up complete vehicle models from

scratch repeatedly, it is useful to develop a predefined vehicle model architecture. The VMA should allow the exchange of subsystem models between different organizations (e.g. part/subsystem vendors, design organizations, universities) without the need to "rework" the models to fit into existing vehicle system models. It should also simplify the handling of alternate vehicle system configurations by allowing substitution of one particular subsystem or strategy implementation for another [6].

In this thesis, A study will be done on various existing state-of-the-art architectures and analyze them based on the quality attributes such as reusability, modularity and scalability to reduce complexity. The main purpose of this work is to propose a new classification of integrated centralized control architectures to overcome the limitations posed by traditional architectures and also to accommodate the growing demand of the automotive industry.

1.1 Ethical and social aspects of modeling and simulation

Increasingly, prototypical self-driving vehicles are participating in public traffic [7] and are planned to be sold starting in 2020 [8]. Self-driving vehicles will combine data from inside vehicle with external data coming from the environment (other vehicles, the road, signs, and the cloud). In such a scenario, different applications will be possible: smart traffic control, better platooning coordination, and enhanced safety in general. However, the basic assumption is that future self-driving connected cars must be socially sustainable. Simulation platforms enable the artificial intelligence brain powering an autonomous vehicle to run in a photo-realistic world that mimics real-life traffic [9]. Manufacturers can also test hardware using a process called hardware-in-the-loop, in which one server simulates the driving environment while another contains the computer that will eventually run in the car. When combined with public road testing, this type of simulation creates a robust validation process in a fraction of the time it would take to drive an equivalent distance in the real world. While scripting the infinite number of potential traffic situations may be impossible, enabling diversity and spontaneity in simulation is a vital way to test a car's reaction to unforeseen scenarios, validating autonomous vehicle technology without sacrificing safety [9]. Autonomous vehicles—particularly those that are passenger cars—could significantly affect the country's ability to cut greenhouse gas emissions and move toward a carbon-free economy. Therefore, autonomous vehicles must be assessed not only for their safety but also for their effect on carbon emissions levels [10]. Efficient simulation of different scenarios causing carbon emissions can relatively reduce the emission impact on society.

1.2 Background - Role of architectures in Open Innovation Lab

Vehicle manufacturers are investigating open innovation and third party development as a source of creativity and new ideas [11]. With the growing trend of adapting shorter development time, high functionality and reliability are required to ensure robustness of the product. The Open Innovation Lab (OIL) research project which is being conducted by Semcon AB and four other partner organizations aims at taking the innovative step of exploring the potential of simulation-based testing methods as support for open innovation and third party development, by establishing a common platform and tool-chain for collaboration around different simulators.

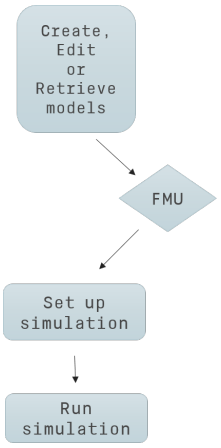

Figure 1.1: The current work-flow in OIL project

OIL is a collaborative project among five automotive organizations, who would be contributing their generic models to be stored in the platform's repository for engineers/users to access them. Fig. 1.1 shows the simple work-flow in the OIL project. The number of models would grow exponentially when the user makes variants of the existing models to test and verify different features. In the automotive industry, there is a thriving demand for innovation and new functionality in modern cars, this calls for a paradigm shift in automotive system architectures both in terms of hardware and software. The key intent of OIL for this thesis is to propose a standardized model architecture that would provide consistent signal interface and system decomposition to promote plug-n-play interoperability between models and also to support the quality attributes such as reusability, modularity and scalability to reduce complexity. Architectures for vehicular electrics, electronics, communication, and software are facing several challenges driven by strategic trends like automated driving, artificial intelligence, drivetrain electrification and connected systems and services. It require innovations to reduce system complexity and improve reusability, scalability, modularity and system integration, consequently driving the introduction of new technologies. Factors like control hierarchies, distribution of functionality, arbitration and conflict resolution etc. are considered for careful evaluation and a summary of the analysis is presented.

1.3 Aim

The master thesis project aims at:

- Analyzing the current state-of-the-art architectures and highlighting the limitations posed by them.
- Proposing an architecture that overcomes the posed limitations and integrate the growing demands of the automotive industry.
- Evaluating the quality of the current and proposed architecture using quality attributes(reusability, modularity and scalability) to reduce complexity.
- Structure the architecture in such a way that it takes in new/innovative ideas but also stays relevant to the current automotive community.
- Initiating a step towards using the architecture as a guideline for potential solutions within future vehicle architecture design decisions.

1.4 Limitations

The main focus of the thesis project is described in the previous section 1.3 which are found to be feasible for the time span of the project. Apart from these, a substantial amount of issues/tasks would need extensive analysis which would be out of scope of this thesis work

- The goal is to analyze and propose an architecture based on analysis. Creating a prototype simulation architecture to see the full benefits of the architecture is not under the scope of this thesis work.
- The role of Plant modeling (causal or acausal) approach is discussed. However, a thorough study regarding the same is necessary to view it's benefits in architecture design which will not be covered in this thesis work.

1.5 Problem definition

1. The current state-of-the-art architecture is outdated or will soon be, considering the technology advancements in vehicle industry.
2. Rework in building the models and consequently vehicle modeling architecture is proven to be expensive and time consuming.
3. With the growing software components, the VMA seems to be getting complex with each addition.
4. The current architectures are rigid and do not look scalable to future demand in vehicle functionalities.

1.6 Research questions

The thesis is based on the following research questions:

1. **What are the current state-of-the-art VMAs? How do they differ from each other? What are their limitations?**
2. **How can the structure of current vehicle modeling architecture change to accommodate growing future automotive functions.**
3. **How does the new revised VMA differ from the current VMAs based on quality attributes such as reusability, modularity and scalability to reduce complexity**

1.7 Outline of the report

The report is partitioned in the following way: Chapter 2 (Methodology) examines the term Modeling & Simulation and modeling architecture, how it is used in automotive context. It further discusses the concepts of quality attributes which is necessary to understand the discussion in later chapters. Chapter 3 (Analysis) consists of detailed analysis of VMAs and the problems encountered by automotive community due to limitations in VMAs and rapidly growing technology in vehicles. Along with this, different ways of tackling the problems is also discussed. Chapter 4 (Result) proposes a revised centralized control architecture that is formulated after considering the limitations and demands of the current industry. This chapter also discusses in detail about the proposed architecture and how it overcomes the limitations at the same time giving way to accommodate future technology demands. In the final Chapter 5 (Conclusion and Future work), a conclusion of the entire work is presented along with discussion about above mentioned research questions. Potential future work that needs to be carried out to yield better benefits from the vehicle modeling architecture is also discussed.

2
Methodology

2.1 Introduction to Modeling and Simulation

Simulation is a representation of the functioning of a system or process. The act of simulating a system or process first requires the development of a simulation model. A simulation model is a descriptive composition of a process or system including parameters that allow the model to be configurable to represent a number of different systems or process configurations [12]. It is said to incorporate logical, mathematical and structural aspects of a system. Simulation allows the identification of problems, bottlenecks and design shortfalls before building or modifying a real system. Evaluation, comparison and analysis are the key reasons for doing simulation. Prediction of system performance and identification of system problems and their causes are the key results[12]. Simulation proves to be advantageous in the following scenarios among others [13]:

- Analyze the situation with sufficiently accurate calculation of the complex analytic model.
- When the real system has some level of complexity, interaction or interdependence between various components, or pure size that makes it difficult to grasp in its entirety. In particular, it is difficult or impossible to predict the effect of proposed changes.
- When a new system or functionality is being developed, considering major changes in physical layout or operating rules in an existing system, or being faced with new and different demands.
- When a large investment in a new or existing system is being considered with system modification of a type for which little or no experience is present and hence considerable risk can be foreseen.
- To compare several alternative designs and rules of operation.

At times, Simulation can pose certain limitations, for instance simulation often requires extensive resources to be spent on building and validating models. It does not give an explicit solution to the problem, but rather a comparison between different solutions. Nevertheless, a good simulation model not only provides numerical measures of a system, but provides insight into desired system performance for each configuration of interest [12].

2.1.1 Modeling architecture

A Modeling architecture is the functional decomposition of a whole system into parts, with specific relations among the parts. One of the key requirements of modeling architecture for complex systems is that the model construction should be supported by hierarchical and modular composition of reusable models in a uniform and scalable way [14]. It aids people to work cooperatively and productively together to solve a much larger problem. Simulation and architecture are closely interlinked, quality attributes such as reusability, modularity and scalability are every bit as important as obtaining accurate results. The modeling architecture helps in addressing the demand to meet the quality attributes in the following ways [15]:

- If the focus is on reusability, the system is expected to encompass careful separation and granularity of different parts of the system so that these can be reused in multiple configurations as and when necessary.
- If the focus is on scalability, the system can be carefully separated among the parts of the system, so that when a change affects one element, that change does not ripple across the entire system.
- If the focus is on modularity, the system is deployed incrementally, by releasing successively larger subsets, the architecture helps in reducing the interdependency to avoid the "nothing works until everything works" syndrome.
- Reusability, scalability and modularity in turn aid in reducing the complexity of the system.

2.1.2 Quality Attributes : Reusability, Modularity and Scalability

Reusability, modularity and scalability can be defined as three main quality attributes in modeling of automotive systems which can help in reduction of complexity of the system. Quality attributes are the characteristics which provide basis for evaluating quality of the system under discussion [16]. The key approach in the project would be to enable architecture analysis of existing systems via above mentioned quality attributes.

Reusability

Reuse is the isolation, selection, maintenance and utilization of existing software artefacts in the development of new systems [17]. Reuse can mean different things in various context from the reuse of small portions of code, through component reuse, to the reuse of complete models. On a more abstract level, component design, model design and modeling knowledge are prime candidates for reuse. The reuse of simulation models is especially appealing, based on the intuitive argument that it should reduce the time and cost for model development [18]. The idea of modellers saving time and money by reusing their own, or other peoples' models and model components is appealing, and technology is apparently making it more possible [17].

A spectrum can be used to represent different types of software reuse as shown in the Fig. 2.1. The spectrum is cast in terms that are fairly recognizable to the simulation community [17].

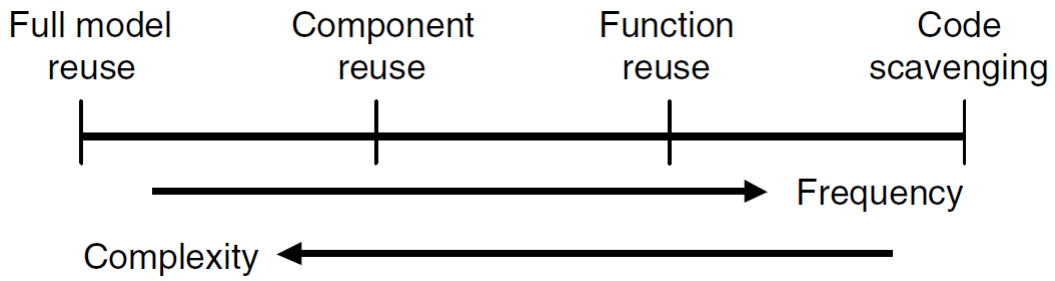

Figure 2.1: A spectrum of reuse

Code scavenging is a process of reusing existing code with slight or no modification (by copy and paste). Since the working code is already available the complexity in reusing the code is low.

Function reuse is the next step along the spectrum of reuse which involves reusage of built-in functions from particular languages or systems. The functions that are reused in this way are usually very specific in their functionality and are fine grained, which enables checking that the function is performing as required.

Component reuse For present purpose, a component is defined as an encapsulated module with a defined interface, providing limited functionality and able to be used within a defined architecture. Components are usually larger than functions and the complexity increases in reusing these components. It gets a little more tricky as the components get bigger and offer broader functionality.

Full model reuse Full model reuse might imply, at one extreme that the executable model is used in an environment other than that for which it is developed. This, clearly raises many issues about validity. On the other hand, the model might be reused many times for the same purpose, which is relatively straightforward. However, the complexity increases significantly when reusing the full model.

The spectrum in Fig. 2.1 shows four types of software with two different horizontal axes. The first, frequency, indicates that reuse is much more frequent at the right-hand end of the spectrum, which is code scavenging. The second axis, complexity, runs in the opposite direction, making the point that reuse of code is relatively easy, whereas successful reuse of entire simulation models can be very difficult indeed.

In the analysis that follows in the paper, Component/Model reuse will be of major interest, model reuse is considered to be synonymous with software reuse and deserves no special treatment in a simulation scenario [18]. The benefits of reuse can be classified into both qualitative and quantitative benefits [18] are discussed below:

Qualitative benefits:

- Fewer defects: Reusing models involve more feedback and thus more debugging that finally improve the quality and reliability of models.
- Productivity improved: Although the introduction of systematic reuse can have a negative impact on productivity in the beginning, long term systematic reuse improves productivity.
- Interoperability: Applications that share the same component are made de facto interoperable if the shared component allows them to communicate.

Quantitative benefits:

- Reduced development: and thus less time spent for developing new models.
- Time to market: is shortened due to the reduction of development time.
- Less documentation to write, reusing models means that we could also reuse dedicated documentation.
- Less maintenance: Reusability reduces the amount of variability among models and hence does not need huge maintenance tools leading to reduction in maintenance costs.
- Additional training costs: An additional cost to train engineers to handle reusable components must be took into account, but that cost is quickly amortized long-term.
- Small team: Dependency on experts/people for development of high fidelity models within a certain domain is reduced. It results in resource utilization and better communication between team members and thus an increase in the productivity.
- Quick prototyping: due to the assembly of existing component.

Challenges in reusability

Though reusability has the above discussed benefits, it does introduce new challenges [19]:

- How can a model user be confident that a planned reuse of the model is within the range of uses intended by the model creator?
- How can one characterize the uncertainty of a model that is reused (possibly with some adaptations to a new context)?
- How can one characterize the uncertainty of simulation models obtained through the composition of multiple models?
- How can one accelerate the process of adapting and reusing models for different purposes? What are the fundamental limitations of technologies for model reuse?

Modularity

Modularity is a design pattern that is built around the idea of autonomous modular components that can be independently created, easily configured and reconfigured into different systems [20]. It is a key feature of complex engineered systems for a number of reasons. Firstly, by definition, complex systems are highly interconnected. In these highly interconnected systems, the cost of interaction will be very low. This low collaboration cost makes it much easier to unbundle homogeneous systems, distribute them into modules, and then reconnect them into a composite whole system. Secondly, complex systems are composed of autonomous elements, meaning the complex systems cannot fully be constrained within one integrated top-down model. Componentization gives each element a degree of autonomy, thus allowing it to adapt. For developing a modular system, it requires, a module, an interface, and a set of protocols for interconnecting those modules. Firstly, in order to create modules, it is necessary to unbundle the monolithic system, this is called separation of concerns. The idea is to capture what makes each component a separate concern, that is to say, autonomous and different from everything else. Modularization and mass distribution of components can be used as a highly effective way of engineering complex system.

Advantages of modularity: Modularity provides the following benefits

- It can enable distributed collaboration and problem solving, when dealing with a very large complex system that would require an expert to fully grasp and manage. Through modularizing the system, various functions can be more easily distributed across a large team with no team member creating or even understanding the whole system.
- It enables sufficient reuse of modules. Modules can be combined and recombined, making it more likely to be a sustainable solution. It should also be much easier to manage and maintain the life-cycle of the system. Individual components showing fatigue can be replaced without having to replace the whole system, and within minimal time-frame.
- Modular systems are much more versatile, adaptive and can be customized more easily. The modular design employed in the automotive industry makes it easier for automotive manufacturers to offer a wide variety of customizations, where they simply “snap in” upgrades, such as a more powerful engine or seasonal tires.
- Modular design can be a very important mechanism in creating autonomy. By designing for autonomy, the dependencies are reduced within the system and the modules act as natural buffers to disaster spreading and for maintaining security.

Limitations of modularity:

- Because everything has been dis-aggregated, everything will have to go through some network of interactions to take place. Unless the cost of interaction is

very low, this will place a very high burden on the system.

- Unless the protocols and interfaces to the modules are well designed, we can waste a lot of time continuously negotiating contracts between modules.
- Excessive modularization can lead to a fractured system
- modular systems may be good for a lot of things but they are not optimized for performance.

Challenges related to coping with modularization [21]:

- Automotive suppliers are required to acquire more systematic technology for parts. Systematic technology is related to creation of sub-functions, components and the structure of the product, in order for it to perform its specific functions [22]. This is required to optimally combine and integrate parts.
- It is necessary to improve technology that functionally and structurally combines parts, such as improving developmental capabilities for new processing, materials and component technologies. This is the direction that has been taken from the past in technological development and formation at automotive suppliers.
- Automotive suppliers who undertake design development and production of modules require the ability to adjust functional and structural interfaces in complex product designs. There is a trend towards ongoing expansion in entrusting development tasks to automotive suppliers. If modularity in design is sufficiently advanced, it is possible for automotive suppliers to devote their efforts to development within their modules and easily reduce the developmental process, costs and lead times.

