Information and Software Technology 73 (2016) 136–150

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A functional reference architecture for autonomous driving

Sagar Behere*, Martin Törngren

KTH The Royal Institute of Technology, Brinellvägen 83, Stockholm SE-10044, Sweden

ARTICLE INFO

Article history:
Received 31 August 2015
Revised 16 December 2015
Accepted 17 December 2015
Available online 29 December 2015

Keywords:
Autonomous driving
Functional architecture
E/E architecture
Reference architecture

ABSTRACT

Context

As autonomous driving technology matures toward series production, it is necessary to take a deeper look at various aspects of electrical/electronic (E/E) architectures for autonomous driving.

Objective

This paper describes a functional reference architecture for autonomous driving, along with various considerations that influence such an architecture. The functionality is described at the logical level, without dependence on specific implementation technologies.

Method

Engineering design has been used as the research method, which focuses on creating solutions intended for practical application. The architecture has been refined and applied over a 5 year period to the construction of prototype autonomous vehicles in three different categories, with both academic and industrial stakeholders.

Results

The architectural components are divided into categories pertaining to (i) perception, (ii) decision and control, and (iii) vehicle platform manipulation. The architecture itself is divided into two layers comprising the vehicle platform and a cognitive driving intelligence. The distribution of components among the architectural layers considers two extremes: one where the vehicle platform is as "dumb" as possible, and the other, where the vehicle platform can be treated as an autonomous system with limited intelligence. We recommend a clean split between the driving intelligence and the vehicle platform. The architecture description includes identification of stakeholder concerns, which are grouped under the business and engineering categories. A comparison with similar architectures is also made, wherein we claim that the presence of explicit components for world modeling, semantic understanding, and vehicle platform abstraction seem unique to our architecture.

Conclusion

The concluding discussion examines the influences of implementation technologies on functional architectures and how an architecture is affected when a human driver is replaced by a computer. The discussion also proposes that reduction and acceleration of testing, verification, and validation processes is the key to incorporating continuous deployment processes.

1. Introduction

Autonomous driving is considered to be the 'next big thing' in the automotive domain. From the universities during the DARPA Grand and Urban challenges in 2004/2007 to technology showcases like the Google self-driving car, autonomous driving technology has shown a steady maturation. Today, most major truck and passenger car OEMs across the world (Daimler, BMW, Audi, Ford, Nissan, Volkswagen, Volvo, …) have active development projects in this area and typically exciting demonstrators are exhibited every year at popular public events like the Consumer Electronics Show (CES) in Las Vegas, USA.

The autonomous driving demonstrators developed so far involve some sort of "perception and higher intelligence" plugged on top of a base vehicle platform which usually incorporates

* Corresponding author. Tel.: +46 87907372.
E-mail addresses: behere@kth.se (S. Behere), martint@kth.se (M. Törngren).

http://dx.doi.org/10.1016/j.infsof.2015.12.008
0950-5849/

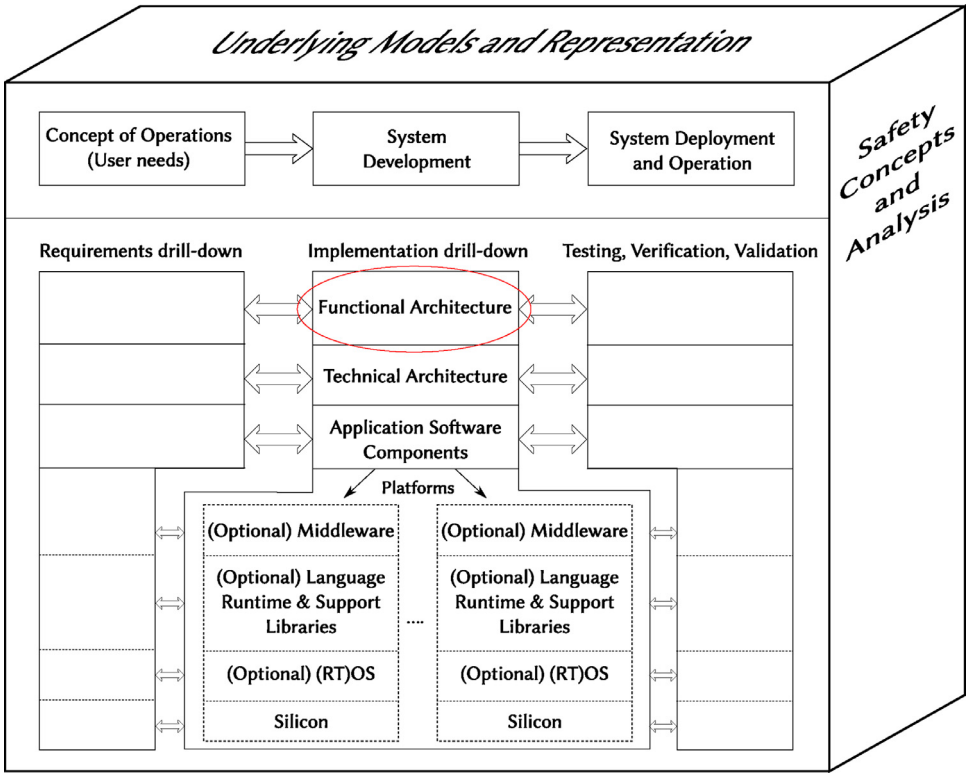

Fig. 1. Simplified context of a functional architecture.

computerized control of functions like propulsion and braking. As the technology readiness levels (TRLs) [1] increase and autonomous driving features move closer to series production, it is necessary to take a deeper look at the electrical/electronic (E/E) architectures for autonomous vehicles. These architectures cover not only the hardware, software, and communication stacks within the various electronic control units (ECUs) inside the vehicle, but also the functional hierarchies required for autonomous driving and their distribution across architectural elements. Factors like horizontal and vertical control layers, distribution and allocation of components, arbitration and conflict resolution, fault propagation and isolation of system failures, system safety, optimality of implementation, cognitive complexity etc. need to be considered for each particular deployment related to autonomous driving.

1.1. Functional architecture

We use the term 'functional architecture' with a definition corresponding to the notion of 'functional concept' in the ISO26262 functional safety standard [2]. The standard defines a functional concept as, *"specification of the intended functions and their interactions necessary to achieve the desired behavior"*. A functional architecture then refers to the logical decomposition of the system into components and sub-components, as well as the data-flows between them. It does so without reference or prejudice to the actual technical implementation of the architectural elements in terms of hardware and software. An analogous term to functional architecture is 'functional view' of the architecture description. This term is recommended by ISO 42010 [3] and pertains to the architectural description of software intensive systems, from a functional *viewpoint*. In this paper we will use the terms 'functional architecture', 'functional view' and their combination 'functional architecture view' (FAV) synonymously.

"Fig. 1 shows a simplified systems engineering context within which the artifacts may be represented in the form of models." The top part of the figure shows that the process of systems engineering begins by understand the user's needs, which are captured in a high level 'Concept of Operations'. These are the basis for the high level requirements which the system is intended to fulfill. The system is then developed and subsequently, deployed and operated. The lower part of the figure shows an amplified view of system development. It shows that there are broadly three aspects (i) requirements drill-down (ii) implementation drill-down, and (iii) testing, verification and validation. The implementation drill-down consists of a number of levels that go from the more abstract toward the more concrete. Each level has corresponding requirements and test cases. The first level is the functional architecture, which shows the logical components and their inter-connections, without reference to how they can be technically implemented. The technical implementation details are filled out in the next level - the technical architecture. The technical architecture components are then mapped to specific application software components, which in turn execute on a "platform". The platform consists of the silicon (microprocessors, microcontrollers, GPUs etc.) with optional layers of (real-time) operating systems, language runtimes and board support packages, and any middleware that abstracts the platform details. The notions of Functional and Technical architecture agree with the terms used in the ISO26262 functional safety standard. As the implementation gathers more detail, there is a corresponding increase in the number of associated requirements and test cases. The correspondences are depicted with bi-directional arrows in Fig. 1. These correspondences are sometimes referred to as "vertical and horizontal traceability links" in the systems engineering process. For example, requirements derived from a high level requirement may be linked with blocks in an architecture diagram and both can then be linked with specific test cases. If a change occurs at either end of a link, a "flag" can be raised on the link to indicate a change in status. In model based systems engineering, the various artifacts generated during the systems engineering process are represented in the form of inter-linked models.

Throughout system development, safety considerations are applicable at all levels and this is notionally depicted in the figure by the 'Safety Concepts and Analysis' dimension.

The FAV constitutes a generic solution pattern for a given set of system behaviors, which may then be implemented in a variety of ways. Thus, it constitutes a reference solution and the term *reference architecture* is also associated with the FAV. The reference architecture may then be *instantiated* to create a particular solution. In this paper, we omit the term 'reference' in 'functional reference architecture' where such an omission does not impede understanding. The FAV closely corresponds to the functional view of the system software architecture, since autonomous systems are highly software intensive. In fact, the FAV is a necessary precursor to the design of all information and software enabled functionality in an autonomous driving system.

1.2. Goals, contribution, and scope

The goal of this paper is to provide a proven reference solution for autonomous driving architectures, up to L4 levels of autonomy.[1] This solution shall assist architects of autonomous driving systems by raising into "public" debate the various considerations and solution possibilities that influence the architecture of a self-driving vehicle. These may then be evaluated within the contexts of individual projects and associated constraints, leading to specific architectural solutions. Regardless of which particular choices an architect makes, it is important that he or she is aware of the range of possibilities for each choice, the pros, cons and tradeoffs associated with it, as well as the metrics used to assess an architecture and take decisions. A functional architecture serves this goal well, since it is the starting point for architecture design and potentially, a common ancestor for a wide variety of implementations.

This paper provides three contributions to the literature in this area: 1. A discussion of the key elements in a functional architecture for autonomous driving, 2. A proposal on the division of the architecture into layers and reasoning on the distribution of the architectural elements across these layers, and 3. A proposed functional architecture for autonomous driving.

The scope of the paper is restricted to the functional architecture only; not its technological implementation. The architecture is for a single vehicle, with further delimitation to the vehicle motion specific subsystems. The vehicle may, however, communicate with other vehicles and the infrastructure. Priority is given to the division of the architecture into layers and components, describing the component functionality and their distribution among the layers, and the associated rationale. A rigorous description of the component interfaces for fully constraining their definition is avoided. This is because we believe such a description is more important for the Technical Architecture. The context of the discussion assumes the possibility of creating a legacy free, "clean sheet" vehicle design, although due care is taken to acknowledge the constraints imposed by existing vehicle architectures. Lastly, although the architecture itself is applicable to both prototype as well as series production vehicles, we delimit the discussion in this paper mostly to development of prototypes. Therefore, concerns related to series production are not particularly well-reflected in the discussed requirements and stakeholder concerns.

1.3. Research method

Research methods of Engineering Design have been used to develop the architecture. Engineering design is one of the research methods in systems engineering [4] wherein researchers address a problem which is important and novel through the activity of designing a solution [5]. The knowledge developed is directly useful for practical application. An additional outcome is theoretical development based on generalization of design experiences.

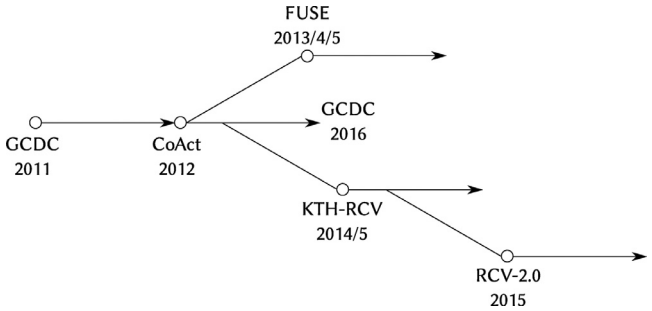

Fig. 2. Research projects contributing to architecture development.

This paper is an aggregated presentation of cumulative research results, from the projects shown in Fig. 2. Our architecture has its roots in the development of a practical solution for the problem of creating a self-driving truck operating in a platoon (convoy) of vehicles. The truck in question, an R730 tractor unit from the Swedish manufacturer Scania CV AB, is already in series production. We developed a plug-in system that required minimal modifications to the truck and enabled autonomous longitudinal motion in a platoon. The truck participated (and performed successfully [6]) in the Grand Cooperative Driving Challenge (GCDC), which took place on a public highway in The Netherlands in 2011. Subsequent generalization of the developed theoretical knowledge led to a reference architecture on cooperative driving [7]. The reference architecture was re-instantiated with additional functionality, using a different hardware stack and re-written software, for a different type of truck, for the CoAct 2012 Challenge. The CoAct Challenge was a Swedish follow up to the GCDC 2011, wherein participating vehicles were also expected to perform lateral maneuvers for overtaking and merging into the middle of a platoon. Once again, the re-instantiation performed successfully, validating the reference architecture on a different vehicle, in different scenarios. Following CoAct 2012, the refined architecture was opened up for extended refinement, stake-holder feedback, and expert analysis in the FUSE project [8], which aims at developing Functional Safety and Architecture for autonomous driving with passenger cars (Volvo car corporation is the automotive partner in this project). Meanwhile, a novel, legacy free, drive-by-wire electric vehicle was developed at KTH [9], to be used as a Research Concept Vehicle (RCV) platform for autonomous driving concepts. Our architecture enables the RCV to be a test platform for novel perception, control, and motion algorithms. The RCV platform is being developed further (RCV 2.0) by a private company with a vision for commercializing novel self-driving vehicles. Our architecture is an important enabler of the overall product vision. In 2016, KTH intends to participate in the second GCDC competition (GCDC 2016) where this architecture is expected to be reused.

The practical development and evaluation has been complemented by a comprehensive state-of-the art survey completed in 2013 [10], looking at the how the domains of Intelligent Control, Cognitive and Real Time Control architectures, Robotics, component based middleware, and software development intersect with the automotive domain and affect autonomous driving systems.

A potential weakness of engineering design, as a qualitative research method, is that of *external validity*. This is addressed here by a multitude of different case studies and engagement with experts in different domains. The applications have involved a variety of commercial and research vehicle projects, in academic and industrial contexts.

[1] The L4 autonomy level is described by the National Highway Traffic Safety Administration (NHTSA) in their policy concerning automated vehicles [http://www.nhtsa.gov/staticfiles/rulemaking/pdf/Automated_Vehicles_Policy.pdf] and refers to "full self-driving automation" where the driver is not expected to be available for control at any time during the drive.

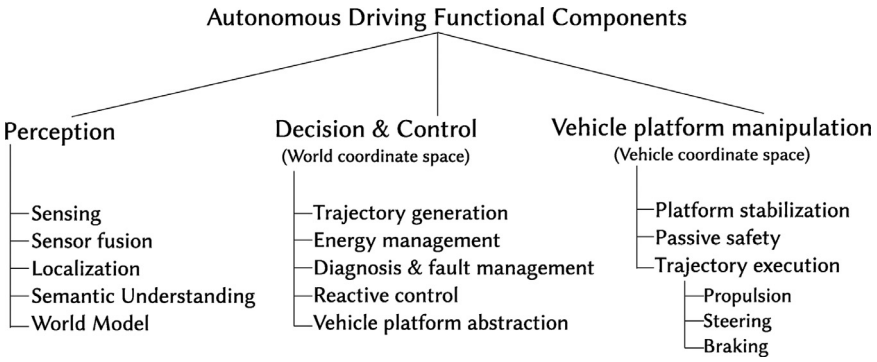

Fig. 3. Components of a FAV of an autonomous driving system.

In summary, the architecture presented in this paper has been in continuous development, application, refinement, and validation since the past 5 years.

1.4. Outline

This paper is organized as follows: Section 2 lists and categorizes the various components needed in a functional architecture for autonomous driving. The functionality of each component is also briefly described. The distribution of these components across different layers in an architecture is then described in Section 3. This is done by presenting two different extremes of functionality distribution, their pros and cons, and finally our recommended distribution and its motivation. Section 1.1 then presents a three layer architecture for autonomous driving. First, the stakeholders and their concerns are described, followed by the actual architecture, followed by a comparison with similar architectures. Section 5 then presents a discussion, which begins by reflecting back on the stakeholder concerns and a commentary on how they are fulfilled. The influences of technical implementation concern on the conceptual architecture are then discussed, followed by how the architecture is influenced when a human driver is replaced with a computer. The discussion also considers the impact of machine learning and the need for continuous deployment in terms of requirements on testing, verification, and validation. Finally, Section 6 concludes the paper and states topics of interest for future publications.

2. Functional components

We have opted to split the principal FAV components of the motion control part of the autonomous driving system into three main categories, as shown in Fig. 3. These categories are related to

1. Perception of the external environment/context in which the vehicle operates
2. Decisions and control of the vehicle motion, with respect the external environment/context that is perceived
3. Vehicle platform manipulation which deals mostly with sensing, control and actuation of the Ego vehicle, with the intention of achieving desired motion.

Each category has several components, whose functionality (from a strictly architectural perspective) will now be described.

2.1. Perception

A commonly heard phrase in the robotics community is, “Sensing is easy, perception is difficult.” Sensing means gathering data on physical variables using sensors, while perception refers to the semantics (interpretation and “understanding”) of that data in terms of high level concepts relevant to the task being undertaken. As such, sensing is just one part of an overall perception system.

The *sensing* components can be categorized into those sensing the states of the ego vehicle and those sensing the states of the environment in which the ego vehicle operates (sometimes referred to as the *internal* and *external* environments). A second and relevant categorization of sensor components, from the viewpoint of systems integration, depends on the amount of processing needed to extract relevant information from the sensor data. In our experience, this usually depends on the TRLs of both the sensor and the integrated system. At lower TRLs, in highly experimental vehicles and/or sensors, it is more common to work with raw sensor data and its filtering, estimation, fusion, association, and possible classification operations are performed as distinct and important parts of the overall system. An example of this is the processing of data from some multi-beam laser rangefinders, which return angle and distance measurements for each beam, typically at rates of 10Hz or more. At the opposite end of the spectrum are high TRL sensors from automotive vendors, which come packaged with data processing elements. Such sensors may directly output detected objects in the environment, along with relevant attributes of the detected objects like relative position, velocity, acceleration etc. Some sensor providers offer even higher level information, like the class of the detected object (car, truck, motorcycle, pedestrian,…). The reason why this type of categorization is important is that it permits the systems integrator to treat the sensor component as a blackbox[2] and the sensor’s output may be routed directly to the semantic understanding component or even directly to the world model. High TRL sensors are more commonly found in the automotive industry in late stage prototypes and when selecting or using them, emphasis often shifts to extra functional properties like failure probabilities, confidence levels of output data, and common mode failures with other sensors. This is because, factors like training datasets which a vendor may have used, data processing algorithms and their properties like time constants etc. tend to be a grey area for the systems integrator, yet the same factors may have a non-trivial impact on the analysis of safety properties of the system. In comparison, when processing of raw sensor data is an explicit part of the overall system, the details tend to be more transparent, although that does not necessarily make the analysis is easier.