Scalability

Scalability is defined as the ability to handle the addition of systems or objects without suffering a noticeable loss in performance or increase in complexity [23]. Scalable architectures provide the flexibility to integrate software from various sources by effectively enabling the integration of new innovations [4]. Scalability is very crucial in large and complex systems, poor scalability can result in poor system performance, necessitating the re engineering or duplication of systems. When the system is said to be unscalable, it usually means that the additional cost of coping with a given increase in complexity or size is excessive, or that the system cannot cope at this increased level at all. The scalability of a system subject to growing demand is crucial to its long-term success. At the same time, the concept of scalability and our understanding of the factors that improve or diminish it are vague and even subjective. Many systems designers and performance analysts have an intuitive feel for scalability, but the determining factors are not always clear. They may vary from one system to another. General reusable design patterns have been discovered in the past for building scalable systems [24]. One such solution is parametrization. Parametrization is a mathematical process consisting of expressing the state of a

system, process or model as a function of some independent quantities called parameters [25]. Parametrized model aims at generating low-cost but accurate models that characterize system response for different values of the parameters. Parametrized model is important for applications in design, control, optimization, and uncertainty quantification—settings that require repeated model evaluations over different parameter values [26]. However, when the system grows and becomes more complex, parametrization has to be planned carefully to avoid unnecessary use of parameters, this leads to large overhead for the system [27]. Parameters can be defined individually for each component or have default common parameters across various platforms. The planning for either kind of parametrization has to be carried out to ensure scalability and flexibility [27].

2.1.3 Complexity

There is a consensus among the simulation community that a simple model is mostly preferable to a complex one [28]. However, models of today are growing continuously in complexity due to combined increase in the amount of hardware and software components, forcing modelers to deal with problems that they are not very familiar with. Despite the importance of reducing complexity, it remains at the lowest priority in simulation research agendas. Although complexity is in some sense an intuitive concept, there is no general definition or a single accepted definition of complexity when applied to a model [29]. There are many ways to describe complexity in modeling and simulation [28], in this thesis complexity is defined as a factor which relates to model complexity with the cognitive aspect, i.e. the difficulty of understanding the system being modeled, according to [30].

Quality attributes are considered to be an integral approach to reduce complexity of the system. Apart from those, few other methods are introduced below to deal with highly complex architectures.

2.1.3.1 Abstraction

Abstraction is a technique for hiding complexity of systems under discussion. It works by establishing a level of simplicity on which a developer/user interacts with the system, suppressing the more complex details below the current level. The developers work with an idealized interface (usually well defined) and can add additional levels of functionality that would otherwise be too complex to handle. A simple example is discussed to understand the concept of abstraction better. Adaptive cruise control (ACC) and emergency brake assist (ABS) are being used as shown in Fig. 2.2, the control units need a coordinating module to ensure safe functioning /conflict avoidance with each other. The complexity increases exponentially if these control units are allowed to communicate in various combinations (combinatorial optimization). This heightened complexity can be tackled by adding the motion management for the two systems ACC and ABS that controls the access to the motor and brake control unit. The coordinator modules for engine and brakes are no longer necessary because their tasks are now carried out by the motion management. Above all,

the functions and the actuators only have to use one communications channel, that being the one to the motion management.

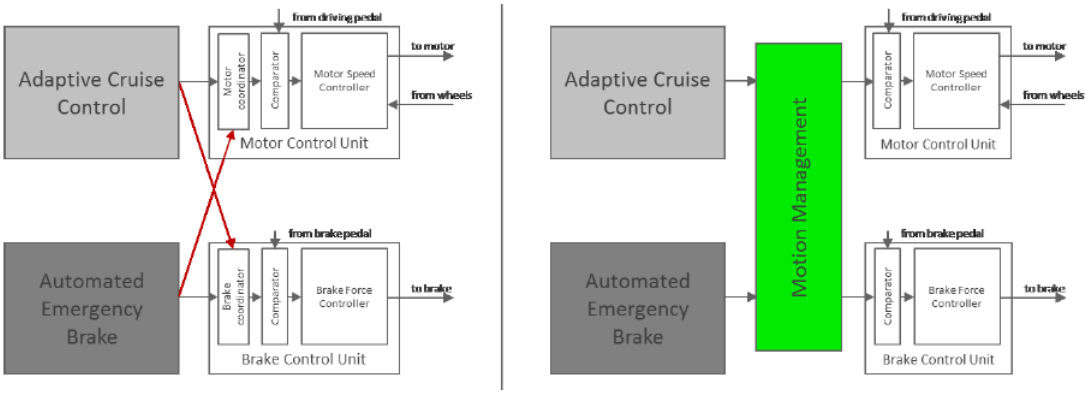

Figure 2.2: Example for abstraction between ACC and ABS to reduce complexity, source: Elektrobit

Abstraction Layer: The process of abstraction discussed above has inspired to develop abstraction layers, Abstraction layer is a way of hiding the implementation details of a particular set of functionality. Abstraction layer, when exploited by system designers enables separation of concerns to facilitate interoperability (between systems, sub systems and components), platform independence (developed software can be run on any vehicle platform without much preparation) and replacement of hardware with software simulation. Layers of abstraction are introduced to support a holistic understanding of the system and its architecture, as well as iterations between the requirements level and a preoccupation with the (overall) integrated system and lower level details of it [31].

While abstraction is a process of hiding complexity of systems, it can also use the below strategies to reduce complexity further.

- **Maintaining simplicity in model design**: While creating a model, the goal should be to pursue simpler models for the purpose it is being created. One way to follow this would be to start with a simple model, validate and analyze the model and when the results are obtained, a short study/discussion can be conducted to decide if the model needs more complexity to be included. If the model is too detailed, then the process of abstraction which initially seemed to simplify the task will complicate the process even further [32]. However, in cases where complex models are already created, simplification effort can be applied to reduce the complexity to some level. One suggested process to transform a given complex model into simpler, generic and more configurable models is *Refactoring*. According to [33], refactoring is "the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure". By refactoring a model it will become well structured and comprehensible[34]. In general, the motivation for refactoring in software development can be transferred to

model implementation: improved comprehensibility of the implementation, facilitated understandability and maintainability of the model code [34].

- **Reduce level of detail and scope of model**: One way of reducing the level of detail in models is by introducing hierarchy. Hierarchy is considered one method of model abstraction [35] and thus can simplify the simulation model. The use of hierarchical modeling can be crucial for the manageability of complex models [36]. It is said to be crucial because it does not always simplify a model. That is because the number of elements in a model that is hierarchically constructed could be the same as in a flat model, with the difference that some of them are hidden by the hierarchy. A simplification could be achieved if it is possible to aggregate some portions of the model (e.g. a set of machines becomes one big machine).

  Another possibility to try to obtain a simpler model (or many, various simpler models) is to attack the scope component of complexity. In this case, divide your system into parts and model each part separately creating a series of simpler models instead of one huge model. Once these parts pass through all phases of the simulation study and if and only if there is a need, integrate these models into a bigger one. But this coarsening in scope can lead to less flexibility of the model [28].

Abstraction can be challenging to achieve in the following ways [37]:

- It is difficult to plan for the new and future versions of models.
- Abstraction should not impact the system performance in terms of available resources and computational time.
- Abstraction needs to be flexible for design alterations without requiring major rework.
- Proprietary abstraction layers significantly limit developer's visibility into the system, if not sufficiently documented, developer is dependent on the abstraction interface. This can restrict design options.

2.1.3.2 Standardization

Companies having a large number of products, built with components/subsystems from a large number of suppliers and selling them to a large number of customers is complex. By standardizing processes and model interfaces, a company will be able to increase its ability to manage this complexity and hence increase its speed and agility in delivering the right product to the right customer. When calling for tenders, the OEM can reference standards, and thereby increasing the level of understanding between the seller and the buyer of a software module. Functions and modules from various manufacturers that is necessary to build a complete system can simply be exchanged, This reduces development costs and risk. Standardizing interface could also result in reduction of profit for suppliers and component developers, as a software module could be sold to several car manufacturers without the need to

produce major changes. Test and tool suppliers can concentrate on the contents of the value creation and not so much on individual interface adaptations[Elektrobit].

Though standardization helps in reducing complexity, it does pose a few limitations [38]:

- Lack of adaption to the different and dynamic automotive market.
- Lack of uniqueness, the modelers cannot act upon opportunities to try different designs and are forced to conform to the standard practice.
- Learning a standard and practising it can be time consuming process.
- Standardization leads to statically defined systems, when revisions are to be made to the systems, backward compatibility should be ensured.

2.1.4 Vehicle modeling architecture VMA

One of the challenges with modeling a vehicle in general is the resource demanding task of building a complete vehicle system one component at a time. To promote the exchange and reusability of automotive subsystem models (e.g. engine, transmission, chassis, etc), the practice of Vehicle Modeling Architecture (VMA) was adapted [6]. The underlying idea behind VMA is that a core vehicle model (including driver and environment), can be defined as modular system, decomposed into several subsystems and components with fixed interfaces between them (discussed in detail in section 2.1.5). The intention of the VMA is to act as a placeholder for models with different levels of fidelity and feature content. As an example of where VMA might prove to be useful, consider the case of an automotive original equipment manufacturer (OEM), within the OEM there are usually different departments working with models and simulation. Each department uses its own set of simulation tools and models, each with their own modeling conventions. A department working on powertrain may have a custom powertrain plant models suited for their testing needs. Another group working on cooling may have their own powertrain plant models for testing. An altogether different department in charge of the control logic has its own powertrain models and so on. None of the departments can easy share or reuse their models between each other without the need of some manual modifications, because they all use different naming conventions, model structure, numbers of ports, and other modeling conventions. VMA basically provides the basis for a standard Modeling architecture with an intention to not constrain the model creators in their way of thinking, but rather to simplify the interconnections between system and subsystems so that a vehicle model can be assembled in a Plug-n-Play fashion using subsystem models from various sources. Achieving a high degree of model reuse is a key benefit of using VMA.

2.1.5 Functional decomposition in modeling architecture

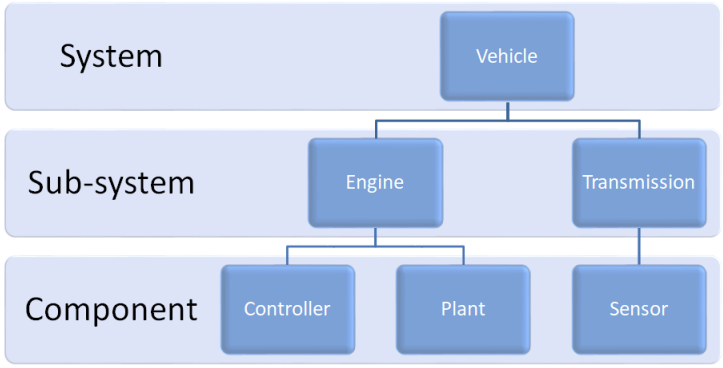

Figure 2.3: Functional decomposition of a system

Functional decomposition is the process of breaking down a complex system into simpler parts, of sub-systems and components as shown in Fig. 2.3. A **System** can be defined as an integrated set of models that accomplish a defined objective (What is to be created) [39]. A **Subsystem** is an extract of the system in its own right, normally the sub-system needs to be integrated with the system or other sub-systems for it to provide useful information [39]. A **Component/Part** is the lowest layer of the architecture, which are models that make up a subsystem or system [39]. The guideline to decompose a system is discussed below:

- The system is first divided into self-controlled modules by their disciplines and the signal interface between these modules is established, this completes the first level of decomposition.
- Each of these modules are further decomposed into sub-systems to reduce complexity and enhance readability, concluding the second level of decomposition.
- The process of decomposition is continued with until the whole system is at the component level, which is the lowest layer of functionality

Creativity, previous experience and analysis is used to compose components/sub-systems into architectures which meet functional requirements of the system under observation. Functional decomposition can pose challenges such as:

- creating unnecessarily numerous, complex signal interfaces.
- It is internally focused, different organizations/departments have different decomposition.
- The decomposition can get too detailed/large easily.
- The hierarchical block structure where the modules are coarse grained can create cross-cutting concerns [40]. A cross-cutting concern is a concern that affects several modules or components, it is directly responsible for tangling or system inter-dependencies. [41]

2.1.6 Evolution in Vehicle modeling architecture

The automotive domain is living an extremely challenging historical moment. Electrification, autonomous driving, and connected cars are some of the driving needs in this changing world. Increasingly, vehicles are becoming software-intensive complex systems and most of the innovation within the automotive industry is based on electronics and software [42]. Modern vehicles can have large number of Electronic Control Units (ECUs) executing gigabytes of software. ECUs are connected to each other through several networks within the car, and the car is increasingly connected with the outside world. These novelties ask for a change on how the software is engineered and produced and for a disruptive renovation of the electrical and software architecture of the car [43]. The architecture of a modern car has to cope with a large amount of concerns, including safety, security, variability management, networking, costs, weight, etc. If not actively managed, architecture and design diverge over time. The architecture is then perceived as outdated and not useful, thus it looses its ability to guide design decisions and implementation. Thus, the VMA has the potential to provide a foundation for managing this so called vehicle evolution if the architecture is reviewed, planned and developed accordingly [44].

2.2 Modeling aspects that aid in Reusability, modularity and scalability to reduce complexity

2.2.1 Physical modeling

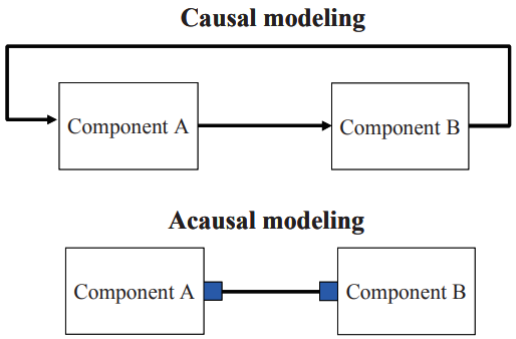

Figure 2.4: Causal and Acausal model signal flow

The traditional approach of modeling the plant model is based on block-oriented schemes with causal relations. The causality is realized artificially to replicate the actual physical scenario and to fulfill appropriate conditions to perform simulations on conventional sequential computers. Fortunately, new concepts which are based on physically oriented connections and algebraic manipulation enable so called acausal modeling. In this section, both approaches are compared and presented.

As shown in Fig. 2.4, an **causal model** is a model that uses the principle of cause and effect to describe the system's behavior. In causal models, the output is

always an integral function of the input, which induces a time delay from input to output. Conversely, an **acausal model** allows data to flow between function blocks in both direction, the inputs and outputs of the system devices are not fixed (i.e., floating inputs/outputs) and the values of both can be chosen based on the association of the device with the other devices. If a system is to be decomposed with causal interactions, a significant amount of effort in terms of analysis and analytic transformations is essential to obtain a desired structure. It also requires a lot of engineering skills and manpower and it is error-prone. In such a system architecture, acausal approach proves to be beneficial. However, in order to allow the reuse of models, the equations should be stated in a neutral form without consideration of computational order which can be done by acausal modeling. The acausal approach can also be applied in multiphysical packages as it has the functionality of both the forward dynamic and backward dynamic solutions by symbolic manipulation (further explained in 2.2.2). This avoids rearranging the model when switching between forward and backward dynamic models, thereby reducing cognitive complexity.

Both causal and acausal modeling approaches prove to be useful in specific scenarios discussed below [45]:

Causal models can be used in following scenarios

- Controller modeling: Intuitive way to model controllers
- Forced insight: process of formulating the equations can yield insight into how the system works
- Familiarity: its is a friendly and familiar way of modeling
- legacy models: many organizations have already built their models in causal approach and its too expensive for them to switch.

Limitations of causal models

- Enhancements get a lot more complicated for complex models.
- As complexity increases, so does the chance of errors.
- Modifications are hard to accomplish among several developers.
- prevents high fidelity modeling of larger systems, particularly in plant models.
- Hard to visually understand the purpose of the system.
- Difficult to connect due to pre-defined inputs and outputs.

Acausal models can be used in the following scenarios [45]

- Plant modeling: Intuitive way to model that cannot be applied to controller models.
- Ease of modifying models: changing the model only requires changing the connection

- Visual clarity: diagrams are easier to understand via visual inspection.

Limitations of acausal models

- Not efficient for algorithm based models, for e.g., control algorithms.
- Loss of system insight, since the equations are not derived manually

2.2.1.1 Plant models illustration

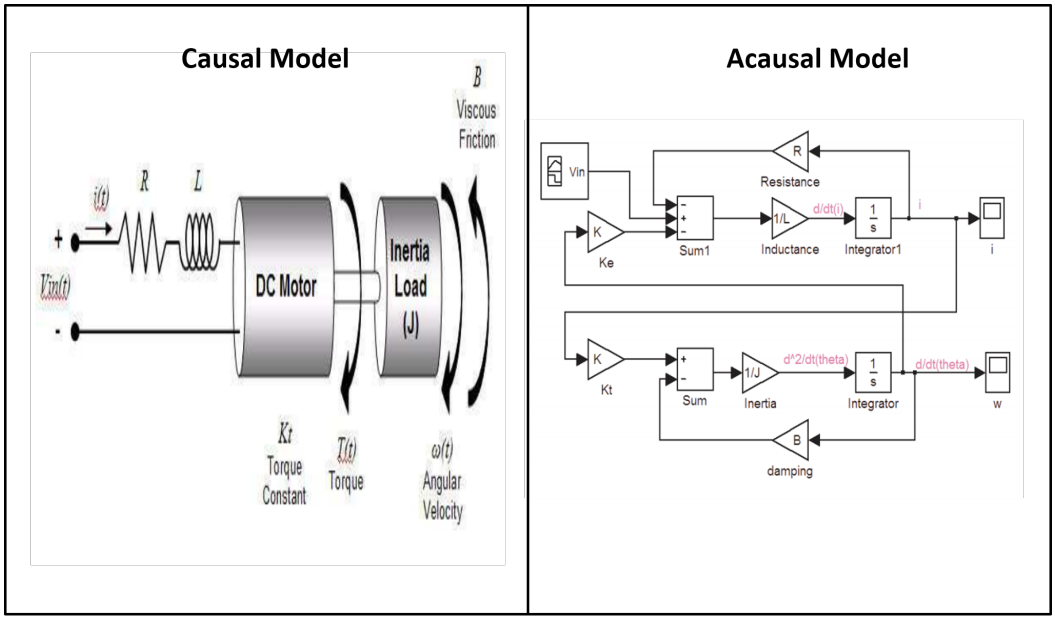

Figure 2.5: example of causal and acausal modeling

Here an example is introduced of a causal model and an acausal model. Fig. 2.5 shows a common representation of DC motor, the left side shows an acausal model of the DC motor whereas the right side shows a causal approach. In the causal modeling approach (right side), the signals flow in connections between individual blocks, transmitting the values of individual variables from the output of one block to the inputs of other blocks. The processing of input information to output information takes place in the blocks. Interconnection of the blocks in Simulink reflects rather the calculation procedure than the actual structure of the model reality. In acausal model (left side), a declarative notation of model is used; this means that individual components of the model (as instances of model classes) are described using equations directly and not the algorithm of their solutions.