The *sensor fusion* component, as the name indicates, considers multiple sources of information to construct a hypothesis about the state of the environment. In addition to establishing confidence values for state variables, the sensor fusion component may also perform object association and tracking. Association refers to

[2] Although some sensors retain the ability to transmit raw as well as processed information.

correlating pieces of information from multiple sensors to conclude that they refer to one and the same object. The process of tracking generates information about an object over a series of temporal readings. This information can be used either to track an object's attributes (e.g. relative velocity), or to classify the object into categories in a subsequent block (e.g. a sequence of readings is more likely than a single reading, to reveal an object as a pedestrian.). Finally, for certain system configurations, the sensor fusion block may also be used to eliminate some un-associated objects and data that is strongly likely to be superfluous or noise. This reduces the computation and communication load on subsequent components, like the decision and control, which need to work with the perceived data.

The *localization* component is responsible for determining the location of the vehicle with respect to a global map, with needed accuracy. It may also aid the sensor fusion component to perform a task known as *map matching*, wherein physical locations of detected objects are referenced to the map's coordinate system. The localization component typically uses a combination of GPS and inertial measurement sensors. Certain algorithms try to improve on the accuracy of localization by identifying visual landmarks via cameras. The base map layers have traditionally been stored onboard, but the trend is to move toward tiled maps, where individual tiles are dynamically streamed from a service provider based on vehicle location, but which may be locally cached.

The *semantic understanding* component is the one in which the balance shifts from sensing to perception. More concretely, the semantic understanding component can include classifiers for detected objects and it may annotate the objects with references to physical models that predict likely future behavior. Detection of ground planes, road geometries, representation of drivable areas may also happen in the semantic understanding component. In specific cases, the semantic understanding component may also use the ego vehicle data to continuously parameterize a model of the ego vehicle for purposes of motion control, error detection and potential degradation of functionality.

The *world model* component holds the state of the external environment as perceived by the ego vehicle. Conceptually, it is possible to think in terms of an extended or compound world model that includes the internal states of the ego vehicle, thus simultaneously representing the vehicle internal and external worlds. However, in practice, we have experienced that such compound world models rarely exist, because requirements and technologies at the technical architecture level usually lead to separated, optimized implementations. Also, the producers, consumers, and processing involved of data originating from the ego vehicle and its external environment, have qualitative differences. Not having a compounded world model does not eliminate a great deal of value and having a compounded world model does not add a great deal of value. Therefore, in most practical implementations, the world model component as described here, only represents the external world of the ego vehicle. We like to characterize the world model component as either passive or active. A passive world model is more like a data store and may lack semantic understanding of the stored data. Therefore, it can not, by itself, perform physics related computations on the data it contains, to actively predict the state of the world given specific inputs. The active world model, on the other hand, may incorporate kinematic and dynamic models of the objects it contains and be able to evolve beliefs of the world states when given a sequence of inputs. Other components (like decision and control) may then request a set of predictions of future world states, for a specific set of inputs, in order to determine the optimal inputs to be applied. The passive world model, as described above, is by far the most common in the autonomous driving projects we have encountered. In fact, there are efforts to create a (semi-) standardized representation [11] of the world in the form of so called 'local dynamic maps' (LDM) [12]. An LDM is technically implemented as a database, but can be conceptually thought of as a layered map. The bottom-most layers represent the most static beliefs about the world, while the topmost layers represent the most dynamic, in the sense of time. For example, the lowermost layer may be populated with a static map of the immediate surroundings of the vehicle (roads, permanent features, etc.). The layer above it may be populated with more-or-less static road objects (traffic lights, lane markings, guard rails). The next layer may contain temporary objects like diversions due to construction work. The final layer would be populated by fast-moving objects detected by the rest of the perception system (other vehicles, pedestrians, etc.). The world model component typically provides an interface to query its contents, add and remove data, concurrency, access control, replication over distributed computational media etc. In specific cases, it also holds historical information about some or all of its contents.

2.2. Decision and control

The decision and control category refers to those functional components which are concerned by the vehicle characteristics and behavior in the context of the external environment it is operating in. Reference is made to the vehicle as a whole and the way it moves in its environment, energy and fault management concerns that affect the vehicle's motion to its destination, as well as reactive control to unexpected events in the environment. The specific details of the vehicle platform that actually generate the desired external behavior and characteristics are not of prime interest.

The *trajectory generation* component repeatedly generates a set of obstacle free trajectories in the world coordinate system and pick an optimal trajectory from the set. The generation and/or selection of an optimal trajectory is constrained by factors like limitations of platform motion (e.g. non-holonomicity), energy availability, and the state of the platform with regards to faults and failures.

The emergence of dedicated, holistic *energy management* components is a relatively recent phenomenon, stimulated by the growth of hybrid and electric vehicles. This component is usually split into closely-knit sub-components for battery management and regenerative braking. Since energy is a system-wide concern, it is not uncommon for the energy management component to have interfaces with other vehicular systems like Heating, Ventilation and Air-Conditioning (HVAC), lights, chassis, and brakes. For autonomous driving, sensors and associated fusion and computation silicon may account for a significant energy consumption.

Diagnosis and fault management throughout the system components is an integral part of any well designed architecture. In the context of decision and control, this refers to identifying the state of the overall system and its components, with respect to available capabilities. The identified state would be used to influence behavior like redundancy management, systematic degradation of capabilities, triggering transitions to and from safe states, and potential driver handover. Note that this functional component notionally unites the various diagnostics and fault management functions associated with architecture components distributed throughout the system, and acts on their output.

Reactive control components are used for immediate (or "reflex") responses to unanticipated stimuli from the environment. Existing vehicle features like collision mitigation by braking may be considered as reactive control. These components execute in parallel with the nominal system and if a threat is identified, their output overrides the nominal behavior requests. Their sense-plan-act loops are typically at least an order of magnitude faster than the nominal system loop. It is sometimes the case that what is

considered reactive behavior in the presence of unexpected events, can be dealt with by very fast deliberative behavior. For example, consider the autonomous emergency braking (AEB) feature in some passenger cars. This is considered a reactive function, that monitors a small subset of sensors (compared to full autonomous driving) and initiates braking action in case of imminent collision with a moving or stationary object. The function is constantly active (when enabled) and may generate a deceleration demand that overrides other demands on the propulsion subsystem. However, if the perception and trajectory generation components are sufficiently fast, they could detect the threat and generate appropriate trajectories (in this case, strong deceleration) as part of their normal operation, negating the need for a specialized AEB system. Such a specialized system may still be implemented as a redundancy measure for supervisory control, if a system safety analysis suggests provable improvement in safety or decrease of ASIL requirements on other parts of the system. However, it wouldn't be a functional necessity. The call for reducing reactive functionality in favor of fast, deliberative functionality needs to be taken in consultation with domain experts and by considering the specific algorithms involved and the characteristics of their technical implementation. In particular, worst case execution times and end-to-end timing analyses are important factors.

The *vehicle platform abstraction* component refers to a minimal model of the vehicle platform. This model is calibrated and parameterized to reflect the actual vehicle platform and is the part that gathers most of the vehicle specific information. It can either form the interface to the vehicle platform, or it may wrap static data parameters representing a specific configuration. Ideally, this is also the part that needs to be changed when transitioning between different vehicle platforms (typically from the same family). This component may also abstract a virtual or hardware-in-the-loop (HIL) vehicle platform, which is useful for the case when the entire cognitive driving intelligence layer needs to be tested without a physical vehicle.

2.3. Vehicle platform manipulation

This category groups the components that are directly responsible for the motion of the vehicle. They abstract the principal actuation subsystems and also provide a minimum level of stability to the platform while it is in motion. Although not directly related to propulsion, components related to passive vehicle safety and occupant protection may be included in this category, since they are closely related to scenarios arising from undesirable propulsion and may be triggered by the decision and control components.

The *platform stabilization* components are usually related to traction control, electronic stability programs, and anti-lock braking features. Their task is to keep the vehicle platform in a controllable state during operation. Unreasonable motion requests may be rejected or adapted to stay within the physical capabilities and safety envelope of the vehicle.

The *trajectory execution* components are responsible for actually executing the trajectory generated by Decision and Control. This is achieved by a combination of longitudinal acceleration (propulsion), lateral acceleration (steering) and deceleration (braking). Most recent vehicles already incorporate such components and they may be considered "traditional" from the perspective of autonomous driving development.

3. Functionality distribution

We consider an autonomous vehicle architecture as broadly comprising of a vehicle platform and a cognitive driving intelligence. These two parts may be considered as two distinct layers of the architecture. It is then necessary to consider at least the following two questions

1. What kind of information should flow between the cognitive driving intelligence and the vehicle platform layers?
2. Are any changes necessary/desirable in a given vehicle platform, if it will be controlled by a computer, instead of a human being?

Answers to both questions depend on the distribution of functionality between the vehicle platform and the cognitive driving intelligence. Currently, most autonomous driving experiments that build on existing, in-production vehicles follow a typical pattern: The vehicle contains a network of electronic control units (ECUs) controlling the basic vehicle propulsion (lateral and longitudinal acceleration, braking). The vehicle manufacturer usually builds a "gateway" that allows the experimenters to send a limited set of commands to the ECUs in the vehicle network. These commands are usually set-points for the various control loops that exist in the vehicle platform. For example, the cognitive driving intelligence may continuously regulate the set-point of the cruise-control function in the vehicle.

From a theoretical viewpoint, the distribution of functionality between the cognitive driving intelligence and the vehicle platform can lie between two extremes, as shown in Fig. 4. In the figure, the vertical placement of the components (above or below dotted line) denotes which layer they are allocated to. It is only this placement that is important; the horizontal placement and location relative to other components in the same layer are purely aesthetic and carry no meaning. On one extreme, Fig. 4(a), the cognitive driving intelligence directly controls the torque outputs of the vehicle platform actuators, in a so-called "distributed I/O approach". There is no greater intelligence in the vehicle platform, which then represents just a set of distributed inputs/outputs. The cognitive driving intelligence then needs intimate familiarity with the vehicle platform and it would be difficult to de-couple and reuse the one without the other. The other extreme, Fig. 4(b), treats both the cognitive driving intelligence as well as the vehicle platform as two cooperating, relatively autonomous entities. Neither knows the intimate details about the other and the driving intelligence makes motion demands of the vehicle platform in world coordinates, which the latter makes a best effort to fulfill. The task of the driving intelligence is to perceive the world and make motion requests in this world, while the task of the vehicle platform is to realize the desired motion requests while keeping its own features and limitations in mind. In such an ideal de-coupling, the same driving intelligence should be able to operate a variety of vehicle platforms with only minor changes to the vehicle platform abstraction, provided the interface between them remains the same.

The functionality distribution shown in Fig. 4(a) leads to simplicity of the vehicle platform, but it has significant drawbacks from the viewpoint of complexity, separation of concerns, and technical feasibility. In order to perform closed-loop propulsion control of the vehicle platform, the driving intelligence would need a fairly detailed model of the platform, including its dynamics and the constraints on the vehicle actuators and sensors. Performing fine-grained (short time horizon) control of the actuators by using motion feedback from the perception system may place unreasonably high demands on the technical implementation and performance of the perception system. The functionality distribution shown in Fig. 4(b) on the other hand, is attractive because it enables a relatively clean separation of concerns. The driving intelligence need not be concerned with the finer details of how the motion it desires is achieved, just a minimal model of the vehicle's motion characteristics, via the vehicle platform abstraction component. The vehicle platform does not need to be concerned with how and why the motion commands are generated - only whether

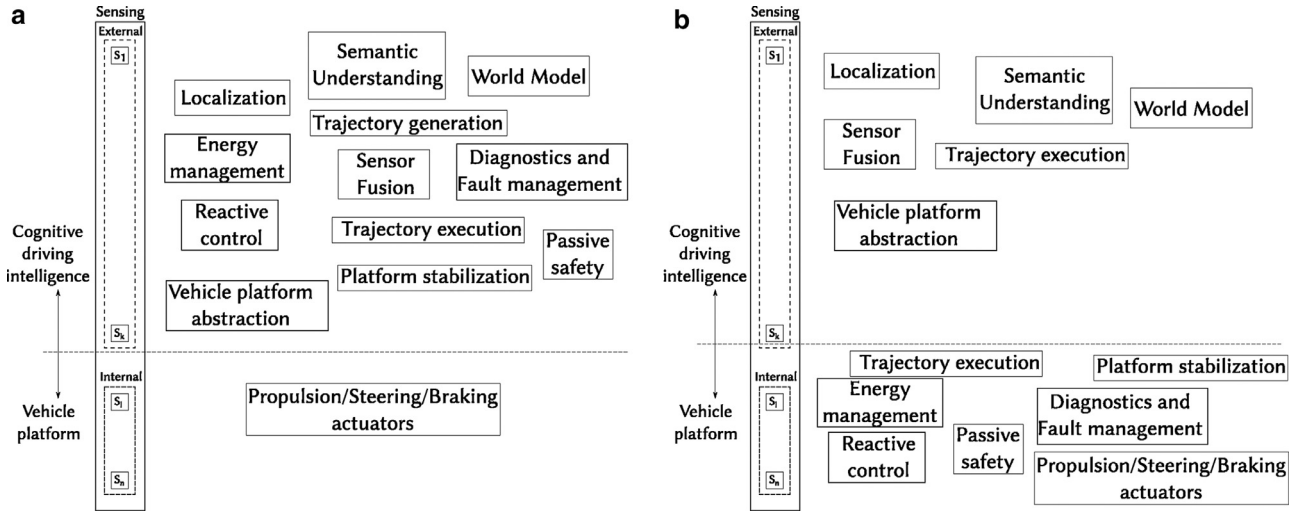

Fig. 4. Distribution of functional components across the layers in an autonomous driving architecture.

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 vehicle platform, without the driving intelligence having to be aware of them.

Our recommendation, based on the experiences described in Section 1.3, as well as the recommended architecting best practices of separating concerns and loose couplings, is to achieve as clean a split as possible, between the driving intelligence and vehicle platform [(Fig. 4(b)]. This lowers the cognitive complexity [13] (cognitive effort needed to understand a model) of the architecture, as well as reduces the potential for feature interaction and other undesirable emergent behavior, for example by clearly delineating the tasks of trajectory generation and its execution. It also enables better re-use of the driving intelligence and vehicle platform in other projects. That said, any assumptions made regarding the behavior and performance of the vehicle platform need to be made explicit via the platform abstraction component. This is especially true for end-to-end latencies on the fulfillment of acceleration requests and interpretation of sensor data by the controllers in the vehicle platform. The approach (of Fig. 4(b)) places high demands on the functionality available in the vehicle platform, with regards to its abilities for keeping the platform stable and retaining basic self-protection measures which may include reactive control. In practice, this is unlikely to be an issue because the high-end vehicles of most automotive OEMs today already incorporate such functionality and it is these high-end vehicles that are the most likely candidates for receiving upgrades to self-driving functionality.

4. Functional reference architecture

This section presents a functional reference architecture for autonomous driving, that has emerged from our work. It brings together all the functional components described so far and distributes them over the cognitive driving intelligence and vehicle platform layers. Also, it follows our proposal of achieving a relatively clean split between the two.

4.1. Stakeholders concerns

An architecture balances stakeholder concerns and requirements. The stakeholder concerns, delimited to a functional architecture, can be classified into two categories: business and engineering.

In our experience, the principal business concerns are

1. Possibility of a smooth upgrade path from existing product architecture. Particularly for OEMs with extensive product portfolios, it is desirable that the architecture of an autonomous driving system is realized as an incremental evolution of products. This facilitates maximum reuse of existing resources.
2. A unified architecture for autonomous and non-autonomous product variants.
3. Modularized architecture enabling simultaneous in-house and out sourced development of different architectural elements.
4. Use of an economic, automotive grade sensor set for perception. For mass produced automobiles, a significant amount of resources can be devoted to reducing prices by the tens of cents. In such a context, it is not economically feasible to utilize perception sensors that may well cost more than the vehicle itself. Thus, even though excellent results may be demonstrated by researchers using an expensive sensor set, the automotive industry as a whole is looking to achieve acceptable results with upgrades to sensor sets already utilized for existing features like Traffic Jam Assist, Adaptive Cruise Control, Pedestrian protection etc. Although prices may be eventually reduce based on volume of demand, strategic decisions to prefer one set of sensors over another result in substantial variation of the perception system architecture.
5. Minimization and acceleration of required testing, verification, and validation activities. Increasing autonomy makes it increasingly difficult to complete verification and validation in a timely manner. Therefore any architecture and implementation strategies that reduce time-to-market are of great business interest.
6. Designs and technologies that are certifiable and which reduce the certification efforts. It is already obvious that existing standards, regulations, and laws will need changes to account for autonomous driving systems. The exact extent of the changes is still not known, but concerns related to provable safety and certification are ranked highly. There is an especial tension here, since safety and certification related concerns advocate conservative technological approaches, while realizing the functionality of autonomous driving inherently requires bold new choices in technology and even the underlying mathematical algorithms.
7. Possibility of post-sale modifications to the in-vehicle software, to leverage improvements in driving algorithms, fix issues discovered late in the lifecycle, as well as to introduce on demand additional software based features. Companies are keen