It is evident from the above example that as the model gets more complex, the blocks in the causal modeling will also increase rapidly and simulation of such a system might increase the risk of errors relating to algebraic loop, whereas the acausal model is straight forward to understand and also less complex to implement with sufficient knowledge of the model and the tool. When it comes to reuse, it can be seen that acausal models (rather than block diagrams) are easier to reuse because

each component model can be formulated independently without knowledge of the equations or causality assumptions used in other parts of the system.

Having said that, in order to achieve maximum flexibility and reuse, one must identify when to use the acausal features. Block diagrams are preferred for conveying strictly one-way information (e.g., the speed requested of a controller or the current gear of a transmission). Using acausal models in such contexts would be awkward and confusing. In cases where simultaneous equations or conservation principles are used, the acausal approach makes it easier to create and reuse models.

Though, it has been strongly recommended to use acausal models as their benefits are larger than causal models, the fact that causal models have been in use in the industry for long cannot be dismissed. The automotive companies already have many models based on causal modeling and switching their tools requires the transformation from causal to acausal modeling diagrams which can be very drastic and expensive. Both causal and acausal models have benefits and limitations of their own, so instead of limiting ourselves to just one kind of practice why not look into a way that would enable to use both causal and acausal modeling in the same architecture. One possible way of achieving this would be to use two way connections as mentioned in JMAAB plant modeling guidelines [46]. For details regarding two way connections, one can refer to JMAAB plant modeling guidelines as this is out of the scope of this thesis work.

2.2.2 Backward and forward modeling

Depending on the direction of calculation, vehicle models can be classified as either forward or backward facing models [47]. Both forward and backward facing approach may not directly contribute to the analysis of quality attributes, but they aid in simulation and computation speed depending on the use case. In **Backward facing approach**, the computation flows upstream, from the wheels to the prime movers, against the physical power flow. This modeling takes the assumption that the vehicle meets the target performance, and calculates the component states. For instance, the vehicle speed is translated into rotational speed while the traction/braking force is converted to torque, during these conversions the efficiency of some components is assumed. The backward-facing approach proves to advantageous or dis-advantageous based on the use case/test case being modelled [48]. The main advantages of backward-facing approach are:

- Experimental tables are often computed in terms of speed and torque, so that they can be directly implemented without the need for any conversions, which makes it easier to simulate models in QSS.
- Backward-facing approach is beneficial in simplicity and computation cost.
- Reduced execution time.

The major disadvantages in implementing backward-facing or QSS approach are:

- Causes the system to become increasingly complex by predicting the effect of several systems in combination (whether between components or subsystems).
- It is not suitable for "best effort" performance simulations, since it requires the theoretical speed profile to be always perfectly matched.
- Since the energy use is estimated by means of quasi-static steady-state experimental maps, its use does not take into account dynamic effects.
- It is not based directly on relevant control signals for the vehicle, e,g. the throttle position and the brake pedal position.

Where as the **Forward facing approach**/Dynamic simulation is best suited when the purpose is to build a simulation model with realistic and appropriate description of control signals, for control hardware and software development [47]. It simulates the physical behaviors of each component with control. In forward-facing approach the driver provides the accelerator and brake-pedal signals, this means the computational flow proceeds from prime movers through the transmission to the wheels and then the vehicle acceleration is calculated. The advantages of forward-facing approach are:

- Measurable and realistic control signals are used.
- Is advantageous in exploiting performance details.

Forward-facing approach can pose limitations in following scenarios :

- Lower simulation speed caused by the need of integration for vehicle components speed. The components are executed at smaller time steps to provide stability and accuracy for the higher order integration.
- It provides more detailed information of the vehicle system, while at the same time introducing heavy computation consumption.

2.2.3 Top-down and bottom-up approaches

Two of the most common ways to approach system modeling is to either build up the system from components or break the whole system into components. If an engineer starts with a concept, then he/she can figure out what that means or how it fits together by breaking it down (called top-down Modeling). If one starts with behaviour or events, he/she can notice patterns and build up to a model or metaphor (called bottom-up Modeling). Usually the modeller will model from behaviour to pattern (bottom-up) while facilitating the client (for e.g., OEMs) to go from story to components and relationships (top-down). In the current trend of building modular “autonomous vehicle” system, a top-down approach based on the definition of the functional requirements for an autonomous vehicle [49] is adapted. Top-down and bottom-up approaches are discussed according to [50]. **Top-Down approach**: With the Top-Down approach, the Requirements and the System are considered first. Additional abstractions and representations of the system and the concept(s) are modeled (for e.g., a dynamic performance model and a structural analysis model)

in such a way that the simulation results directly address requirements. Ideally, multiple concepts would be developed to maximize the opportunities, to explore the design space and best meet the requirements. The promising concepts should be matured by increasing the fidelity of the models and development the sub-systems to the cascaded requirements. The Top-Down approach typically allows for many more concepts and studies to be conducted with greater flexibility in selecting the appropriate tools for simulation resulting in a faster development cycle while raising the probability that the product meets the performance requirements. **Bottom-up approach**: The Bottom-Up approach starts with much lower-level components being developed and integrated into higher-level sub-system models or assemblies. More components at the lower-levels will need to be created before analysis can be conducted and the simulation results may not address the product's system-level requirements. The bottom-up approach not only requires many more components to be created at lower-levels early in the development process but, the work-flow often encourages premature refinement of those components before their impact on the system-level performance can be made. Bottom-up approaches will work best when the designers and engineers can work together very fast and when the simulations that need to be conducted are either integrated or at least highly associative.

3
Analysis

3.1 Analogy between different state-of-the-art vehicle model architectures

The traditional VMAs [51], [6] and [52] consists of a high level breakdown of a vehicle into key sub-systems trying to mimic the structure of the actual hardware as much as possible. For example, low-level control components are grouped with the hardware-based subsystem they control. Also, only systems that are needed for the intended simulation purpose are represented in the architecture, for instance, there is no need for unused subsystems or its components to be represented if they are empty. The VMA will contain whichever subsystem components that are required to simulate the vehicle under testing, the architecture can either have a flat structure with minimum layers or a hierarchical structure with extensively elaborate layers. Both architectures have their own benefits and limitations depending on the scenario being used in.

In this section, three different VMAs will be discussed, the first one from Ford motor company which is relatively flat architecture based on simulink models, a JMAAB architecture which focuses on hierarchical structure and finally, a Modelica architecture where the focus is to use acausal approach to structure the VMA.

3.1.1 Ford motor company simulink based VMA

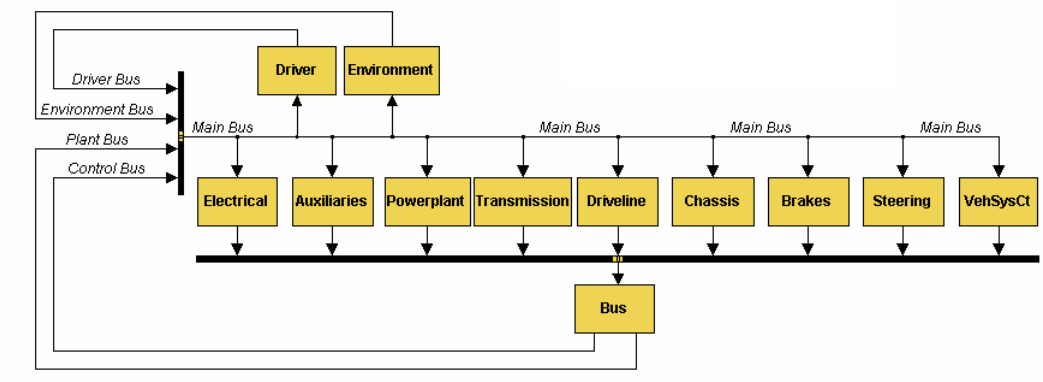

Figure 3.1: Example for the flat vehicle model architecture

At Ford Motor Company [51], a flat structured VMA is customarily used as an important element of an infrastructure to enable a design process for new vehicle systems. The flat vehicle model architecture shown in the Fig. 3.1 consists of three layers where, the top-layer of the vehicle system can contain any combination of the following subsystems: **Driver, Environment, Powerplant, Transmission, Driveline, Chassis, Braking, Steering, Bus**. The connection between the subsystems is achieved via a single main bus that incorporates four sub-buses namely, plant, control, driver and environment bus. The second layer as shown in Fig. 3.2 consists of more detailed sub-systems where one single block handles the interface to the main bus. A bus selector is used to select the required bus from Driver, Environment, Plant and Controller buses for the block. A bus creator is used to produce the plant and controller buses, which are further combined into a bus specific for the subsystem.

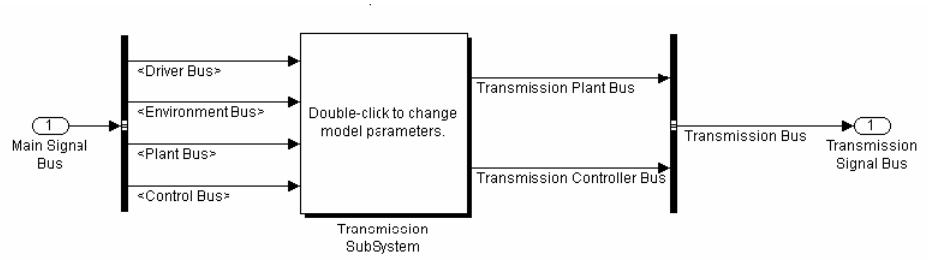

Figure 3.2: Layer2 of the vehicle modeling architecture

Going further down into the third layer, the granularity of the model increases by breaking down the sub-system into components of fine detail. The third layer includes component blocks controller, actuator, plant and sensor (CAPS) to represent local subsystem behavior.

Current design of CAPS subsystem The system to be controlled is called the **plant**; a **sensor** measures the quantity to be controlled; an **actuator** affects the plant; the **controller** processes the sensor signal to drive the actuator; the control law or control algorithm is the algorithm used by the control processor to derive the actuator signal.

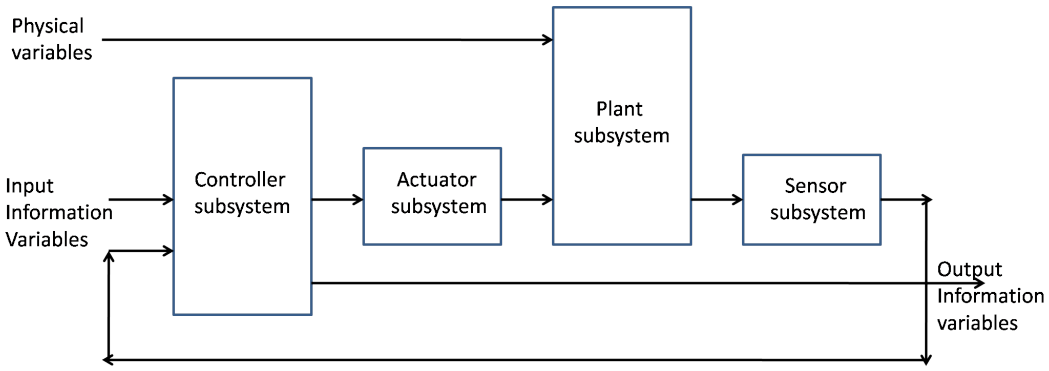

Figure 3.3: CAPS system, often the sensors and actuators are not shown separately

3.1.2 JMAAB hierarchical VMA

The Japan MATLAB Automotive Advisory Board (JMAAB) [52], talks about a hierarchical architecture, where the components are partitioned based on the actual vehicle parts arrangement whenever possible. To define the hierarchy, alternatively one layer is dedicated for component partitioning and another layer for the pair of a controller and a plant.

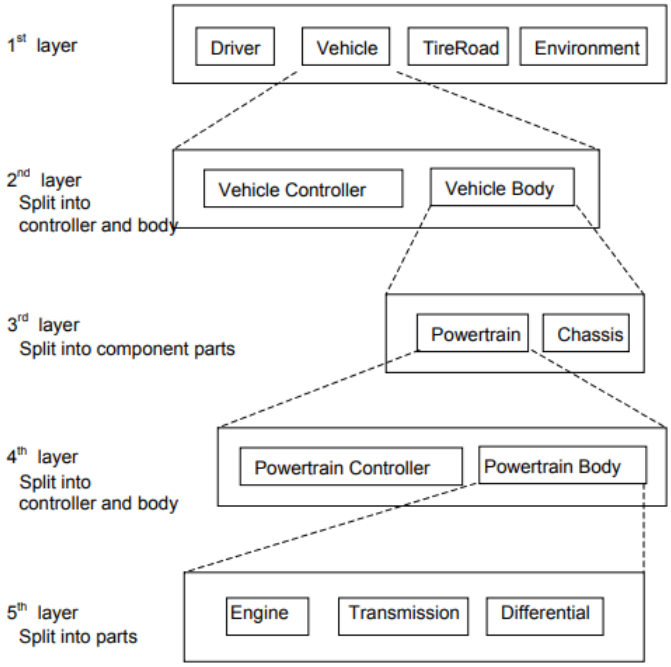

Figure 3.4: Example for the hierarchical vehicle model architecture

The hierarchical architecture shown in Fig. 3.4, consists of multiple layers and the number of layers increase with increase in fidelity of a particular model. The prime difference here is that the vehicle consists of vehicle body and controller and each sub-system also has a separate controller block of it's own. The idea behind this is that, sometimes due to legacy reasons/limitations in signal interfaces, the controller of two or more sub-systems need to be combined and such combined controllers can be placed in the *vehicle controller* and the specific controllers which only comply to that specific plant model/physical signals can be placed in the respective sub-system controllers. Though the hierarchical structure has substantial layers, it contributes to increase in modularity and readability of the vehicle model from the third party developer perspective. Since the components are segregated in a systematic manner, reuse of the artefacts is possible even at the lowest layers of the architecture.

All systems in either (flat and hierarchical) of the VMAs can be categorized as either a containing system or a terminating system. Containing systems consist of one or more subsystems, the system does not contain any models, its only purpose is to describe the structure of interconnections between systems, subsystems and components. Terminating systems consist of a structure into which models can be inserted and any configuration settings needed to provide inputs or calculate

outputs. The architecture of the terminating system must be as flexible as possible in order to support a wide range of application testing and defined in such a way that it is possible to use models of varying degrees of fidelity [51].

3.1.3 Modelica VMA based on acausal approach

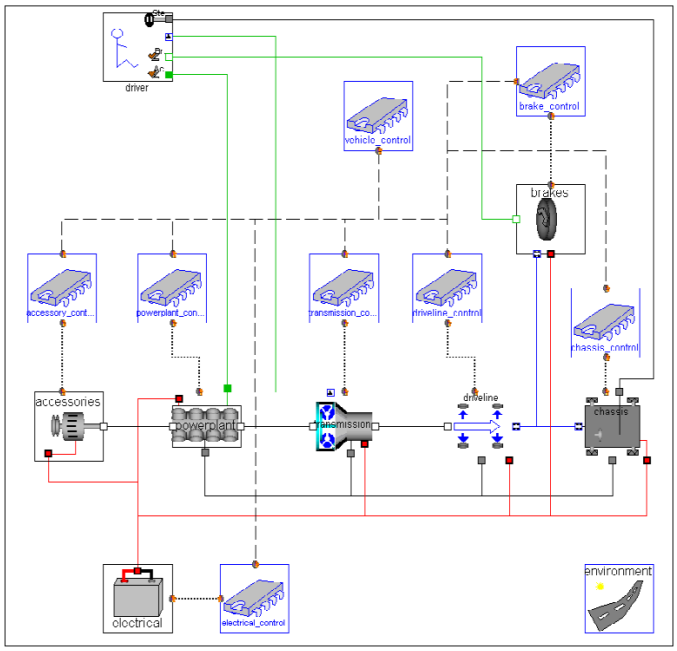

Figure 3.5: Modelica based vehicle modeling architecture

In Modelica architecture [6], a combination of acausal approach (formulating physical connections) and sets of standard interface definitions is adapted from various engineering domains to formulate a vehicle modeling architecture. The architecture uses essentially the same subsystem decomposition, as was done in previous Simulink related architectures, but avoids a priori causality assumptions. The decomposition is shown in Fig. 3.5. A complete vehicle system model takes into account the response of the various physical subsystems, the function of the controller modules (both subsystem and vehicle level) as well as other "external" influences like the environment and the driver. Decomposition of the system is briefly discussed below:

- **Physical Subsystems**: The first decomposition consists of all the major physical subsystems in the vehicle like *Accessories*: which composes of components connected to front end accessory drive(FEAD) of an engine is usually connected to the front side of powerplant.*Electrical*: composes of various purely electrical components in the vehicle. *Powerplant*: represents the primary source of torque for the vehicle, there are physical connections from the powerplant to the accessories and the transmission. The powerplant is also connected to the electrical subsystem. *Transmission*: represents any "gearing" done to deliver power from the powerplant to the wheels. One side of the transmission is connected to the powerplant while the other side is connected

to the driveline. *Driveline*: which is responsible for modeling the distribution of transmission output torque to each of the wheels. *Brakes*: The brake subsystem is physically connected to each wheel, the electrical subsystem and the brake pedal (associated with the driver). *Chassis*: The chassis subsystem represents the vehicle body, frame, wheels and suspension system. Physically, the chassis system is also connected to the electrical system and the steering wheel.

- **Controllers**: Two different control strategies are used in this practice, a simple open loop control strategy for individual plants and controllers and a closed-loop control to capture communication between each subsystem plant and controller pair as well as physical interactions across the various physical subsystems at vehicle level models. The controller decomposition is along the same lines as for physical sub systems decomposition discussed above, *Vehicle system controller*: This vehicle architecture includes a hierarchy of controllers. At the top of this hierarchy is the vehicle system controller. It communicates with each of the subsystem controllers on the vehicle. *Sub system controllers*: These are associated with each physical sub system, these controllers are responsible for controlling the function of their particular subsystem. Each subsystem controller communicates with its associated physical subsystem to exchange sensor and actuator information. In addition, each subsystem may receive supervisory commands from a vehicle system controller.
- **External Influences**: Apart from the physical subsystems and controllers, a vehicle system model must account for two important external influences. The first influence is the driver and the other external influence is the environment. The driver has tremendous response over the vehicle and the environment could potentially influence things like air temperature, road surface effects, obstacles or other vehicles etc.

3.1.4 Analogy of the above mentioned VMAs

The similarities and differences between the above mentioned VMAs is presented here based on the following attributes: **Structuring of the three VMAs** At the top level, all the three architectures are noticed to follow the same vehicle decomposition of Driver, Environment and Vehicle. This decomposition can change at the lower levels of hierarchy in individual systems but at the top layer usually the organizations try to maintain this standard decomposition to accommodate the vehicle industry standards by mimicking the real vehicle and signal interfacing.

Layers of hierarchy The JMAAB, Ford and Modelica VMAs have different hierarchies, but depending on how detailed the system is expected to be, the hierarchical layers can be extended. In cases where the system wants to be maintained simple the existing hierarchical layers can also be removed by combining the common functions. However, it is important to note that the way of creating or removing these hierarchical layers is similar in all the three VMAs.

Signal interfacing between subsystems and components In the Ford and JMAAB VMA, the signal interfacing between the sub system and components is carried out through BUS structure, which are modelled in Simulink. Whereas, in Modelica VMA, the interfacing is accomplished by acausal modeling where a bi-directional physical connection is established between the subsystems and components with no necessity for signal naming convention.

Hardware-Software coupling Ford VMA uses CAPS to describe the local subsystem behaviour, in the sense that it has dedicated controllers for each plant models. While JMAAB and Modelica also have the recommendation to use CAPS system with dedicated controllers to describe the behaviour of subsystems, one clear distinction between them is the additional recommendation to include a supervisory control that aids in supervising the dedicated controllers at respective subsystem levels.

Ease of use and complexity The Ford and JMAAB VMAs are straightforward to understand when the system under discussion is simpler, as the system grows in size, it can get difficult to understand the hierarchies and model functioning, due to the explicit naming convention, hierarchy, dependency from other subsystems etc. Whereas in Modelica VMA, complexity is reduced due to acausal modeling which reduces signal dependency between subsystems and components and also is capable of accommodating the changes when the system grows in size.

Degree of reusability in VMAs In Ford VMA discussed above, it is difficult to reuse models due to the structure incorporated if the system is complex, it resembles more of a flat architecture where common functions are grouped and the CAPS models are tightly coupled to each other. In case of simpler models or carefully maintained models, flat architecture proves to be much useful as it provides an option of setting up the communication buses independent of the components and also the hierarchy is not hard-coded in such a system, giving opportunities to the modeler. The same is the case in JMAAB and Modelica VMA, with one difference being that due to detailed levels of hierarchy, each model is very carefully separated and this can sometimes benefit in reusing the models. This minimizes redundant code and/or configuration options which greatly eases maintenance of the models.

In conclusion, Ford VMA has a relatively flat structure as the system grows in size, structuring gets complex, leading to complex signal interfaces. Due to tight coupling of the CAPS models, it can be difficult to decouple hardware components from software components. In JMAAB VMA, the structure is hierarchical where the models are carefully decomposed and decoupled aiding in better reuse opportunities. The hierarchical structure also reduces cognitive complexity and similar to Ford VMA due to causal nature of signal, the signal interfacing gets complicated as the system grows. Finally in the Modelica VMA, acausal modeling approach is used with limited hierarchical structure, due to clear decoupling of modules, reuse is increased, complexity is reduced due to graphical view of the system and better signal interfacing is provided due to bi-directional signal flow(acausal signal). In both JMAAB and Modelica, the controllers are still tightly coupled with the plant but they still offer some level of hardware-software decoupling due to the presence

of supervisory control.

3.2 Dependencies due to current structuring in VMA

This section will include dependencies caused by structuring of models in different ways. The current VMA structure poses some limitations due to the way it is structured. This section will discuss how the structuring causes dependency in hardware and software modules and also at the market side how it causes the supplier to be dependent on the OEM and vice versa.

3.2.1 Hardware-Software dependency

The current VMAs have specific structures they follow and these structures have a predefined hierarchy with placeholder for components. Though these architecture prove to be advantageous in many scenarios, they sometime causes limitations. In a traditional simulation architecture, the flow in the signal is usually from left to right typically in the CAPS system. The flow is from controller to the plant. This usually limits the architecture by coupling the software and the hardware parts of the vehicle. They have to follow a certain input output structure with fixed ports or BUS structuring. If the hardware is exchanged, then the software component needs to be changed if not they might not work independently since the control algorithm is specifically designed for the physical system it needs to control. This acts as a driving force to decouple the hardware from the software to form a decoupled architecture that allows components to remain completely autonomous and unaware of each other. This approach helps in moving vehicles from a platform deeply entwined with vehicle-specific hardware and available only to embedded and specialist software engineers to a platform that is open and flexible to all the stakeholders [53]. Hardware decoupling in our terms is the process of separating the hardware components from software to reduce the direct dependency between them. One approach to achieve this decoupling is by introducing **Abstraction Layers**, discussed in detail in section 3.5. The advantages foreseen with such a decoupling are [54]:

- Limit indefinite investment in hardware components.
- Decoupling provides the flexibility to easily adapt even to changing application requirements

3.2.2 Supplier dependency with OEM

In the near future, there will be a fundamental shift in how vehicles interact with each other and the outside environment. The increasing complexity of components and the use of more advanced technologies for sensors and actuators, wireless communication, and multicore processors pose a major challenge for building next generation vehicle control systems. Both the supplier and OEM need new ways that enable reliable and cost-effective integration of independently developed system components

[55]. However, to achieve this, major changes to the basic architecture structure is necessary. Fortunately, this will not require new and unproven technologies. By starting with mature architectures used in mainframes and applying the same concepts to vehicle systems, it will be possible to concurrently run separately developed hardware and software components on the same platform. This approach democratizes vehicles for software developers, moving vehicles from a platform, deeply entwined with vehicle-specific hardware and available only to embedded and specialist software engineers to a platform as open and flexible as any other cloud computing environment [56]. A few advantages of decoupling hardware from software in automotive system are discussed below [57]:

- Flexibility: OEMs can more easily switch suppliers and update vehicles with new features after sale.
- Far easier to develop software on known, standard operating systems/hardware platforms.
- Faster deployment by eliminating redundant validation and testing of reused software
- Quicker deployment of diverse implementations of functionalities—e.g., fault-tolerant versions
- Improved engineering productivity
- Minimized risk of bringing updated features to market
- Proven application software can be reused

The disadvantages of decoupling the hardware from software are:

- Suppliers fear losing business
- Lack of standard interfaces for hardware
- Added cost of computing headroom
- Need to better understand hardware virtualization
- Most supplier road-maps don't yet comprehend hardware-software separation
- Standard operating system for central computers is needed

3.3 Motivation for change in current architecture

The complexity of modern vehicles has grown to levels that can hardly be handled efficiently with today's development practices. Yet another increase in complexity can be seen with the trend toward new technologies such as hybrids and electric drives. Although several architectural initiatives have been undertaken [AUTOSAR (www.autosar.org) being a prominent example] an overall architectural concept is still missing which would provide an umbrella for the description of the entire vehicle

system across all functional and engineering domains [58]. There is always a gap between how the architecture is perceived to be and how it actually turns out to be. It was identified in [43] that there is not always an obvious connection between architecture (or top-level design) and design requirements; it seems also that this connection vanishes over time, once development requires changes on the system design. The architecture is communicated as large documents, which are supposed to be read by stakeholders. However, this does not always corresponds to the reality. Automotive companies are competing in their race to develop new technologies and software companies are competing with the vehicle manufacturers. In this race of coping with the advancement of technology, the vehicle is slowly turning into a super computer with more and more features and connectivity, due to this the foundation or the architecture of the vehicle needs constant refinement to be up to date with the software components that are included in it. The change is needed to support the growth in content and complexity

Change in modeling approach: The VMAs discussed in section 3.1, follow different approaches (top down or bottom up) based on the use case and end user choice. Top-down approach is considered to be efficient in structuring architectures [49], however with the increase in software components in vehicle system , it can be seen at the lowest level of hierarchy that the number of interconnecting sub-systems and components increase drastically. This will eventually lead to a state where if a certain module needs to be changed then the interconnecting modules also needs to be changed due to signal interfacing dependency or similar. This is because the module is tightly coupled with its interconnecting module. For e.g. in the CAPS module the controller sits tightly together with the plant model that it is controlling, if either of the plant or the controller is replaced/upgraded then it leads to replacement of the other one as well. This is the same as Hardware and Software dependency that is discussed above in section 3.2.1. This limitation can turn out to be a severe bottle neck with the growing components, for the change to occur in this scenario, it has to be an integrated part of recursive discussions withing the simulation community to shed some light on the limitations and agree collectively upon possible solutions.

Towards a more functionally software driven architecture enabling REUSE: The current VMAs comprise of both the hardware and software components in the architecture, looking at the growing trend in automotive industry, software components are going to outnumber the hardware components. In the sense, the vehicle is expected to be comprised of software intensive units and controlled completely or mostly by software components which would enable easy updates over the air. In order to achieve this, a functional system architecture approach is advised where the intention is to focus on restructuring the vehicle to encompass the separation between software and hardware components so that the hardware components don't hinder the reuse, exchange adding new functionality to the software components. The focus should strictly be on hierarchy, functional separation and definition of interfaces to witness desired results.

Separation of concerns: Each vehicle system structure has various layers and these layers in turn have multiple subsystems and controllers. The integration between these layers are very tight, since all the components and subsystems are exchanging the signal flow (for e.g., request and information). In such a scenario, conflicts might arise in situations when a high priority signal is unable to execute or receive information due to some other interfacing dependency. This will hamper the overall performance of the car and might in some cases cause severe consequences. Restructuring the architecture in a way where the priority of the software components are clearly defined and structured so as to not come in conflict with each other will overcome the limitation. This scenario appears to be slightly more challenging as it is difficult to manage the priority of multiple software intensive units. It is a critical case and demands extra consideration and evaluation of all the software units in the system.

Reduce complexity in the system When the vehicle system gets more complex, it implies that the interfacing between the subsystems and components also get complex. This will lead to extended simulation time and also understanding such a complex architecture will consume a lot of time. As discussed before in Section 2.1.3.1, the current complex vehicle system pushes for a change in architecture to resolve this limitation. One way would be by abstracting the information in the architecture to reduce complexity. With the increasing complexity , this abstraction process should be an iterative change and discussion on this needs to be carried out at a regular basis. The discussion on how abstraction helps reduce complexity can be found in Section 2.1.3.1.

Towards a more scalable system. The current software components, though at the top of updated architecture layer are identified with certain limitations and the changes are being looked into. Similarly the new architecture or the refined architecture will soon become obsolete given the rate at which the vehicle systems are being developed right now. This only means that no certain architecture is ideal. Each architecture should be scalable to accommodate future changes and also the engineers behind the architecture should be open to changes in the trend and hence the architecture. The architecture has to be scalable not only within a single vehicle with varying optional systems, but also throughout all product lines ranging from different vehicle sizes, applications etc.

3.4 Approaches to apply other automotive architecture domains to VMAs

All of the above discussed limitations are the current issues that the automotive community is facing as well and that needs immediate attention. An effort is made in this project to shine some light on these limitations and discuss how they are being handled in other fields of automotive industry and if it is even possible to adopt the same ideas to vehicle modeling architectures.

3.4.1 Current discussion in industry to move towards revised architecture

One struggle car makers have when designing the next safe autonomous systems is the question of distributed versus centralized processing. Why is this a concern? Cars are becoming smarter, more connected and ever more automated. The conventional approach is to distribute processing of sensed data around the vehicle and let the information be processed at different layers of architecture. Another topology under consideration by car makers is the centralized processing model where the raw data coming from the sensors and algorithms built to react to sensed data are in the same high computational layer of the architecture.This will be the focus of the thesis, to look into the centralized control VMAs and how the current industry is working towards it.

Reducing number of ECUs in E/E architecture The increasing use of electronic systems in automobiles brings about advantages by decreasing their weight and cost and providing more safety and comfort [59]. In the earlier days of automotive electronics, each new function was implemented as a stand-alone ECU, which is a subsystem composed of a micro-controller and a set of sensors and actuators. This approach quickly proved to be insufficient with the need for functions to be distributed over several ECUs and the need for information exchanges among functions [60]. Therefore, an architecture of integrated electronic systems in an automobile is important to be designed in order to optimize the total function, cost and productivity [61]. Many automotive OEMs implement most of their functionality in form of software systems. Each ECU is a system embedded with software. It's no longer possible to study an ECU as a stand-alone system. The automotive industry is facing the challenge of the rapidly growing significance of software and software-based functionalities. Research has shown that software complexity is a major reason for project delay and cost overrun. AUTOSAR is a very recent international effort to address the issue of complexity management of highly integrated ECUs for future requirements [62]. Multicore processors having multiple processing units are integrated on a single chip, have emerged to be the main computing controllers not only for high-end servers but also for embedded control systems. Using multicore processors, more centralized architecture designs can be adopted for automotive control systems [63].

Trade-off based on ECU layouts in a functional architecture As electrical and electronic content has continued to add functionality, many wire systems in today's automobiles have become bigger, heavier and more complex than ever. The architecture is analyzed for the following metrics

- Control Latency: In control applications, the electronic system typically interacts in a closed loop with the plant and the environment using sensors and actuators. An interesting measure of performance in such a system is given by the end-to-end latency from sensors to actuators. The delay is measured between sensing a change in the environment and the arrival of the corresponding control signal at the actuator function. From experimental results

conducted in [64], it was established that on a centralized architecture it is easier to achieve shorter latencies. The reasons being:

1. Communication over serial busses is generally slower than communication local to an ECU;
2. It is possible to control the relative order of execution of functions within an ECU. Whereas, across ECU's in an asynchronous communication network, there is no way of ensuring sampling alignment across the receiver and transmitter.

In distributed architectures, the lack of synchronization among ECUs leads to scenarios where the sampling alignment is so unfavorable that it introduces one-period delays at each sample.

- Geometric metrics: In the discussion, three geometric attributes are considered: total wire length, number of cut leads and number of ECUs. These three attributes vary considerably according to the chosen architecture and are the greatest cost drivers for assembly and manufacturing costs. From experimental results conducted in [64], due to the locations of the sensors and actuators, a centralized architecture configuration naturally caused a larger wire length since each I/O wire had to be routed all the way back to the target ECU. On the other hand, the decentralized configuration allowed the modeller to route the signal coming from the I/O to the nearest bus connection, hence reducing the total wire length significantly.
- Serial data metrics: In serial data metrics the measure is based on how extensively the architecture uses the serial bus for communication. The comparison is based on: number of signals communicated locally versus the number of messages transmitted over the bus, the amount of data (number of bytes) in those messages, and the bus utilization (as percentage of the total bandwidth used for communication). It was observed in [64] that the distributed architectures use the serial bus much more extensively than the centralized architecture.

The trade-off though is clearly the reduced latency in a centralized architecture while the decentralized architecture supports a geometrically distributed sensor actuator configuration and is generally more flexible.

3.4.2 Changes in relation to quality attributes: Reuse, modularity and scalability to reduce complexity

One among many motivation for radically departing from traditional VMA to the centralized control architecture is complexity reduction. Since the industry is moving to highly automated system, the back end comes into the system providing dynamic information and more sensors are required for doing highly automated driving functions. It will be challenging to manage the complexity that arises from these new functions if the architecture is not changed. The architecture's main benefit is a significant reduction in the amount of software that must be developed and validated. The outcome of the developed architecture is to reuse the basic platform software, and know that the applications that have already been tested are going to work

in the next ECU, even if it is from a different supplier. Of course the final tests will still have to be conducted. To reduce the complexity of electronics in modern vehicle, OEMs are struggling to integrate as many software components as possible into the existing ECU, without degrading the performance of the ECU.