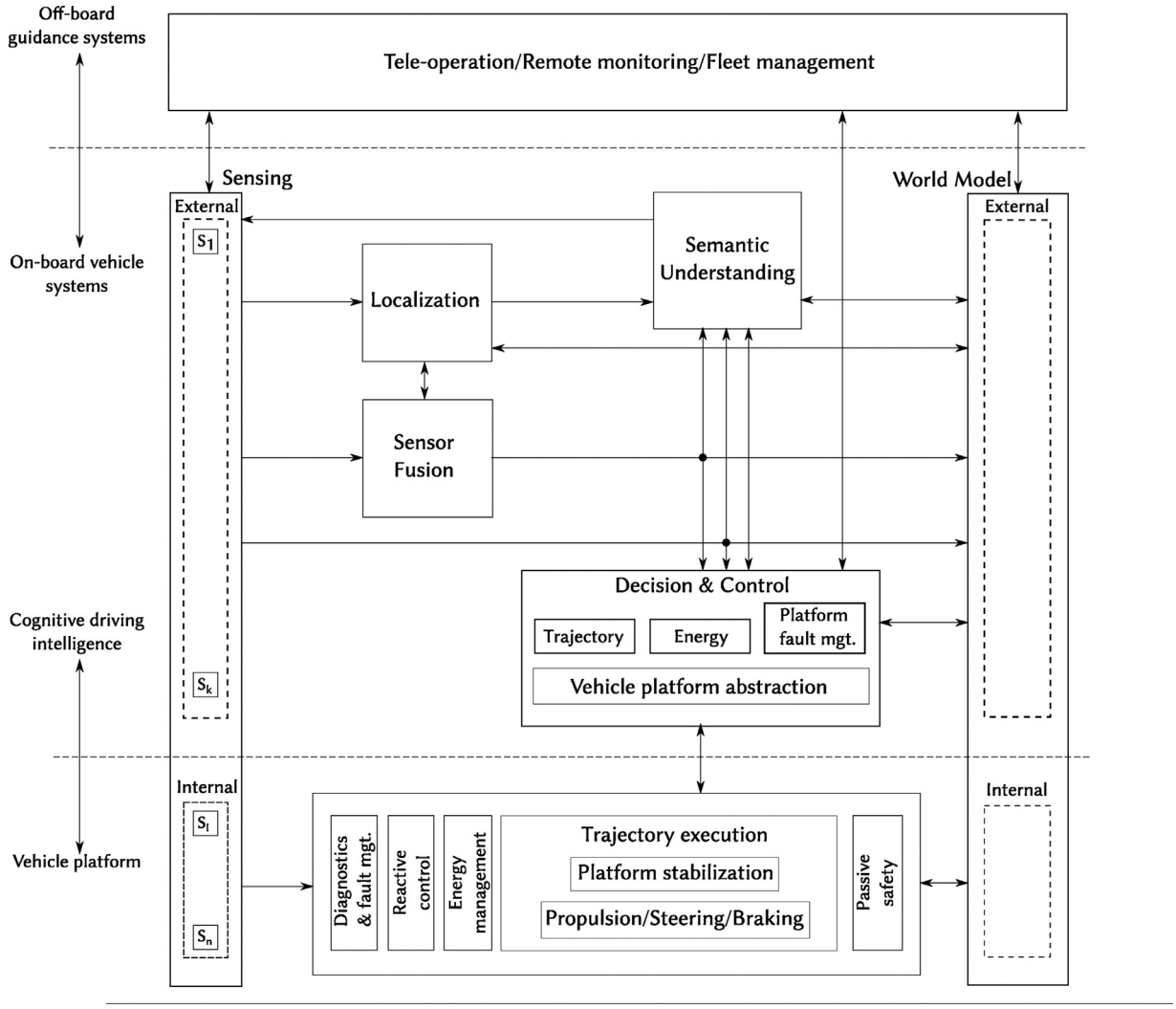

Fig. 5. A functional reference architecture for autonomous driving.

to avoid public product recalls and the ability to fix problems during regular vehicle servicing, or even 'over-the-air' is of great value.

At the functional architecture level, the principal engineering concerns are related mostly to the available functionality and possibilities of developing and testing the functionality in independent chunks

1. At least the functionality provided by the components described in Section 2 needs to be available in the autonomous driving architecture.
2. The ability to develop individual subsystems as independently as possible. This is especially relevant during the algorithm development stage.
3. Virtualized and simulated testing possibilities, to prevent practical vehicle operation concerns from slowing down early development.
4. Acknowledgment and due consideration given to technical and implementation concerns that rise up to the conceptual architecture level. While theoretically, a top-down approach will indeed not be prejudiced by technological and implementation concerns during functional architecture design, in practice it is quite important to keep technical implementation constraints in mind even during the functional architecture phase. At the very least, this helps to create functional components that map more cleanly to the components in the technical architecture.

4.2. Architecture

The proposed architecture is shown in Fig. 5. For technical and practical reasons, some components like energy management and diagnostics are allocated to both the vehicle platform as well as the driving intelligence. However, each allocation has slightly different responsibilities and scope of operation. For example, diagnosis and fault management appears both in the platform and the cognitive intelligence layer. The sensing and world model components, although conceptually unified, are split into those dealing with the external environment of the vehicle and those dealing with the Ego vehicle platform. The split helps to achieve separate technical implementations, if required, when the functional architecture is eventually refined to a technical architecture. The inter-component arrows in Fig. 5 represent data-flows in the direction of the arrow. As shown, the outputs of the sensing components go to the rest of the perception and decision and control components, either directly or indirectly, depending on the level of processing and fusion that is needed.

In our experience, it is useful to establish a data link between localization and sensor fusion. Certain sensors may exhibit repeatable tendencies at fixed locations along specific routes, like

increase in false positives, dropouts etc. Changing the level of confidence in a sensor, based on geographical location is an interesting line of research and the architecture should not be a limiting factor.

Another interesting data link in Fig. 5 is the connection from the semantic understanding component to the sensor components. This is useful in at least three scenarios. First, some specialized autonomous driving situations benefit from so-called *focused attention* mechanisms. Focused attention means exploring a specific part of the environment more deeply. This may require physical motion of the sensors and/or configuration changes to the sensors (panning a camera to a different field of view, changing the 'zoom' of a lens, etc.). Today, most sensors of most autonomous vehicles are physically fixed to a constant pose with respect to the vehicle coordinate system. But in the domain of mobile and cognitive robotics, it is quite common to have, for example, a pan-tilt-zoom camera to aid the robot in a search task. Second, calibration changes to the sensors may be needed at runtime (e.g. changing exposures based on time of day, triggering re-calibration if changes in physical alignment are suspected). Third, if communication transceivers are considered as a kind of sensor/actuator, the semantic understanding component can use it to respond to incoming communication requests, publish ego vehicle information and make asynchronous requests for information. Such communication requirements are often an integral part of scenarios like cooperative driving, where a vehicle maintains constant communication to the infrastructure and other vehicles in the vicinity.

The decision and control components include energy management from the perspectives of mission completion and overall vehicle energy demands (interior and exterior lights, HVAC). This is in contrast to the energy management component in the vehicle platform, that manages blending of hybrid propulsion systems, regenerative braking, and parts of the battery charge management and cell load balancing in electrical vehicles.

The interface between the cognitive driving intelligence and the vehicle platform consists of a multitude of trajectories in the form of motion vectors. Each vector is a time series of requested vehicle motion parameters like acceleration, velocity, etc. In the extreme case, instantaneous parameters may be sent along the interface, rather than a time series. However, if the vehicle platform is aware of the expected motion demands in the upcoming, short time horizon, it is possible to use more optimized motion control algorithms. We propose that at least two trajectories be sent continuously to the vehicle platform. One which takes the vehicle to its desired destination and another which takes it to a safe(r) state via open-loop control, in case of sudden loss of the cognitive driving intelligence layer.

The reactive control in this particular architecture is allocated to the vehicle platform, since our particular technical implementations of the perception and decision and control components have not been fast enough to deal with unexpected events as part of the deliberative control. Also, having the reactive control in the vehicle platform makes it easier to assure a basic level of self-protection for the vehicle platform, in case the cognitive driving system is totally disabled. Since the passive safety components like airbags, pre-tensioners of seat belts etc. are very tightly coupled to the vehicle platform and unlikely to be easily reused in other vehicle platforms, their control components are also a part of the vehicle platform.

We have not shown the interactions between the functional components in the vehicle platform, keeping space limitations of this paper in mind. Recent vehicles already incorporate components like platform stabilization, reactive control and abstraction of the motion control actuators. Thus, the novelty in the vehicle platform is lower, compared to that of the driving intelligence. Nevertheless, it is important to clarify that the physical actuation systems are abstracted by the 'Propulsion/Steering/Braking' component in Fig. 5.

So far in this paper, we have completely neglected to describe the reverse or return flow of data from the vehicle platform to the driving intelligence. Partially this is because the contents of the dataflow depend not only on the distribution of functionality, but also on the particular algorithms within each functional component. Although, it is tempting to consider only one-way communication from the driving intelligence to the platform and letting the perception system close the loop, in practice, the platform can provide a constant flow of states and possible asynchronous notifications that are useful for feedback and feedforward to the driving intelligence. This is particularly true in case of degraded platform functionality, where the driving intelligence must quickly make sure that the generated trajectories are still realizable. The driving intelligence usually incorporates a model of the vehicle platform and the reverse data flow may be used to continuously learn and adjust the model parameters. Furthermore, comparisons with the model and actual vehicle states can lead to better fault detection and associated reasoning related to the vehicle platform.

Finally, we note the presence of an off-board layer for teleoperation and remote monitoring, at the very top of the architecture. It is important to elaborate a bit on this, even though the scope of this paper is delimited to an individual vehicle. The off-board layer, depending on its functionality, needs to tap into various parts of the on-board architecture. In our experience, the maximum amount of data exchange occurs with the World Model, since it holds practically all the useful information needed by the off-board layer. This includes the information regarding the current state of the on-board systems, the perceived external environment, as well as any upcoming motion decisions that may be in the execution pipeline. At the remote end, all received information is typically accumulated in a database, which in turn feeds application specific views of the gathered data. Active teleoperation [14] is foreseen in use-cases where a fleet of autonomous vehicles is overseen by a command-and-control center. In such use cases, the vehicle may be able to "call home" when it gets stuck, or the remote center may actively claim control in potentially hazardous situations. In these cases, the tele-operation part of the system architecture needs to communicate with the Decision and Control part of the on-board systems. The commands sent are usually brief motion requests relative to the current location of the vehicle, or reprogrammed destinations. In our experience, the remote commands are directed at the cognitive intelligence and have relatively low bandwidth requirements. However, viewing raw sensor data (e.g. from high resolution cameras) has higher bandwidth demands and so does direct control of the components in the vehicle platform. This is rarely possible over large distances with existing communication networks.