3.5 Abstraction layers

The term Abstraction Layer originates from operating systems, where it is the layer between computer hardware and the OS kernel. The interface between application software and hardware of a highly integrated system leads to a high complexity. That complexity needs to be hidden (or managed) in order to operate that many actuators and sensors precisely for a certain control application. However, for good performance the components must have the most direct access to all actuators and sensors. The challenge here is to provide a high-level hardware abstraction that is convenient for the intended user and still allows high-performance.

From the control designer's view point the Abstraction Layer is the separation of the hardware components from low-level control (i.e. software). In this way, the control layer is analogous to the operating system layer that provides the basic functionality for subsystems/components. The system has following requirements for the implementation of an Abstraction Layer:

- The layer is expected to comply to the system's domain model (vehicle system framework)
- It is expected to be applicable in real-time and enhance or at least maintain efficiency of the architecture
- The focus of the layer is to be on dedicated hardware (protocols, platforms)

The abstraction layer is thought to validate all input values from control and the validation of the hardware state represented mainly by the sensor values. This would be to ensure that all values that enter the control algorithm from the abstraction layer and vice versa, are confirmed to be valid. The abstraction layer is not only a convenient interface to the hardware it is the ONLY interface. This can be exploited by system designers to hide necessary changes of hardware implementation, e.g. sensor measurement principles or the hardware can be replaced by a software simulation. The concept of model based approach that governs the interaction between system components and the concept of an abstraction layer that governs the location of hardware composition, setup, calibration, and validation promise a way of tackling the complexity of highly integrate vehicle system [65]. The other intention of introducing abstraction layer is because it enables hardware and software decoupling, for e.g., if at any point the software has to be updated or changed then it can be done without having to change the hardware components. The abstraction layers are built to be scalable to all the future changes, in the sense that if new hardware is introduced then the abstraction layer is the only one needing an update to include the abstraction between the components, assuming that the hardware being exchanged still functions the same with the old software.

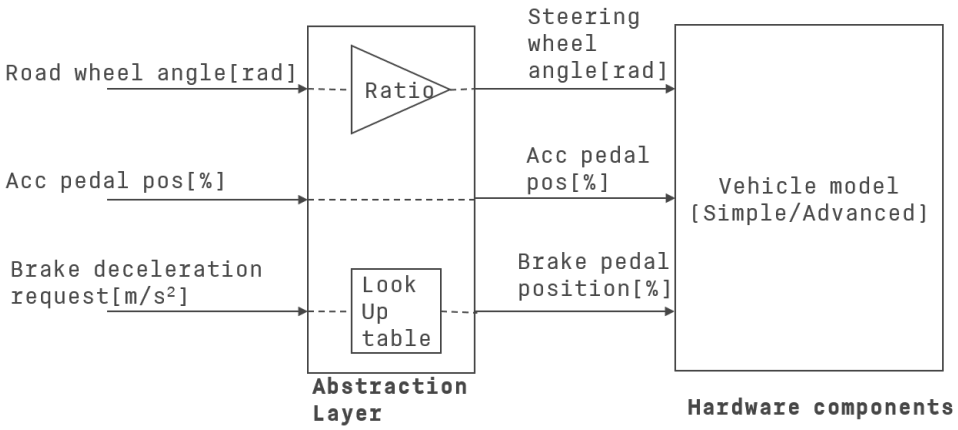

Figure 3.6: Abstraction layer to showcase hardware-software decoupling

Fig. 3.6 shows an example of an example of how abstraction layer aids in hardware and software decoupling [66]. The vehicle model is the hardware component under discussion which takes the following inputs:

- Steering wheel angle[rad]
- Accelerator pedal position[%]
- Brake pedal position[%]

But, the output signals from software are in a slightly different format:

- Road wheel angle[rad]
- Accelerator pedal position[%]
- Brake deceleration request[m/s2]

If the abstraction layer was not present, then either the software (control algorithm) or the hardware component had to be changed to accommodate these inconsistencies. This shows clear dependency between the software and hardware component. The abstraction layer aids in reducing this dependency by providing a fix to match the signal inconsistencies. Road wheel angle is multiplied by the steering ratio to obtain the steering wheel angle. The mapping between brake deceleration and brake pedal position is done using the look up table. As the vehicle model and software algorithm both accept accelerator pedal position, no change is made to this signal. This is just an example to exhibit the usage of abstraction layer.

4

Results

4.1 Revised Centralized control Vehicle Modeling Architecture

This section will utilize some of the concepts discussed in the previous chapter to try and present a possible improvement to the current vehicle modeling architecture. The purpose of partitioning the architecture in this way is to support models of varying levels of detail, provide common interfaces and system decomposition in order to promote model reuse and to support a wide domain of vehicle engineering activities. The introduced architecture is based on AUTOSAR and other pioneer VMAs, the vehicle architecture is partitioned in a way that tries to mimic the other standards as close as possible so as not to deviate too much from the current community standards but does however try and extend the system to be more re-usable, scalable, modular and less complex. AUTOSAR layered architecture is considered as an inspiration to build the revised architecture, various individual concepts introduced in AUTOSAR are also considered (for ex., abstraction layer) and integrated into one single architecture. The revised vehicle architecture is intended to accommodate the autonomous driving software functions but at the same time also works perfectly with the standard vehicle without advanced software units, this is elaborated in Section 4.1.3.9 after introducing the entire architecture.

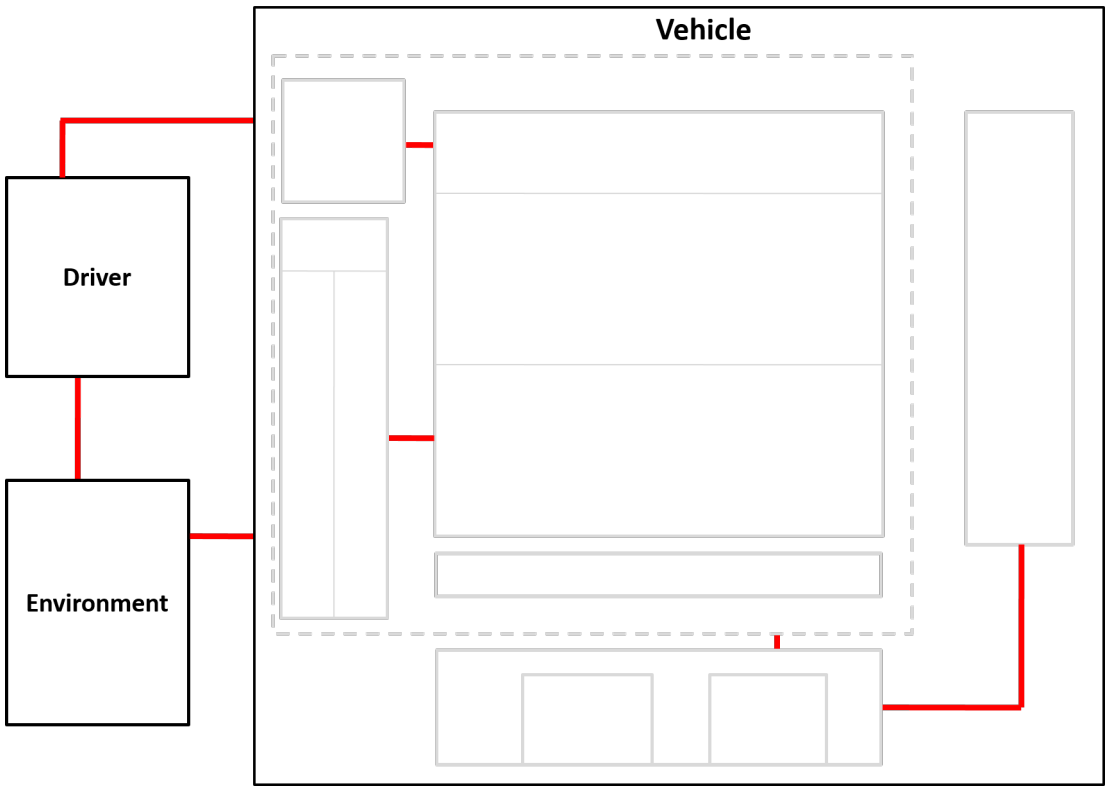

Figure 4.1: Top level of the centralized control vehicle architecture

The top-level of the vehicle architecture describes the system at it's most basic level of functional operation. The functional partitioning of the vehicle consists of: Driver, Environment and Vehicle (control system and physical plant systems) as shown in Fig. 4.1. The red lines in the architecture represent the signal interfacing between the system, subsystem and components in the architecture.

4.1.1 Driver

The first source of excitation comes from driver, it can either be a human driver or a control algorithm that is involved in controlling the vehicle and it's subsystems. The driver or the control algorithm interacts with the Human machine Interface HMI, this could for instance include the sensors/buttons which the driver uses to send information to different parts of the vehicle's embedded motion functionality (i.e, actuators or controllers). The signal interface between the driver and the vehicle subsystems allows for bi-directional interactions. The driver subsystem shown in Fig. 4.2 includes human machine interface (HMI) and the Driver management (signal interpretation and feedback control) placed in the vehicle subsystem. Also, sensing of vehicle's state and environment can further improve the interpretation of driver's intention, and/or to correct a control error.

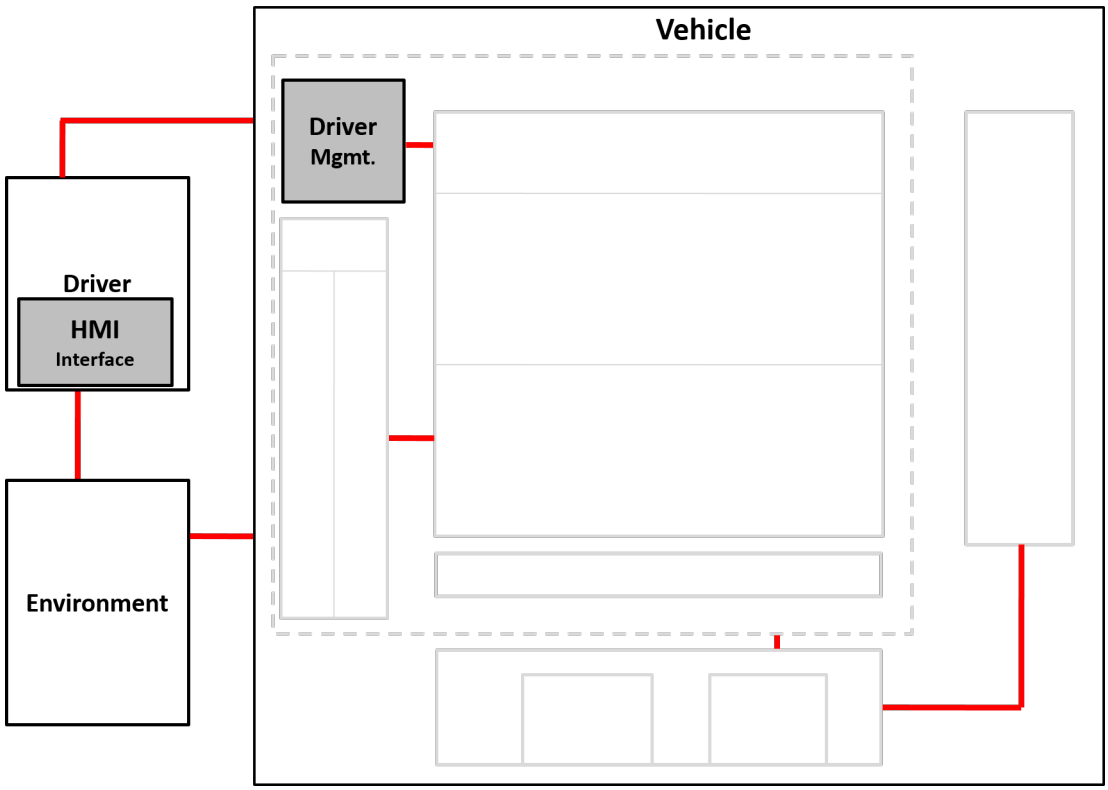

Figure 4.2: Driver subsystem in centralized control vehicle architecture

4.1.2 Environment

The Environment Subsystem defines the physical state of the exterior space surrounding the vehicle and driver that affects operation and performance of the vehicle system and subsystems. It provides the vehicle with external inputs dependent on its global position and velocity and provides each vehicle road-wheel with local road topology and condition inputs. It is also capable of providing continuous information on ambient air conditions. The environment subsystem communicates with other vehicles (V2V) and infrastructure (V2I) and map information. Inputs to environment model is the vehicles position, including orientation in global coordinates. Outputs are the relative position to each obstacle, in vehicle coordinate system.

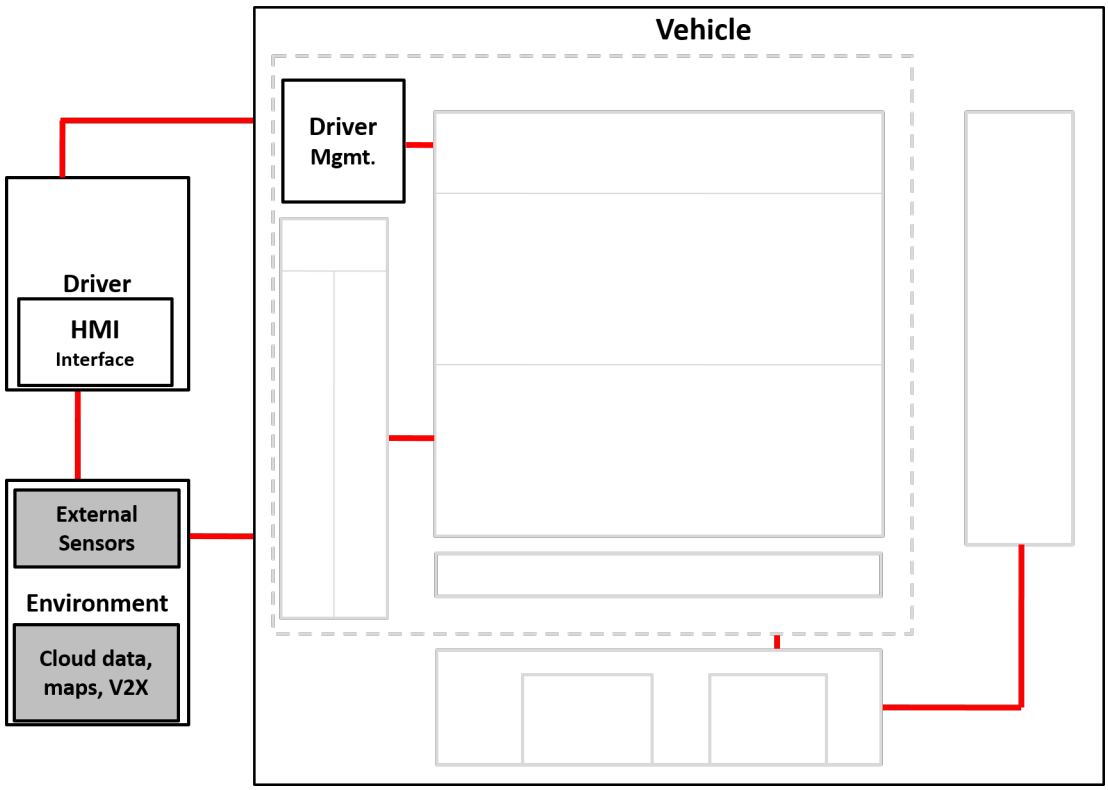

Figure 4.3: Environment subsystem in centralized control vehicle architecture

The surrounding physical state of the vehicle can be described in terms of the conditions of atmosphere, road/terrain, and traffic/surroundings of the vehicle [67].

- **Atmosphere**: The Atmospheric Subsystem defines the physical state (for e.g., atmospheric conditions) of the atmosphere surrounding the vehicle and driver that affects operation and performance of the vehicle system and subsystems.
- **Road/Terrain**: The Road/Terrain Subsystem defines the physical state of the ground surface at the contact patches of the wheels/tracks. It describes the road/terrain conditions (e.g., surface coefficient of friction, surface slope or gradient (surface pitch and roll), surface geometry, etc.).
- **Traffic**: The Traffic Subsystem defines the local traffic around the vehicle. It describes the conditions (e.g., vehicle dynamics, position, velocity, and volumes/footprints of surrounding vehicles, pedestrians, other occupied spaces, and unoccupied spaces, etc.), the vehicle dynamic states of the vehicle can be described in both local and global coordinate systems.
- **Surroundings**: The Surroundings Subsystem defines the infrastructure surrounding the vehicle. It describes surroundings conditions of the infrastructure (e.g., traffic lane configuration, lane width, clearance height, speed limits, traffic lights, etc.)

4.1.3 Vehicle

The Vehicle Subsystem at the most basic and abstract level of purpose and function is partitioned into the following subsystems: **Sensor abstraction, Centralized control, Actuator abstraction, Actuator devices, Internal sensors and Plant models**. Together they represent a conceptual breakdown of the vehicle according to key functions.

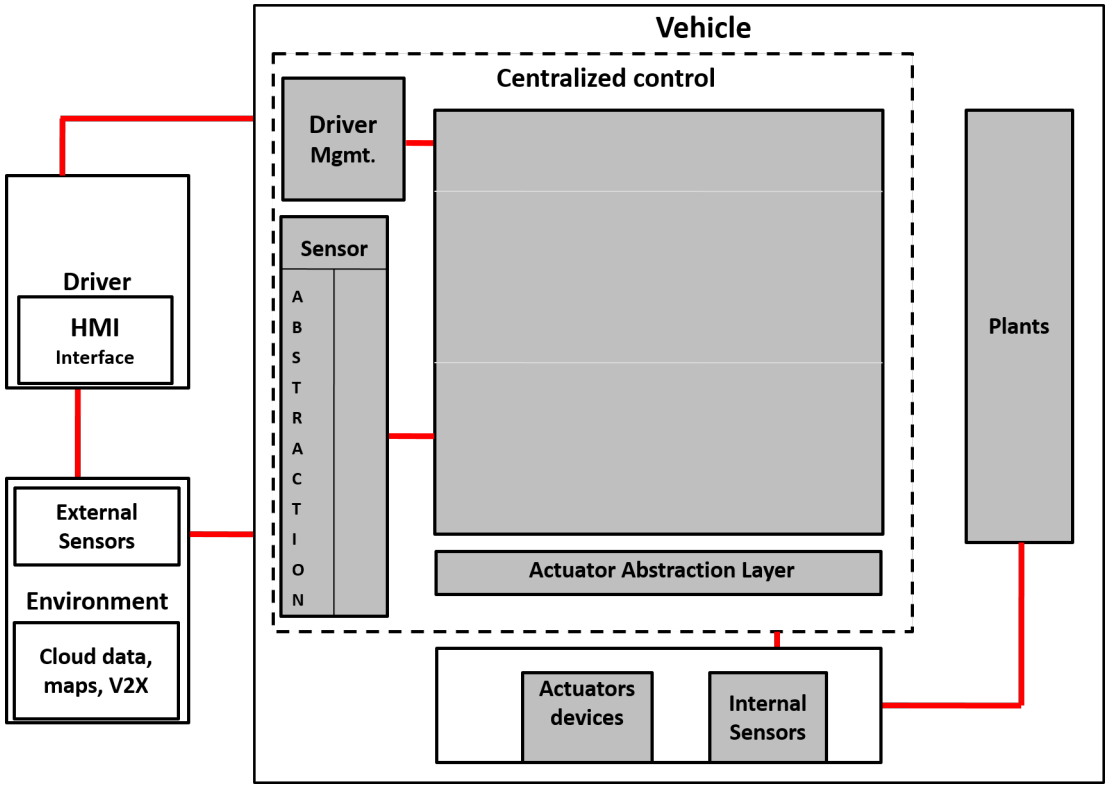

Figure 4.4: Vehicle subsystem in centralized control vehicle architecture

4.1.3.1 Sensors

High specification modern vehicles have environment sensors (camera, radar, GPS with electronic map, etc.) that can give information (relative distance and speed, etc.) about objects ahead of subject vehicle. It can be both fixed objects (road edges, curves, hills, ...) and moving objects (other road users, animals, ...). Based on the positioning of sensors and the information it provides, the sensors can be classified into either Internal or External sensors:

- **Internal sensors**: The internal sensors focus on the dynamic states of the car and it's internal data. These sensors are positioned at the car and focus on vehicle itself. Typical representatives are gyros, accelerometers, steering angle sensors as well as sensors for wiper activity, indicators and further more.
- **External sensors**: The external sensors focus on the car's surrounding environment. These sensors are mounted on the car but record the immediate

environment. Typical representatives are radars, lasers, ultrasonic sensors, cameras and further more.

Apart from these two, there are also meta sensors which are not the actual sensor but rather source of data derived from measurements of other sensors. Typical representatives are cloud data, navigation maps, Car2X and further more. This meta sensor information is included as a part of environment.

It can be seen in the Fig. 4.5, that the internal and external sensors are positioned separately from each other. The reason being, Internal sensors are considered to be carrying critical information about the vehicle and it's states for e.g., the wheel speed etc, therefore it is positioned closely to the centralized control unit in order to have a low latency, internal sensor data is abstracted by actuator abstraction layer. Where as external sensors consist of raw data that is be abstracted before it is fed to the control algorithms and also it is the external sensors that allow fusing of one or more types, which is why the external sensor data has to pass through the sensor abstraction and sensor fusion layers before entering the centralized control.

Individual shortcomings of each sensor type cannot be overcome by just using the same sensor type multiple times. Instead, it requires combining the information coming from different types of sensors, which is why the concept of **Sensor Fusion** is adapted. In Fig. 4.5, the sensor fusion layer is introduced for the exact reason, it is placed in close proximity to the abstraction layer and external sensors, since it is a common practice to combine external sensor types. Sensor fusion takes inputs from different sensors and sensor types and use the combined information to perceive the environment more accurately. That results in better and safer decisions than independent systems could do. An example is presented to better understand the concept of sensor fusion: for e.g., radar might not have the resolution of light-based sensors, but it is great for measuring distances and piercing through rain, snow and fog. These conditions or the absence of light do not work well for a camera, but it can see color (street signs and road markings) and has a great resolution. Radar and camera are examples of how two sensor technologies can complement each other very well. In this way a fused system can do more than the sum of its independent systems could. Sensor fusion also helps in maintaining some basic functionality in case of emergencies or malfunctions.

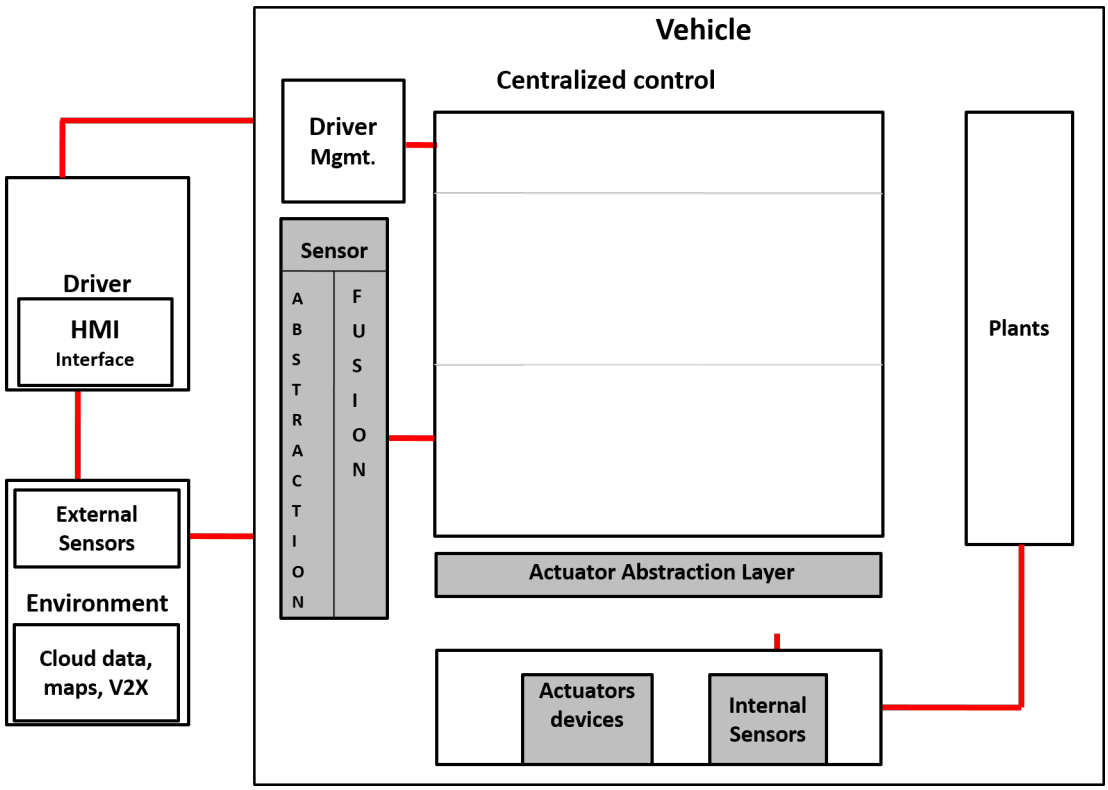

Figure 4.5: Sensors in centralized control vehicle architecture

4.1.3.2 Actuators

Actuators are an essential part of electronic control systems in vehicles. It is their job to convert the electrical signals from the control unit into an action. In Fig. 4.5, the actuator devices are seen to be placed in direct connection to the plant models since they receive command signals from the controller and implements the action on the physical plant models accordingly.

4.1.3.3 Sensor and Actuator Abstraction layer

The abstraction layer introduced in Section. 3.5, is integrated as a part of the centralized control vehicle architecture for sensors and actuators. The abstraction layer enables the ability to change sensors and actuators without re implementing the control model. The sensor abstraction layer on the left side addresses the task of transforming individual sensor data into a format that's understood by the centralized control, the transformation can be simple in some cases like a simple conversion of acceleration from the unit [g] to [m/s2] ans complex in other cases depending on the type of sensor that is being processed.

The actuator abstraction layer addresses the task of transforming the control signals into the format easily understood by the actuator devices, the transformation can be as simple as a unit conversion of an angle from [rad] to [degree] or complex in some cases for e.g., vehicle dynamic influencing actuators. The sensor and actuator abstraction layers have the following advantages:

- It simplifies and allows the architecture to be more "portable" by removing (hardware and software) dependency.
- With emerging XML standards, the abstraction layer guarantees compatibility and interoperability with future technologies.
- It establishes a two-way communication channel with sensors and actuators so sensor/actuator-specific commands can be sent and their results collected and passed up the hierarchy.
- It enhances sensor/actuator capabilities by providing transparent support for any sensor/actuator-specific commands.

4.1.3.4 Centralized control

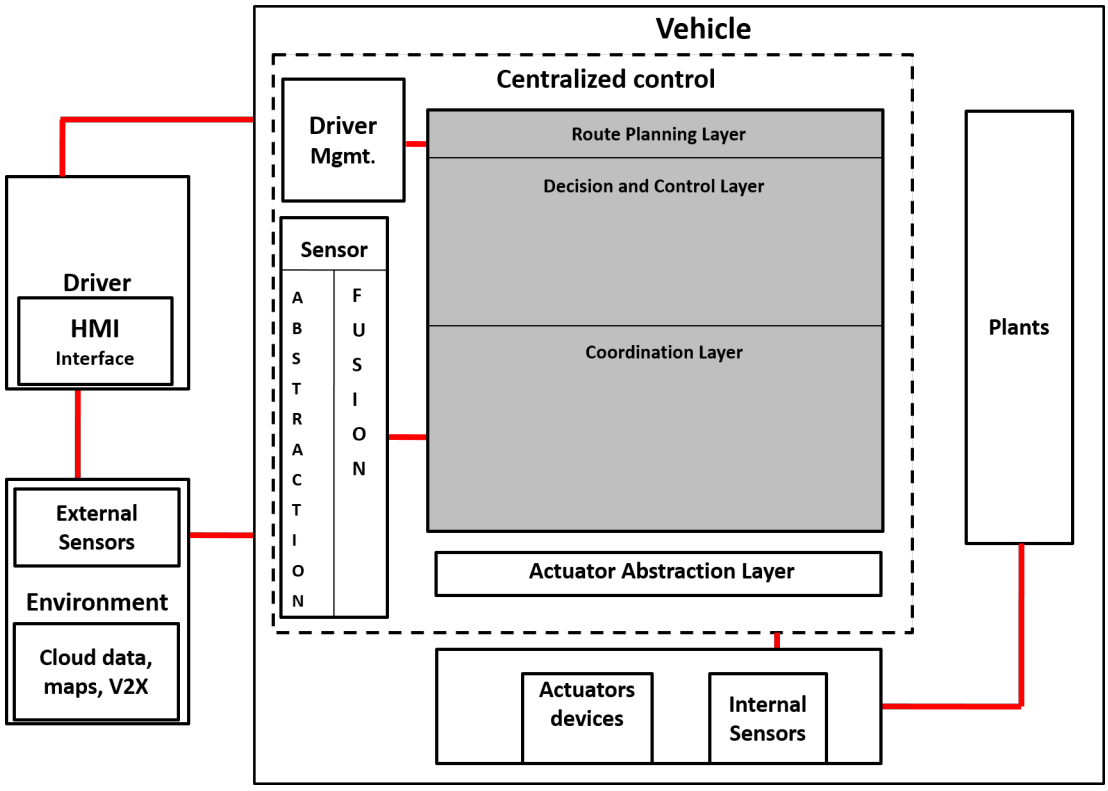

Figure 4.6: Control system in centralized control vehicle architecture

The centralized control system presented in Fig. 4.6 comprises of three different layers. All layers work in a sense-plan-act loop, this loop is expected to be at least an order of magnitude faster than the nominal system loop to execute fast deliberative behaviour in presence of unexpected events. The first layer from the top is the **route planning layer**, this layer is dedicated to strategic functions that involve planning of trajectories or the motion of the vehicle in discussion from point A to point B, this layer is intended to receive information from environment and external sensor

subsystems. The trajectory/route is planned in this layer and then fed to the lower layer for further evaluation of the control signals.

The second layer in the centralized control system is **Decision and control layer**, as the name suggests the main functions that are carried out in this layer is to make decision based on the trajectory planned, decisions are made with the help of control algorithms. Decision and control layer refers to the functional components which are concerned by the vehicle characteristics and behavior in the context of the external environment it is operating in. The control algorithm in the layer contains reactive control algorithms to the unexpected events in the environment, for e.g., collision avoidance by braking etc.

The third layer **Coordination layer**, identifies the current driving situation first, then decide how to coordinate the system's actions based on input from decision and control layer. An operational coordination layer includes events that can generate vehicle motion by estimating the vehicle states and has the capability to send command signals to the actuator. The most important characteristic for this layer is that it deals with continuous dynamic behaviour of the vehicle as a whole system. It is a layer where a failure can result in significant physical damage or economics losses or even threat to human life. This is the level where most of the controller for the vehicle propulsion system is designed.

4.1.3.5 Functionality distribution and Control hierarchy in centralized control

The control system is separated in different hierarchical layers as discussed in section 4.1.3.4. Each layer can be seen as a functional domain that encapsulates several other control systems internally. The reason behind the partitioning of the layers is the complexity of the vehicle system and the different demands in execution time. At the lowest layer control systems that are time critical are placed while at the higher level the layer consists of functionalists that are not time critical. Higher layers are responsible for the overall goals and objectives of the integrated vehicle system, while lower layers are responsible for solving the resulting sub-problems. The layers are organized in such a way that the most time critical modules or functionalities are placed at the lowest layer, the execution time for functions in coordination layer is around 10ms. Basically, the coordination layer includes time critical functionality that has low computational load and high priority in terms of conflict in signals. As one goes higher up in the hierarchy, the time criticality reduces, decision and control layer takes more time (roughly around 1s) than the functions in the coordination layer since it mainly consist of decision and control algorithms. The top most layer which is route planning is the last when comes to priority and time criticality since it does not pose any severe threats to the vehicle or the person in the vehicle. It takes roughly around 10s to execute and has low priority in terms of conflict resolution or emergencies.
The intention behind this structure of control system is to consider the Decision and control (cognitive driving intelligence) as well as the coordination layer(vehicle motion) as two cooperating, relatively autonomous entities. Neither knows the in-

timate details about the other and the decision layer makes motion demands of the vehicle platform in world coordinates, which the latter makes a best effort to fulfill. The task of the decision layer is to perceive the world and make motion requests in this world, while the task of the coordination layer is to realize the desired motion requests while keeping its own features and limitations in mind. In such an ideal de-coupling, the same decision and control layer should be able to operate over a variety of vehicle motion operations with only minor changes, provided the interface between them remains the same.