An additional benefit of the off-board architecture and its integration with the on-board systems is related to testing, verification, and validation of new functionality and learning systems. When algorithms for new functions and active supervisory control need to directly affect the vehicle motion, such functions can be enabled to the extent that their actuation decisions are not realized, but reported to the remote monitoring systems. Given a fleet of vehicles operating in realistic situations, performance indicators of the new functions (e.g. false positives) can be gathered fairly rapidly and the functions can be enabled when sufficient confidence in their performance has been established.

4.3. Comparison with similar architectures

Given the proliferation of autonomous driving projects, 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". This is because, in contrast to the domain of task-specific algorithms (e.g. anti-lock braking), most of which can be objectively and quantitatively assessed at fine-grained levels, the domain of systems architecting in the automotive world is still largely driven by qualitative aspects like legacy considerations, brand values, organizational and development processes, commitments to specific technology partners and so on. An architecture needs to be evaluated in its context, because the context imposes unique constraints with associated implications on the design. Thus, we choose 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.

In this section, we make comparisons with the architectures of Junior—Stanford's entry in the 2007 DARPA Urban Driving Challenge, the HAVE-IT project, and a Mercedes Benz autonomous car. These architectures are relevant because they represent a steady improvement of functionality and implementation, over the past decade. Junior is a successful example of a self-driving vehicle from the early days of the technology and a largely academic proof-of-concept. The HAVE-IT project consortium had strong representations from OEMs and Tier 1 suppliers from the automotive domain, as well as independent research institutes and universities - the project focused on highly automated driving and advanced driver assistance systems. The Mercedes Benz autonomous car development had the automotive OEM Daimler AG as the majority stakeholder.

The architecture of Stanford University's DARPA Urban Challenge entry, Junior [15], is also relevant. This is an early example of an autonomous driving architecture and the interface to the VW Passat vehicle is via steering/throttle/brake controls, rather than direct longitudinal and lateral acceleration demands. This can probably be explained by the assumption that the autonomous driving architecture was designed exclusively for tight integration with one particular vehicle, which lacked general vehicle dynamics interfaces for setting acceleration and deceleration setpoints. The architecture is divided into five distinct parts for sensor interface, perception, navigation, user interface, and the vehicle interface. The localization is integrated into the perception part and there seems to be no effort to classify detected obstacles. This architecture also explicitly includes a component/layer for 'Global Services' dealing with functionality like file systems and inter-process communication. We do not describe these services because they do not strictly fit into an architecture's functional view, as defined in Section 1.1. The architecture is not strictly divided into layers, nor is there an explicit component to abstract the view of vehicle platform.

The layered approach to architectures and their description is also found in the European HAVE-IT project [16], which had its final demonstrations in June 2011. This project architecture consists of four layers: 'Driver interface', 'Perception', 'Command' and 'Execution'. The Perception layer consists of environmental and vehicle sensors and sensor data fusion. There is no mention of localization, perhaps because the system operates in close conjunction with a human driver. The Command layer contains a component named 'Co-Pilot', which receives the sensor fusion data and generates a candidate trajectory. A 'mode selection' component in the Command layer then switches between the human driver and the 'Co-Pilot' as a source of the trajectory to be executed. The selected trajectory is then handed to the Execution layer in the form of a motion control vector. The Execution layer consists of the Drivetrain control, which in turn controls the steering, brakes, engine, and gearbox. This execution layer corresponds closely to our vehicle platform layer in that it pertains to drivetrain control and *"..to perform the safe motion control vector."* [16]. Also similar is the usage of a motion control vector as an interface to the vehicle platform/execution layer. Our architecture additionally incorporates energy management as an explicit part of the decision and control component, which is especially valuable for electric and hybrid drivetrains, since then considerations of estimated range can be incorporated in the long term trajectory planning. The HAVE-IT architecture evolved in the context of Advanced Driver Assistance Systems (ADAS) with a strong reliance on the human driver and emphasis on driver state assessment components in the command layer; it remains unclear how well it can be adapted to L4 autonomous systems, where a human driver may not be present.

Close comparisons can be made with the architectural components of Bertha, the Mercedes Benz S-class vehicle, that recently (2014) completed a 103 mile autonomous drive from Mannheim to Pforzheim [17]. In the system overview presented in [17], components like perception, localization, motion planning, and trajectory control are clearly identified. These agree well with the components we have described in this paper, however this is hardly surprising. Every autonomous driving system requires these functional components and they are likely to show up in practically every architecture for autonomous driving. The system overview in [17] does not explicitly acknowledge the existence of components for semantic understanding, world modeling, energy management, diagnostics and fault management, and platform stabilization. It is possible that some or all of these were present, but not mentioned. This is especially true for diagnostics and fault management. A part of semantic understanding, related to classification of detected objects can (and often is) put in the Perception component, as in [17], but we see benefits in the explicit separation of sensor fusion and semantic understanding advocated by our architecture. The isolation of semantic understanding from raw sensor fusion enables faster and more independent iterations and testing of newer algorithms, without affecting the rest of the system. This is directly relevant to the engineering stakeholder concerns of independent development of individual subsystems, as well as their virtualized simulation and testing. Further, the raw object data from sensor fusion is still of value to the Decision and Control components, despite a lack of accompanying semantic understanding. This is because, although knowledge of whether a detected object is a pedestrian or motorcycle is useful for optimized path planning, collision with the object still needs to be avoided regardless of its classification. In a similar vein, incorporating a distinct component for world modeling, enables incremental sophistication in internal representations while retaining (backward compatible) interfaces. The existence of a distinct world model components makes it easier to answer questions like, "How will the world evolve if I perform action X instead of action Y?" Although [17] mentions the existence of a 'Reactive Layer', it makes no mention of any other layers in the architecture and how the components are distributed across them. In our paper, we make a clear distinction between the vehicle platform and cognitive driving intelligence layers and provide a rationale for our proposed component distribution.

Comparison of these and a few other architectures with our proposed architecture leads us to believe that the explicit recognition of semantic understanding, world model, and vehicle platform abstraction components are unique to our architecture. This is not entirely coincidental, since our incorporation of these components is, to some extent, a deliberate action to resolve the shortcomings we perceived during our early state of the art surveys. Furthermore, our architecture has been applied to a larger variety of vehicles (commercial trucks, passenger cars, as well as novel, legacy free designs) and therefore necessarily incorporates features related to greater isolation of functionality into distinct components and abstraction of vehicle interfaces. The aggressive partitioning of architectural components provides significant freedom to the component developers to modify and test new algorithms, without affecting the rest of the system. It also reduces

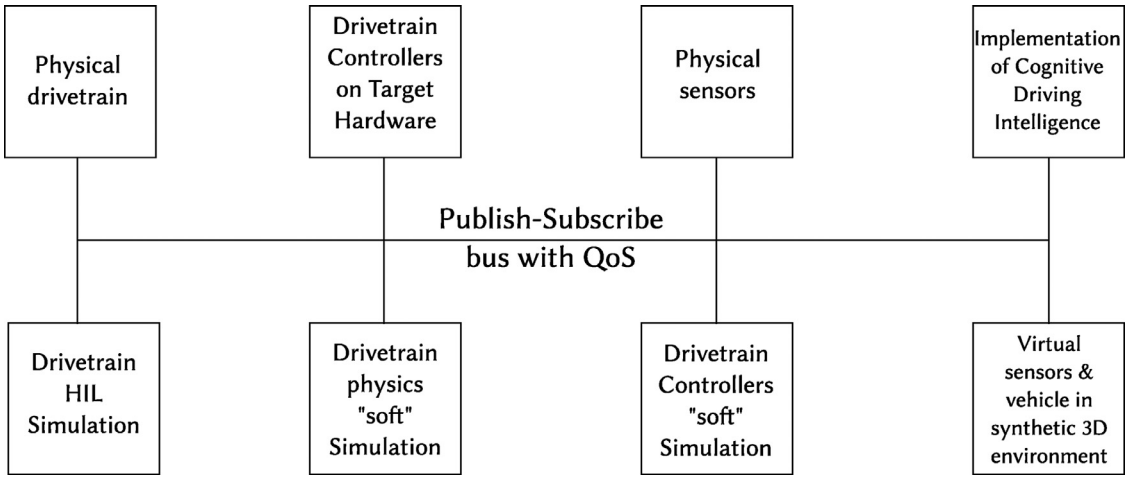

Fig. 6. A setup for development and testing of the architecture.

the cognitive complexity of the system and makes it relatively easy to foresee potential pitfalls and debug causes of objectionable behavior.

5. Discussion

A functional architecture is merely the first step in the architecting process recommended by safety standards like ISO26262 and other design methodologies. Yet, it already yields enough material for an engaging discussion and has far reaching consequences for the overall system design. In this section, we reflect back on the presentation so far and highlight a number of additional topics relevant to the discussion

5.1. Stakeholder concerns

In Section 4.1, we pointed out that business concerns required a unified architecture for autonomous and non-autonomous product variants, and a smooth upgrade path from the existing product architecture. These concerns are fulfilled by the architecture by virtue of the proposed layers and distribution of functionality between them. Under the reasonable assumption that it is an OEM's modern flagship product which will be upgraded to autonomous driving, it can be argued that the vehicle platform layer corresponds closely to the existing non-autonomous product. This is because it incorporates all the functionality needed by a non-autonomous vehicle and rather little functionality needed by an autonomous one. The existence of components to realize given acceleration or velocity setpoints is already present in modern (non-autonomous) vehicle architectures. Such components are necessary, because they are integral for realizing existing ADAS features like adaptive cruise control, lane keeping assist, automatic emergency braking etc. The upgrade path to an autonomous vehicle variant therefore consists of adding the cognitive driving intelligence layer to the existing vehicle platform. The functionality mentioned in the stakeholder concerns relating to environmental perception, localization, trajectory generation and tracking, propulsion control, and fault handling is present and accounted for in the architecture, via the different architectural components shown in Fig. 5.

The key difference between implementations of the vehicle platforms for autonomous and non-autonomous vehicles would be with respect to available redundancies in the system, that ensure at least fail-safe operation. One the one hand, it can be argued that such redundancies are unnecessary. Such an argument states that since human driven vehicles do not incorporate redundancies, they are not necessary when the human is replaced by an equivalent driving intelligence. Such arguments miss out on the fact that social acceptance for accidents caused by autonomous driving is likely to be substantially lower than accidents caused human drivers. Furthermore, for autonomous driving to be justified, the accident rate needs to be *lower* than for human driven vehicles. Therefore, in event of failures in the vehicle platform components, it should be possible to bring the vehicle to a safe state. One way of achieving this is via redundant systems. However, redundancies add to vehicle cost and sometimes there is no physical space remaining in the vehicle to accommodate redundant units. Both these factors can, to some extent, be mitigated by smarter actuator and component design (multiple independent brake systems within the same housing, redundant ECU cores operating in lockstep, etc.), as well as utilizing inherent system redundancies to provide degraded functionality (e.g. steering by braking). The challenge for employing systematic degradation of complex system behaviors, in order to avoid full redundancy, is that verifying all degradation modes and assuring system properties like safety is more complex. In the absence of established theoretical methods-of-proof that are acceptable to certification authorities, the industry sometimes prefers to simply bypass the problem by installing a full redundant system which will take over in case of primary system failure. Such a pattern is well-known and accepted, reducing the burden-of-proof demanded by certification. However, the mentioned problems of cost and space claim will no doubt drive the development of acceptable techniques in this area.

Our proposed functional architecture makes no assumptions on any particular sensor setup. As such, it neither promotes nor limits the usage of particular automotive grade sensors. However, the component definitions permit independent development and testing of whole subsystems and even vehicle architecture layers. This is instrumental to speeding up the testing, verification, and validation processes. In our experience, an experimental setup as shown in Fig. 6 provides maximum flexibility to implement and test different parts of the architecture. The setup distinguishes between concrete artifacts, final software implementations on targeted hardware, and simulations. The simulations are further split into hardware-in-the-loop (HIL) rigs, those running on a general purpose computer in software like Simulink ("soft simulations"), and virtualized sensors and vehicles operating in a synthetic three dimensional environment, like a 3D gaming engine. Each of these can be swapped in and out as needed during development, making individual subsystem developers less reliant on the availability of other functioning subsystem implementations. For example, the drivetrain controllers on target hardware can be tested against

a drivetrain HIL rig, in the absence of a physical drivetrain. Similarly, code in the cognitive driving intelligence can be tested often against a virtual vehicle in a synthetic 3D world, eliminating the need for a functioning physical vehicle to be constantly available.

The components, partitioning, and layers of the architecture acknowledge technical implementation concerns and typical technologies, as discussed in the next subsection.

5.2. Influences from the technical architecture

The way functionality is separated into layers in the proposed architecture is an acknowledgment of the underlying technical implementation differences between the cognitive driving functions and the vehicle platform functions. Each layer can be implemented by a different team, using different technology stacks, tools, programming styles and even differing educational backgrounds. The importance of this should not be underestimated in any serious development project.

In our repeating experience, due consideration needs to be given to differences between three different domains, with respect to technical skills of their practitioners and their tools of choice. These are the domains of control engineering, embedded systems programming, and computer science.

- Control engineers are used to implementing their algorithms in tools like Simulink. They initially test and verify their algorithms using rapid control prototyping tools like the dSpace MicroAutoBox, Simulink Real-Time, NI LabView etc. Such tools make it possible to execute models at the push of a button and the engineers do not need to think about things like memory allocation, threads, mutexes and semaphores etc. The engineers do not need to be expert programmers and any relatively minor programming effort can be encapsulated in constructs like Simulink's S-Functions. Entire vehicle functions can be developed, prototyped, tested, calibrated, and verified without touching a single line of source code, focusing only on algorithmic (control specific) concerns.
- Embedded systems programmers are typically used to programming with resource constrained microcontrollers, often at the register and peripheral levels. The code executes either on the bare silicon, or with lightweight, deterministic RTOSes. Primary concerns are to keep the programming efficient, real-time, and conservative. Scheduling of tasks, input/output, interrupt handlers, safety critical development practices etc. are important as well.
- Computer scientists generally run their code on relatively "big" general purpose computer systems, typically with operating systems like Linux, gigabytes of RAM and storage, heavyweight middleware like the Robot Operating System (ROS) and many software programs concurrently executing on the computers. Tools for live graphical visualization of data, data loggers, network connectivity, etc. all run concurrently on such systems. The programmers often do not need to perform rigorous analyses for real-time, worst case execution times, schedulability, and resource consumption, so long as the computers are oversized and powerful enough.

At least during the early prototyping stage, it becomes the responsibility of the architects to enable the domain experts to go ahead with their own preferred development tools and practices. This usually becomes a matter of selecting suitable toolchains and development platforms, which are connected together in the vehicle using networks like CAN and/or Ethernet.

In the proposed architecture, the components in the vehicle platform are typically implemented by control engineers and embedded systems programmers. The components of the cognitive driving intelligence are implemented by computer scientists. The former execute on "micro-controller" like platforms, the latter on general purpose computers, with their associated differences in toolchains. Moreover, with the setup shown in Fig. 6, the different components can be developed and tested independently, to a large extent. *The components of the conceptual architecture can be mapped without being split up, onto computational platforms in the technical architecture*.

5.3. Reusing the vehicle platform

The emphasis on separation of concerns and system partitioning naturally raises the question of whether it is possible to reuse the cognitive driving intelligence and vehicle platform layers independently of each other. This would require the definition of functional and extra-functional requirements on both layers and a means of encoding those requirements formally into the system. An example of the latter is the vehicle platform abstraction component in the proposed architecture, which encodes the motion specific vehicle platform specification. In our implementations, this component has been rather simple; it held the kinematics and vehicle dynamics parameters specifying the limits and capabilities of the vehicle platform. It was used mostly by the trajectory generation component to generate trajectories that would be realizable by the vehicle platform. The specification of a complete set of parameters that constrain the vehicle platform definition is, we believe, still an active area of research. In this section, we highlight some key factors that influence the definition (and reusability) of a vehicle platform.

- The *distribution of functionality* across components in the two layers, as described in Section 3, has the strongest influence on reusability. This is because the interfaces between the two layers strongly depend on the capability of each layer, which in turn is defined by the components present in it. For a cognitive driving intelligence layer to be reused on another vehicle platform, it is necessary for the two vehicle platforms to have identical functionality. For the component distribution we propose, the vehicle platform needs to have the capability to accept and execute motion trajectories which are specified either as instantaneous values of motion vectors, or a time series of motion vectors. A vehicle platform may be capable of accepting two simultaneous trajectories, one of which takes the vehicle to the desired destination, which the other takes it to a safe(er) state. The structural and semantic interface between the two layers needs to be identical.
- The *extra-functional requirements on signals* flowing between the two layers, like accuracy, precision, frequency of transmission etc. are heavily influenced by the vehicle motion specification. The maximum desired acceleration and deceleration, velocity, braking distance, allowable trajectory tracking error in the vehicle platform have a close inter-play with the timing of processes in the cognitive driving intelligence. For example, the period of time elapsed between two successive sweeps of a lidar sensor has a direct impact on the maximum safe velocity of the vehicle platform. Too high a velocity would mean that the vehicle covers a significant distance between two sensor readings, leading to potentially dangerous situations in case of system failures. This can be mitigated by using time offsetted readings of redundant sensors, which in turn puts requirements on the sensor fusion, perception, and classification processes.
- The *energy management capabilities* of the platform and strategy for prioritizing supply of energy to platform components can significantly influence the decision and control parts of the cognitive driving intelligence. For example, a coupling at the working fluid circuit level between energy demanding subsystems like the vehicle interior Heating Ventilation and Air

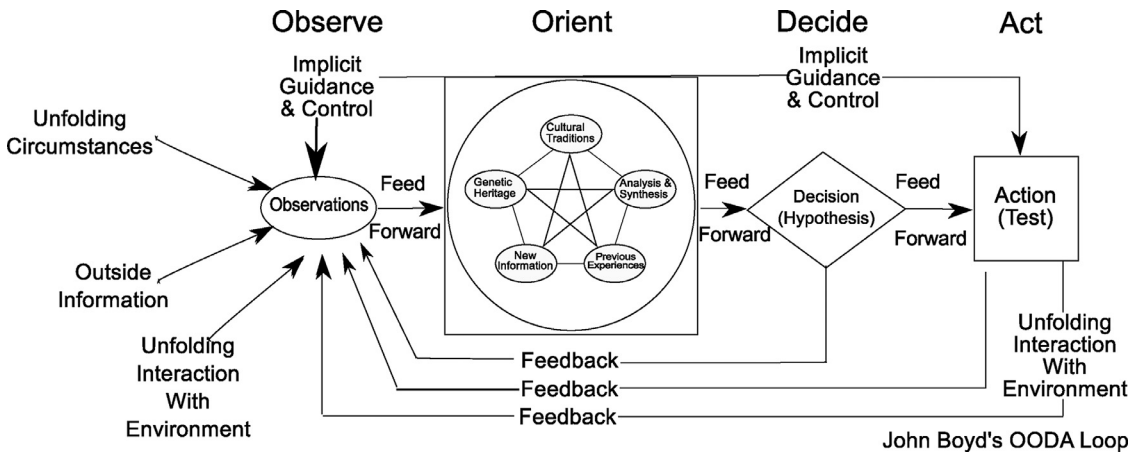

Fig. 7. The OODA loop [source: adapted from CC-BY 3.0 licensed image by Patrick Edwin Moran].