4.1.3.6 Signal flow

The request signals are flowing downwards and the actual values that is needed to perform this calculations are flowing upwards. Apart form the request signals and actual value signals, a third kind of signal is being used in the structure which is the capability signal. This signal basically is carrying the capability of the actuator and internal sensors. The purpose of doing this is to make sure at all point of time that the critical devices such as actuators and internal sensors are performing as desired. The capability of devices is measured by basically checking the health of the signal being collected from the devices. A quick preprocessing of data with min/max range can be used to check the capability of the actuator and sensor devices. The capability estimation can be added in the coordination layer as that is in direct communication with the actuator and sensor devices. This can help in early testing of the hardware components during HIL (hardware-in-loop) testing and also to check the validity of the plant models with control algorithms being designed for respective hardware components. Capability estimation of components also reduce the risk of component failure thereby reducing cost. The three signals in discussion can be seen in Fig. 4.7

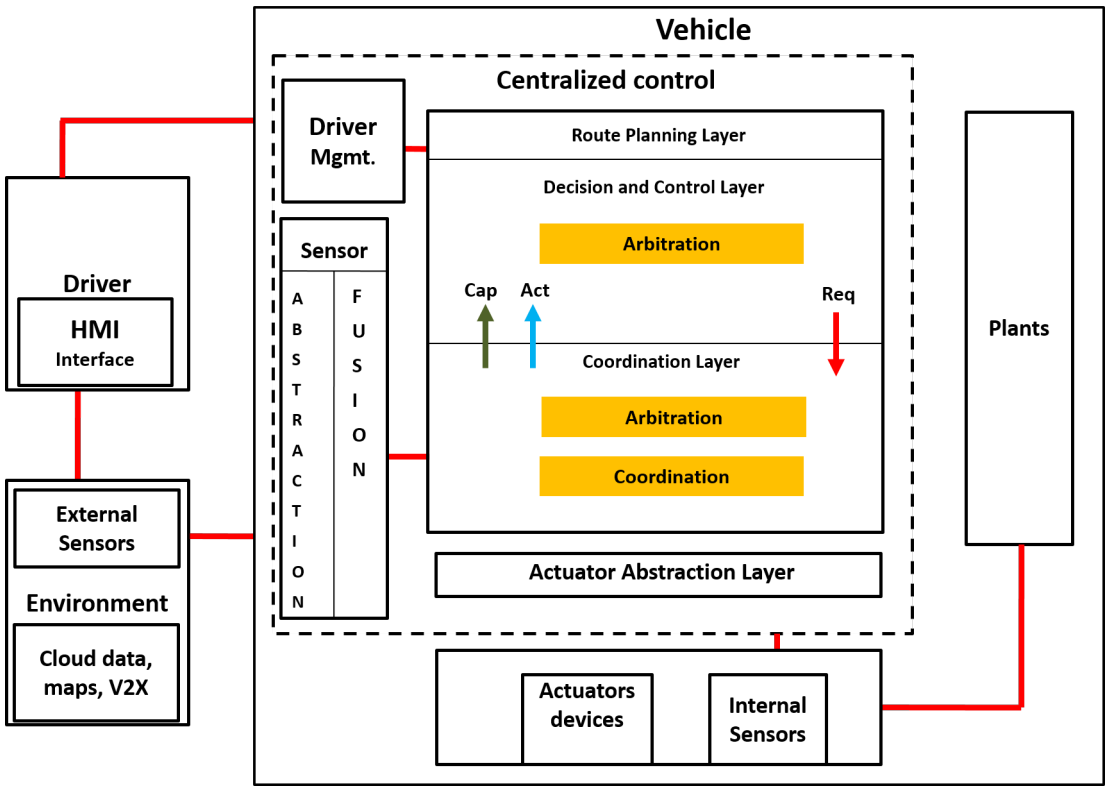

Figure 4.7: Signal flow and conflict resolution in centralized control vehicle architecture

It can be seen that the decision and control layer also include arbitration and coordination units. The Arbitration concept is the process of settling an argument or a disagreement by an entity that is not involved [68]. For e.g, Arbitration allows the selection of different automation modes (for instance if the driver chooses to disable or enable a certain ADAS system) and decides on the control signal that is most suitable to set as command. e.g. lets assume that we are driving on a straight road with Autonomous Emergency Braking (AEB) active. We are closing in to a vehicle which the system deems unsafe, the arbitration unit will either warn the driver or override the drivers controls if a collision is expected. Coordination unit is used as distribution strategy between different requests, e.g. we want to brake in a situation but we have multiple ways of performing that maneuver. Do we use engine-braking, disc brakes or some kind of energy recovery system, the coordination unit will aid in the distribution of the request command to different actuators.

4.1.3.7 Signal flow illustration with ACC functionality

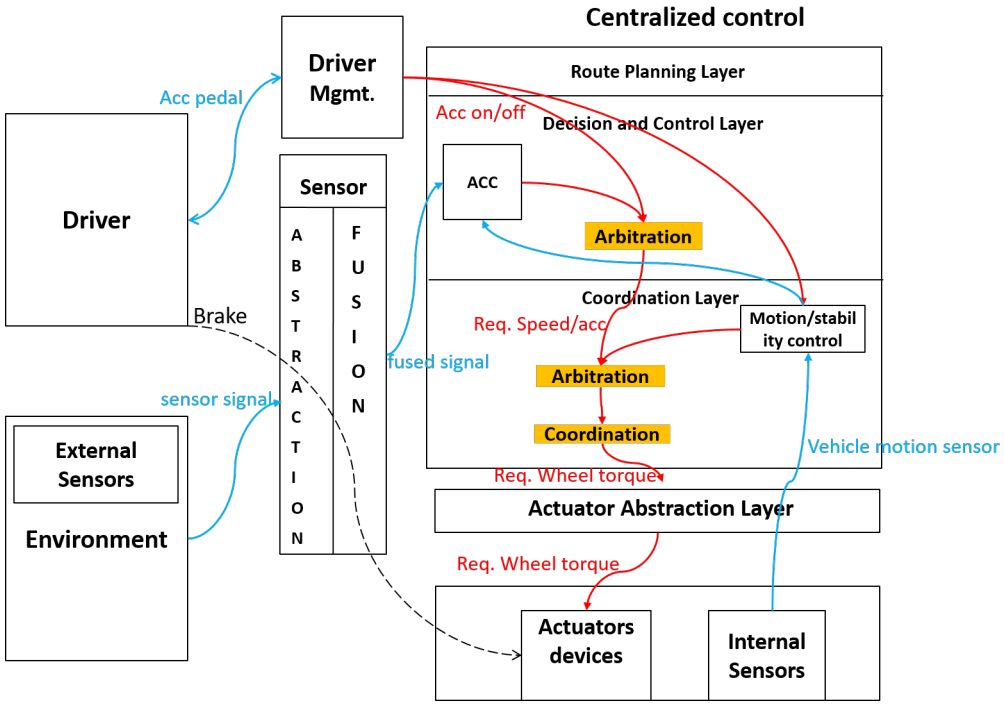

Figure 4.8: example for centralized control vehicle architecture

Fig.4.8 shows an example to better understand the layering and functionality in each layer. Consider an example where the vehicle wants to move from point A to point B with Adaptive cruise control,

- The driver model is used to send out information about the accelerator pedal position and if ACC functionality should be turned on or off.
- The request for ACC functionality service to be activated is sent as a request to decision and control layer.
- The ACC algorithm in decision and control layer receives the actual signal value of distance sensors from external sensors which has been abstracted and possibly fused.
- The arbitration in decision and control compares the ACC algorithm output with that of driver's interpretation and accordingly sends Speed/Acceleration request to coordination layer.
- In coordination layer, the arbitration takes into account requested speed/acceleration from the layer above and compares it with vehicle motion stability control algorithm to send a request to the coordination.
- The coordination will take the request from the arbitration and distributes the powertrain requests to the right components accordingly.

- Coordination layer sends Wheel torque as a request to respective actuator devices through abstraction layer.
- Vehicle motion sensors which are part of internal sensors are sending sensor data to motion/stability control in coordination layer.
- In addition, motion and stability control block estimates the current vehicle states and gives feedback to the ACC algorithm in decision and control layer.

The route planning layer generates the trajectory that the vehicle needs to follow using the information from the environment subsystem and the trajectory (the coordinates in world map) is sent as an output to the decision and control layer. Apart from these functionalities, few other critical functionalities which are proving to be mandatory like safety and diagnostics management and energy management are included in the example. The **safety and diagnostics management** subsystem is expected to perform plausibility checks on the data transferred between modules in the architecture and collect all plausibility check results and derive suitable strategies in terms of safety and error handling. Since the safety subsystems consists a lot of checking and decision making to perform it is placed in decision and control layer where it will have direct access to the data necessary to perform the safety checks. The **Energy management** subsystem consists of electrical and thermal management in the vehicle, these are placed in the coordination layer because they determine the vehicle motion in case of energy critical conditions especially in hybrid and electric vehicles. For e.g., if the vehicle is an hybrid electric then the energy management subsystem is responsible for measuring the state of charge of the battery and in situations when the engine has to take over the energy manager has to step in and send the respective signals, this is a time critical event and can lead to severe consequences which is why it is placed in coordination layer.
It is important to note that this is not a set standard of how the control structure should look like, this is just one way of doing it. For e.g., the number of layers in the control system structure will change depending on the type of vehicle under discussion, if the control system is being designed for a truck then an additional payload management layer can be included above the route planning layer, the addition of payload information in the control structure can yield better results. From control design point-of-view the integrated vehicle control consist of five potentially distinct layers:

- The physical layout of local control based on hardware components, e.g. ABS/EBS, TCS, TRC, suspension.
- Layout of simple control actions, e.g. yaw/roll stability, ride comfort, forward speed.
- The connection layout of information flow from sensors, state estimators, performance outputs, condition monitoring and diagnostics.
- The layout of control algorithms and methodologies with fault-tolerant synthesis, e.g. lane detection and tracking, avoiding obstacles

- Layout of the integrated control design.

This can be used as an inspiration and all or a combination of them can be used based on different use case.

4.1.3.8 Separation of concerns

The functionality distribution in the centralized control system enables a relatively clean separation of concerns. The functionalities in decision and control layer need not be concerned with the finer details of how the motion it desires is achieved, it is only responsible for generating command signals based on the set control algorithms. The vehicle motion in the coordination layer does not need to be concerned with how and why the motion commands are generated - it is only concerned whether they are realizable and if so, how to best realize them given the current platform capabilities. Concepts related to stabilization of the platform, like traction control, anti-lock brakes etc. are transparently realized by the coordination layer, without the decision and control layer having to be aware of them.

4.1.3.9 What is different about this architecture

In the analysis a set of current state of the art architecture are discussed and above in the results, the centralized control vehicle architecture is presented. This section will highlight what are the key differences between the traditional architectures and the centralized control architecture:

- One key difference is the centralizing of the control system into one single module with different layers based on time criticality and reuse, and the unique signal flow between these layers.
- The addition of abstraction layers for sensors and actuators which enables hardware-software decoupling and increased reuse opportunities between the devices and control software
- The segregation of internal and external sensors based on the critical data they output, measurement of capabilities of the critical devices like actuators and internal sensors.
- Including the concept of sensor fusion along with the abstracted external sensor data before feeding it to the control system.
- Separation of concern by decoupling the functionality of the layers in the control system structure. Since the layers are not concerned with how the other layer carries out the task, it creates a clean separation of concern in the architecture.

The Layers/structuring in centralized control architecture has the ability to adapt to the future changes of incorporating autonomous software functions or working seamlessly with the current vehicles with limited autonomous functions. An example is provided to better explain the above claimed difference. In Fig. 4.7, the arbitration in decision and control layer, gets commands from route planning layer

and the driver interpretation based on planned trajectory. In case of autonomous driving, the input to this arbitration can be from the autonomous function and in cases where there is no such input, then the arbitration just considers driver interpretation from an actual driver. The same is applicable to coordination layer as well. The functional distribution among these layers can be decided based on use case.

4.2 Evaluation and comparison of centralized control architecture

The revised architecture with centralized control architecture is evaluated based on the quality attributes of reusability, modularity and scalability and then a brief comparison is done with other existing pioneer vehicle architectures.

4.2.1 Evaluation of centralized control architecture based on quality attributes

In this section, the centralized control architecture is briefly evaluated based on previously discussed quality attributes such as: reusability, modularity, scalability to reduce complexity.

4.2.1.1 Reusability

The centralized control vehicle architecture aids in reusability of components in the following way:

- The separation of concerns in centralized control results in independent functionality distribution. This makes it easier to reuse the control algorithms and other software functionality as they are not tightly coupled with each other. Each module in the centralized control is working as a individual entity.
- The hardware-software decoupling enables reuse of both hardware and software components. For e.g, the plant models can be replaced/changed by still using the same control algorithm or vice-versa. The sensors and actuator units can also be reused. Abstraction is the approach used to decouple hardware-software and enhance reusability.

4.2.1.2 Modularity

Modularity as defined previously, is configuration and reconfiguration of the modules in the architecture. The centralized control architecture is structured in a way to enable plug and play of components. The layers in the control system have specific signal interfacing which makes partitioning and reconfiguration of system even easier. These fixed signal interfaces and abstraction layer helps in decomposing the modules into independent components, these components can then be either used as it is or bunched together to perform desired functionality.

4.2.1.3 Scalability

The centralized control architecture accommodates the current day standard vehicle requirements but also is scalable to future automotive changes. One example for this is the ability to adapt from manual vehicle to autonomous vehicle by help of arbitration in decision and control layer. Also, the layers on the whole can be added or changed depending on use case for e.g., if the centralized control is being designed for trucks then a separate layer can be added for payload management of the vehicle since it plays a major role in the simulation. The abstraction layer makes it further easy to update hardware components with minimum changes to the software units and vice-versa.

4.2.1.4 Complexity

All the above quality attributes can be used to reduce complexity of the system. Enhancing reusability, modularity and scalability in a system leads to reduced functional and cognitive complexity.

4.2.2 Compare with traditional architectures

In this section, we make comparisons with the architectures of ford motor company's VMA, Junior - Stanford's entry in the 2007 DARPA Urban Driving Challenge and the HAVE-IT project. It is useful to compare the proposed architecture with those from other similar ongoing/previous projects. The intent of the comparison is to highlight similarities and differences, rather than make claims of which architecture is "better". The reason being, the system architecture is still largely driven by qualitative aspects like legacy considerations, brand values, organizational and development processes, commitments to specific technology partners and so on [69].

4.2.2.1 Comparison to Ford motor's VMA

A comparison is made to the VMA being used at Ford motor company(2003), though this VMA was presented a long time ago, it is still majorly in practice in the current automotive industry. In the VMA, the primary decomposition of the architecture consists of Driver, environment and vehicle, this has been a standard for decomposing the top layer of architecture, the proposed architecture also follows the same ideology. Going further down into decomposition, the VMA has CAPS system, closely coupling the controller to the plant and also to actuator and sensors to some level to mimic the real vehicle onto the simulation architecture. Whereas in our proposed solution, the controller is decoupled especially from plant models and additionally, the sensors and actuators are in standalone structure with abstraction layers to support exchange and reusability of the hardware and software components. The signal flow in the traditional VMA is only concerned with data transfer, in the proposed architecture an extra step has been taken to consider the signal flow so as to also measure the capabilities of critical sensor and actuator devices. The same comparison also holds good for Modelica architecture, in Modelica architecture, the structure is the same as VMA just the plant models are modelled in an

acausal approach to gain better understanding, reuse and reduced complexity. In the proposal, it is strongly recommended to opt acausal modeling approach but also an alternate option is provided to use the causal models using two way connectors to gain the benefits of acausal plant modeling.

4.2.2.2 Comparison to European HAVEit architecture

Similar comparison has been done with the European HAVEit (Highly automated vehicles for intelligent transport) architecture [70]. This architecture consists of four layers: 'Driver interface', 'Perception', 'Command' and 'Execution'. The Perception layer consists of environmental and vehicle sensors and sensor data fusion. The Command layer contains a component named 'Co-Pilot', which receives the sensor fusion data and generates a candidate trajectory. The selected trajectory is then handed to the Execution layer in the form of a motion control vector. The Execution layer consists of the Drive-train control, which in turn controls the steering, brakes, engine, and gearbox. These layers correspond closely to the route planning, Decision and control and coordination layers in the centralized control of the architecture being proposed in this thesis work. Both the architecture follow Sense-Plan-Act approach where the environmental data is sensed and the desired reaction is planned in cooperation with the control algorithms and finally the action is executed. Also similar usage of a motion control vector as an interface to the vehicle platform/execution layer can be seen in the proposed architecture. Proposed architecture additionally incorporates energy management as an explicit part of the coordination layer, which is especially valuable for electric and hybrid drive-trains and safety and diagnostics as a part of decision and control layer. This comparison highlights the fact that the proposed architecture is in-line with the common thought of the automotive industry.

4.2.2.3 Comparison to Open interface reference architecture

Finally, the proposed architecture is compared to the open interface reference architecture (in close relation to AUTOSAR) by Elektrobit Automotive GmbH. In the open architecture, they create a specification of their own and add vehicle abstractions to convert the specifications back and forth as the data flows from sensor to actuator. In the proposed architecture the abstraction layers are introduced specifically for sensors and actuators to enhance reusability and decoupled way of working in the industry. The idea of separating the internal and external sensors is encouraged by open architecture, where the abstractions on sensor data are based on type clustering of sensor devices into interoceptive, exteroceptive and other sensors. The open architecture is specifically designed for vehicle motion, which is why the control system only includes situative behaviour arbitration and motion management, these two components mainly focus on decomposing the functional behavior, distribution of interpretation (vehicle/driver) and centralization of decision. Though, it is just for vehicle motion, the overall idea still resonates with the proposed architecture where the layers in the centralized control are intending to serve similar purpose. Apart from these similarities, the proposed architecture incorporates safety and di-

agnostics as an integral part of control rather than having a separate unit for it as done in open architecture. The open robinos specification (open reference architecture), is to facilitate common understanding of the architecture and to enable and accelerate the development of highly automated systems across the industry. Open robinos has similar such specification for other automotive features as well and the specification might differ based on use case but the over all architecture still remains the same as the one presented above.

The above comparison has lead us to believe that the explicit separation of concerns, decoupling of hardware and software components and vehicle platform abstraction components are unique to the proposed centralized control architecture. The incorporation of these functions is, to some extent, a deliberate action to resolve the short-comings perceived during early state-of-the-art surveys. Furthermore, the proposed architecture is scalable and can been applied to a larger variety of vehicles (commercial trucks, passenger cars, as well as novel, legacy free designs) and therefore can necessarily incorporate features related to greater isolation of functionality into distinct components and abstraction of vehicle interfaces. The unique partitioning gives the modelers the opportunity to develop and test specific algorithms without having to change or modify the rest of the architecture. The structure also reduces the cognitive complexity of the system and makes it relatively easy to foresee potential pitfalls and debug causes of objectionable behavior.
Having discussed how the proposed architecture works towards fulfilling the current need of automotive industry, it is only reasonable to point out that the centralized control architecture has some limitations of itself too. An architecture needs to be evaluated in its context, because the context imposes unique constraints with associated implications on the design. Thus, it is chosen to believe that every architecture that works has merits in its own context and that there is rarely a definitively best solution to any given architectural problem.

5
Conclusion and future work

5.1 Conclusion

The paper, introduces the concept of modeling and simulation and the benefits these give to realize the desired change in the vehicle industry. To better simulate a real vehicle an architecture has to be adopted, developing this architecture every time a simulation needs to be done, is the same as reinventing the wheel which is why standardization is encouraged in architecture design to enable reuse. The traditional vehicle modeling architectures are introduced and benefits and limitations of the same are discussed, in the process it was noticed that the current architectures are outdated compared to the growing demand of highly automated vehicles. The analysis of current state of the art architectures also showed potential concern in relation to quality attributes such as reusability, modularity and scalability to reduce complexity. A proposal for a centralized control architecture is presented which works on sense-plan-act strategy, by including layers for strategic (route planning), tactical (decision and control) and operation (coordination) functions. The automotive industry and suppliers can rely on this architecture to ease of use in the development and utilization of ECU software. It is needed in order to handle increasing functional complexity in a cost efficient way. At the same time, the fact that there are no uniquely and definitively correct solutions in architecting is acknowledged. It is believed that patterns exist for highly automated vehicle architectures and these patterns ought to be documented and debated. In this paper, an effort has been done to touch upon the same concern by describing the principal concerns of reusability, modularity and scalability and respective proposal to overcome the said concerns, together with some reasoning regarding their distribution across the architecture. A specific architecture incorporating the ideas has been presented in section 4. The architecture highlights the need for a virtual integration environment that allows the architect to take advantage of the architectural degrees of freedom and efficiently analyze the impact of the changes

5.2 Discussion

In this section, research questions are revisited and discussed based on the revised architecture with centralized control.

1. **What are the current state-of-the-art VMAs? How do they differ from each other? What are their limitations?** Three different current state-of-the-art VMAs, ford motor company's simulink based VMA, JMAAB

hierarchical VMA and Modelica's acausal approach based VMA are presented in section 3.1. Later in the same section, all the three architectures are compared to find, Form motor company's VMA is relatively flat in hierarchy with minimal layers and tightly coupled CAPS system. The JMAAB architecture is hierarchical in nature, along with individual controllers coupled to plant models it has a vehicle supervisory control at the top level that supervises other controllers. The Modelica VMA, though hierarchical in nature it follows acausal approach to structure the system. The comparison is mainly done on structuring of VMAs, layers of hierarchy, signal interfacing, hardware-software decoupling, ease of use and complexity of system and finally, degree of reusability in VMAs. The limitations of the three current VMAs are discussed based on quality attributes of how reusable are the model/components? How scalable is the architecture and how modular is the architecture? These quality attributes in-turn help to solve the challenges posed by complexity on the VMAs.

2. **How can the structure of current vehicle modeling architecture change to accommodate growing future automotive functions.** The current VMA though hugely in practise needs changes to it's structure to accommodate future changes in automotive industry. The architecture is analyzed based on the modeling approach and the structure is revised in a direction towards a more functionally software driven architecture to enable higher reuse. It is further analyzed in section 3.3 how separating the concerns in the architecture aids in achieving the results for quality attributes discussed. Later in Chapter 4, a new revised architecture is presented, which involves restructuring of the virtual vehicle to accommodate the future growing needs and also the improvements for the discussed current VMA's limitations.

3. **How does the new revised VMA differ from the current VMAs based on quality attributes such as reusability, modularity and scalability to reduce complexity** Section 4.1.3.9, covers a detailed evaluation and comparison of the revised VMA with centralized control compared to traditional state-of.the-art VMAs. The comparison is done based on quality attributes such as reusability, modularity and scalability to reduce complexity of the architecture. Along with comparison to traditional state-of-the-art architectures, the revised architecture is compared to European HAVEit architecture and open interface reference architecture by Elektrobit(based on AUTOSAR), which have been an integral part of discussion in the automotive community.

5.3 Future scope

The proposed vehicle modeling architecture in this thesis is a high level abstraction and needs further realization and validation by implementing the architecture in vehicle simulations. The outcome of the concepts presented in this architecture is expected to trigger further research initiatives in this challenging area. The system modules at its current state is more focused on vehicle motion control, it could

be of interest to extend the architecture to cover other system modules like driver interfaces and integration with vehicle infotainment. Metadata could also be one potential approach to enable reuse more effectively by including data about the models and how they are supposed to be used correctly. Further investigation is still needed in this area, for e.g., decide what kind of formalization is needed to describe component data, what characteristics of a model should be expressed in the data, increase of effectiveness by developing a standard vocabulary, can metadata be generated from model components and perhaps verified against templates. Furthermore it could also be of interest to investigate if the reuse process can be automated by algorithms and frameworks that select the right component for a certain simulation scenario and confirmed by validation.

Bibliography

[1] P. Gao, H.-W. Kaas, D. Mohr, and D. Wee, "Automotive revolution–perspective towards 2030 how the convergence of disruptive technology-driven trends could transform the auto industry," *Advanced Industries, McKinsey & Company*, 2016.

[2] J. Batteh and M. Tiller, "Implementation of an extended vehicle model architecture in modelica for hybrid vehicle modeling: development and applications," in *Proceedings of the 7th International Modelica Conference; Como; Italy; 20-22 September 2009*, no. 043. Linköping University Electronic Press, 2009, pp. 823–832.

[3] ——, "Implementation of an extended vehicle model architecture in modelica for hybrid vehicle modeling: Development and applications," 10 2009.

[4] V. M. Navale, K. Williams, A. Lagospiris, M. Schaffert, and M.-A. Schweiker, "(r) evolution of e/e architectures," *SAE International Journal of Passenger Cars-Electronic and Electrical Systems*, vol. 8, no. 2015-01-0196, pp. 282–288, 2015.

[5] P. Pelliccione, E. Knauss, R. Heldal, M. Ågren, P. Mallozzi, A. Alminger, and D. Borgentun, "A proposal for an automotive architecture framework for volvo cars," in *2016 Workshop on Automotive Systems/Software Architectures (WASA)*, April 2016, pp. 18–21.

[6] M. Tiller, P. Bowles, and M. Dempsey, "Development of a vehicle model architecture in modelica," in *Paper presented at the 3rd International Modelica Conference*, 2003.

[7] M. Persson and S. Elfström, "Volvo car group's first self-driving autopilot cars test on public roads around gothenburg," *Volvo Car Group Media Relations*, pp. 04–29, 2014.

[8] J. D. Stoll, "Gm executive credits silicon valley for accelerating development of self-driving cars," *The Wall Street Journal*, vol. 10, 2016.

[9] S. Danny, "To be safe in the real world, avs must spend time in a virtual one," 2018.

[10] A. C. Myriam Alexander-Kearns, Miranda Peterson, "The impact of vehicle automation on carbon emissions," 2016.

[11] M. Broy, "Challenges in automotive software engineering," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 33–42.

[12] J. S. Carson, "Introduction to modeling and simulation," in *Proceedings of the 2004 Winter Simulation Conference, 2004.*, vol. 1, Dec 2004, p. 16.

[13] ——, "Introduction to modeling and simulation," in *Proceedings of the 2004 Winter Simulation Conference, 2004.*, vol. 1, Dec 2004, p. 16.

[14] A. Kara, F. Deniz, D. Bozağaç, and N. Alpdemir, "Simulation modeling architecture (sima), a devs based modeling and simulation framework," pp. 315–321, 07 2009.

[15] A. Koziolek, *Automated improvement of software architecture models for performance and other quality attributes.* KIT Scientific Publishing, 2014, vol. 7.

[16] G. Falcone, *Hierarchy-aware software metrics in component composition hierarchies.* Logos Verlag Berlin GmbH, 2010.

[17] R. Reese and D. L. Wyatt, "Software reuse and simulation," in *Proceedings of the 19th conference on Winter simulation.* ACM, 1987, pp. 185–192.

[18] S. Robinson, R. E. Nance, R. J. Paul, M. Pidd, and S. J. Taylor, "Simulation model reuse: definitions, benefits and obstacles," *Simulation modelling practice and theory*, vol. 12, no. 7-8, pp. 479–494, 2004.

[19] R. Fujimoto, C. Bock, W. Chen, E. Page, and J. H. Panchal, *Research challenges in modeling and simulation for engineering complex systems.* Springer, 2017.

[20] J. Colchester, "Modular systems design," 05 2015.

[21] M. Takefumi, "Automotive parts modularization and its challenges for local suppliers in hiroshima region."

[22] R. M. Henderson and K. B. Clark, "Architectural innovation: The reconfiguration of existing product technologies and the failure of established firms," *Administrative science quarterly*, pp. 9–30, 1990.

[23] B. C. ord Neuman, "Scale in distributed systems," *ISI/USC*, 1994.

[24] D. V, "Building blocks of a scalable architecture," January 09, 2018.

[25] D. Hughes-Hallett, A. M. Gleason, D. Flath, P. F. Lock, S. P. Gordon, D. O. Lomen, D. Lovelock, W. G. McCallum, D. Quinney, B. G. Osgood *et al.*, *Calculus: Single Variable.* Wiley, 1998.

[26] W. R. Group, "An introduction to parameterized model reduction, multifidelity modeling, and uncertainty quantification," 2018.

[27] O. TIGGES, "Scalable software systems, from developer laptops to server farms," November 26, 2014.

[28] L. Chwif, M. R. P. Barretto, and R. J. Paul, "On simulation model complexity," in *Proceedings of the 32nd conference on Winter simulation.* Society for Computer Simulation International, 2000, pp. 449–455.

[29] R. Brooks and A. Tobias, "Choosing the best model: Level of detail, complexity, and model performance," vol. 24, pp. 1–14, 08 1996.

[30] M. W. Golay, P. H. Seong, and V. P. Manno, "A measure of the difficulty of system diagnosis and its relationship to complexity," *International Journal Of General System*, vol. 16, no. 1, pp. 1–23, 1989.

[31] M. Broy, M. Gleirscher, P. Kluge, W. Krenzer, S. Merenda, and D. Wild, "Automotive architecture framework: Towards a holistic and standardised system architecture description," 2009.

[32] G. J. Holzmann, "Designing executable abstractions," in *Proceedings of the second workshop on formal methods in software practice.* ACM, 1998, pp. 103–108.

[33] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

[34] C. Triebig and F. Klügl, "Refactoring of agent-based simulation models." in *Multikonferenz Wirtschaftsinformatik*, 2008.

[35] J. J. Luna, "Hierarchical relation in simulation models," in *Proceedings of the 25th conference on Winter simulation*. ACM, 1993, pp. 132–137.

[36] T. Daum and R. G. Sargent, "Scaling, hierarchical modeling, and reuse in an object-oriented modeling and simulation system," in *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future-Volume 2*. ACM, 1999, pp. 1470–1477.

[37] V. Siddha, K. Ishiguro, and G. A. Hernandez, "Hardware abstraction layer," Aug. 28 2012, uS Patent 8,254,285.

[38] S. Martínez-Fernández, C. P. Ayala, X. Franch, and E. Y. Nakagawa, "A survey on the benefits and drawbacks of autosar," in *Proceedings of the First International Workshop on Automotive Software Architecture*. ACM, 2015, pp. 19–26.

[39] A. P. Sage, *System of systems engineering: innovations for the 21st century*. John Wiley & Sons, 2011, vol. 58.

[40] S. Apel, C. Kästner, A. Größlinger, and C. Lengauer, "Feature (de) composition in functional programming," in *International Conference on Software Composition*. Springer, 2009, pp. 9–26.

[41] G. Kamble, "Aop-introduced crosscutting concerns," in *Proceedings of International Symposium on Computing, Communication, and Control (ISCCC 2009)*, 2009.

[42] P. van Staa, "How kets can contribute to the re-industrialisation of europe," in *European Technology Congress, Wroc law, June 12-13, 2014: http://docplayer. net/21724658-Date-2012-how-kets-can-contribute-to-the-re-industrialisation-of-europe. html*, 2014.

[43] P. Pelliccione, E. Knauss, R. Heldal, S. M. Ågren, P. Mallozzi, A. Alminger, and D. Borgentun, "Automotive architecture framework: The experience of volvo cars," *Journal of Systems Architecture*, vol. 77, pp. 83–100, 2017.

[44] O. Barais, A. F. Le Meur, L. Duchien, and J. Lawall, "Software architecture evolution," in *Software Evolution*. Springer, 2008, pp. 233–262.

[45] S. Chad, "Physical modelling," Director of Maplesim development.

[46] A. Ohata and S. Komori, "Jmaab plant modeling guidelines and vehicle architecture," , pp. 484–487, 2009.

[47] R. Trigui, M. Desbois-Renaudin, B. Jeanneret, and F. Badin, "Global forward-backward approach for a systematic analysis and implementation of hybrid vehicle management laws. application to a two clutches parallel hybrid power train." in *EET-2004 European Ele-Drive Conference*, 2004, pp. 12–p.

[48] I. Briggs, M. Murtagh, R. Kee, G. McCulloug, and R. Douglas, "Sustainable non-automotive vehicles: The simulation challenges," *Renewable and Sustainable Energy Reviews*, vol. 68, pp. 840–851, 2017.

[49] R. Matthaei and M. Maurer, "Autonomous driving–a top-down-approach," *at-Automatisierungstechnik*, vol. 63, no. 3, pp. 155–167, 2015.

[50] L. Matthew, "Bottom(s)-up or is the top-down? a working definition of systems engineering," 11 2013.

[51] C. Belton, P. Bennett, P. Burchill, D. Copp, N. Darnton, K. Butts, J. Che, B. Hieb, M. Jennings, and T. Mortimer, "A vehicle model architecture for vehicle system control design," SAE Technical Paper, Tech. Rep., 2003.

[52] A. Ohata and S. Komori, "Jmaab plant modeling guidelines and vehicle architecture," in *2009 ICCAS-SICE*, Aug 2009, pp. 484–487.

[53] E. SYSTEMS, "Decoupling Hardware from Software in the Next Generation of Connected Vehicles," Tech. Rep., 05 2018.

[54] M. Kersting, "Independent and future-proof: decoupling of hardware and software through image abstraction," Tech. Rep., 2015.

[55] I. N. E. EPAM Systems. (2018, May) Decoupling hardware from software in the next generation of connected vehicles. [Online]. Available: https://www.epam.com/content/dam/epam/en/free_library/whitepapers/Automotive-White-Paper.pdf

[56] R. Baheti and H. Gill, "Cyber-physical systems," *The impact of control technology*, vol. 12, no. 1, pp. 161–166, 2011.

[57] P. Hansen, "The hansen report on automotive electronics," 2002.

[58] M. Broy, M. Gleirscher, S. Merenda, D. Wild, P. Kluge, and W. Krenzer, "Toward a holistic and standardized automotive architecture description," *Computer*, vol. 42, no. 12, 2009.

[59] S. Chakraborty and S. Ramesh, "Programming and performance modelling of automotive ecu networks," in *VLSI Design, 2008. VLSID 2008. 21st International Conference on*. IEEE, 2008, pp. 8–9.

[60] N. Navet, Y. Song, F. Simonot-Lion, and C. Wilwert, "Trends in automotive communication systems," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1204–1223, 2005.

[61] Y. Furukawa and S. Kawamura, "Automotive electronics system, software, and local area network," in *Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*. ACM, 2006, pp. 2–2.

[62] G. N. Vo, R. Lai, and M. Garg, "Building automotive software component within the autosar environment-a case study," in *Quality Software, 2009. QSIC'09. 9th International Conference on*. IEEE, 2009, pp. 191–200.

[63] K. Senthilkumar and R. Ramadoss, "Designing multicore ecu architecture in vehicle networks using autosar," in *Advanced Computing (ICoAC), 2011 Third International Conference on*. IEEE, 2011, pp. 270–275.

[64] S. Kanajan, C. Pinello, H. Zeng, and A. Sangiovanni-Vincentelli, "Exploring trade-off's between centralized versus decentralized automotive architectures using a virtual integration environment," in *Design, Automation and Test in Europe, 2006. DATE'06. Proceedings*, vol. 1. IEEE, 2006, pp. 1–6.

[65] S. Jörg, J. Tully, and A. Albu-Schäffer, "The hardware abstraction layer—supporting control design by tackling the complexity of humanoid robot hardware," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 6427–6433.

[66] V. Patil, "Generic and complete vehicle dynamic models for open-source platforms," 2017.

[67] J. S. Albus, "4d/rcs: a reference model architecture for intelligent unmanned ground vehicles," in *Unmanned Ground Vehicle Technology IV*, vol. 4715. International Society for Optics and Photonics, 2002, pp. 303–311.

[68] D. González, J. Pérez, V. Milanés, F. Nashashibi, M. S. Tort, and A. Cuevas, "Arbitration and sharing control strategies in the driving process," *Towards a Common Software/Hardware Methodology for Future Advanced Driver Assistance Systems*, p. 201, 2017.

[69] S. Behere and M. Torngren, "A functional architecture for autonomous driving," in *Automotive Software Architecture (WASA), 2015 First International Workshop on.* IEEE, 2015, pp. 3–10.

[70] R. Hoeger, A. Amditis, M. Kunert, A. Hoess, F. Flemisch, H.-P. Krueger, A. Bartels, A. Beutner, and K. Pagle, "Highly automated vehicles for intelligent transport: Haveit approach," in *ITS World Congress, NY, USA*, 2008.