Conditioning (HVAC) and cooling of motion critical components like electric motors, battery packs etc. can lead to differences in the energy utilization strategies at the cognitive driving intelligence level. As such, details of the vehicle platform's mechanical construction need to be a part of the vehicle platform specification.

- For systems where handover to a human driver is expected, the *driver monitoring* features that may be built into the vehicle platform need to be a part of the state information communicated from the vehicle platform to the cognitive driving intelligence. Ideally, such functionality would not be a part of the vehicle platform, but this depends on legacy considerations that would vary from one platform to another.
- *Levels of redundancy* available in the vehicle platform and the capabilities and limitations of the redundant subsystems are an important platform characteristic. While the cognitive driving intelligence need not be aware of the finer details of subsystem redundancies, it needs to know how their functioning affects the platform capabilities, at least from a fail-safe or fail-operational perspective, so as to determine when it is no longer an option to continue autonomous driving.
- The *support for teleoperation* at the vehicle platform level has an influence on reusability. When creating the overall vehicle architecture, the locations at which the teleoperation stack has hooks and interventions into the architecture needs to be determined. If the teleoperation stack provides low-level commands (like wheel torques or engagement of brakes) to components in the vehicle platform, in addition to providing higher level motion trajectories to the cognitive driving intelligence, then it is necessary that the structural and semantic teleoperation interface is a part of any reusable platform specification.
- The high and low voltage *DC power bus* architecture (for electric vehicles) and *communication network topology* affects how the cognitive driving intelligence works with the vehicle platform, at least from the perspective of power supply, energy management, and redundant means of communication with the vehicle architecture and its components. Also, the behavior of the vehicle platform in case of loss of communication with the cognitive driving intelligence is an important factor when making safety arguments pertaining to overall vehicle behavior.
- The *non-motion related items* of the vehicle platform like horn, interior and exterior lighting, body, interior controls, and climate systems, Human Machine Interface etc. also need to be a part of the vehicle platform specification, in the context of reusability.

Given the richness of the vehicle platform specification and the deep dependencies between many different aspects of the vehicle platform and the cognitive driving intelligence, it remains to be seen whether reusability across product portfolios is a realistic goal for the near future. Nevertheless, we consider that the separation of concerns and system partitioning is relevant for managing the system complexity and fits the purpose of a functional reference architecture, as an ideal model.

5.4. Replacing the human

One useful way to assess the architecture is to compare its components and their functionality with the capabilities of a human driver. We choose to make the comparison by discretizing the human driver's activities into a set of categories and then reasoning on the vehicle architecture effects of each category. One of the models to discretize the human driver's activities is the Observe-Orient-Decide-Act (OODA) model [18]. The OODA loop, shown in Fig. 7, was developed by US military strategist John Boyd and has found wide acceptance in many fields of human endeavor related to rapid judgment and decision making in an uncertain environment.

Broadly speaking, the Observe, Orient and Decide parts of the loop, as applied to overall vehicle motion, would be performed by the cognitive driving intelligence components. The Observe part corresponds to the sensor components in the architecture, whereas the Orient and Decide parts correspond to the semantic understanding and Decision and Control components respectively. The Act part of the OODA loop corresponds to the vehicle platform. It is important to point out that this mapping is with respect to the overall vehicle motion. For the operation of individual parts, like the vehicle platform control, similar OODA loops or other patterns may be present at a more fine-grained level. Both the observe and orient parts can apply to internal contexts i.e. the Ego vehicle states, as well as to external contexts i.e. the environment within which the Ego vehicle operates.

With regards to observation (sensing), the suite of sensors on an autonomous vehicle may collectively match (and even exceed) the sensory capabilities of ordinary human drivers. However, the Orientation part, corresponding to the perception and interpretation of sensory data remains far superior in humans. This is because humans can construct the external context far more rapidly than computers can and they can reason more broadly and deeply about it than a computer can. Furthermore, humans can also apply rapid learning to their orientation capabilities. With computers, the extent of learning is still rather limited and restricted to specific contexts. Learning also presents a particularly difficult problem when it comes to product deployment, as discussed in the next section.

The 'Act' part of the OODA loop is implemented via the vehicle platform. The main changes to the vehicle platform, when driven by a computer, relate to increased redundancy and self-protection. Redundancy needs to be provided not just to ensure safety, but also to increase the availability (non-disruption of provided service) in order to complete the mission.

An alternative to the OODA model is the Monitor-Analyze-Plan-Execute plus Knowledge (MAPE-K) model [19,20]. This model had its origins in the management of Information Technology systems, but has found wide applicability in architectures for robotics applications and adaptive systems. In the MAPE-K loop, sensors and actuators are attached to a 'managed element' and they connect the managed element to an 'autonomic manager'. The autonomic manager consists of processes to monitor the managed element, analyze the monitored data, as well as to plan desired outcomes and execute the plans via the actuators. The managed element typically represents a hardware or software resource. The combination of the managed element and its autonomic manager is together referred to as an 'autonomic element'. Hierarchical MAPE-K loops can be applied to an architecture for autonomous driving, such that a managed element at one level of the hierarchy can be an autonomic element in a lower level. A detailed analysis of our architecture in the context of a MAPE-K loop has not yet been made. However, we believe that the MAPE-K loop can be especially applicable to redundancy, monitoring, and supervisory elements of the architecture.

The replacement of a human driver with a computer changes the nature of communication with the driving platform. In the case of the human, the communication is more qualitative in nature and performed by operating a set of buttons, levers, and pedals. A human driver does not know (or need to know) quantitatively, the amount of acceleration he or she is requesting, when pressing the gas pedal. In contrast, a computer communicates with the vehicle platform in a more precise and quantitative manner. A precise value of acceleration or deceleration is requested and it is expected that the resulting motion can be quantitatively evaluated for conformance with the request. The driving computer may also have access to far more details about the internal state of the vehicle platform, compared to the knowledge a human driver maintains about the vehicle, while driving it. However, with the proposed split of the architecture into a driving intelligence and a vehicle platform, it is expected that the level of information that the two components need to know about each other, is minimized and encapsulated in the vehicle platform abstraction component.

Finally, it is unlikely that in the near future, computers will demonstrate reasoning capabilities that match or exceed humans. Also, autonomous vehicles are envisioned as just one component of a future Intelligent Transport System(ITS). Both these factors point to the need for some sort of off-board intelligence (human or machine) that will assist the autonomous vehicle in a multitude of ways. This is captured in the proposed architecture using the topmost layer for off-board services. The redistribution of functionality between on-board and off-board vehicle systems will thus be an interesting area of exploration.

5.5. Learning and continuous deployment

Automated driving depends heavily on software and software is prone to having bugs. As the volume and complexity of software in vehicles increases, it becomes increasingly difficult to achieve sufficient test coverage and it is likely that problems in software can be discovered after product launch. Fixing these problems requires post-sale modifications to the in-vehicle software, which is a stakeholder concern discussed in Section 4.1. However, this is just one motivation for introducing the ability of continuous software deployment in vehicles.

Another motivation is related to the very nature of the software itself. Software algorithms can leverage learning over time to improve and optimize their performance. Incorporation of learning poses problems in the traditional business model. Traditionally, OEMs prefer to extensively validate the core software prior to commercial deployment. Once the vehicles are deployed, any modifications to the software usually require re-validation of not just the individual modification, but possibly of the entire modified vehicle. Revalidating the entire vehicle every time a change is made is not economically feasible. Also, with the current methods of testing and verification, the entire validation process takes just too long. We foresee that addressing this challenge would require both reduction and acceleration in the amount of testing and verification that is needed. This requires advances in two distinct areas: The virtual testing and verification of vehicles, and the development of 'correctness-by-construction' methods. The former would be needed in a scenario where field vehicles submit learned information to the OEM "cloud" where the learning can be extensively tested on virtual vehicles in virtual scenarios, in accelerated time. Only that learning which measurably improves the vehicle performance and is established to be safe, would be accepted and subsequently deployed. The latter advancement (in correctness-by-construction methods) is crucial in order to remove the need for verification of the entire vehicle, when changes are made to specific subsystems. Such methods should assure, for example, a 'side-effect-free' integration of individual components. Techniques like design by contracts, contracts based programming, etc. are useful in completely constraining the definition of individual (sub)components and proving that when a replacement component fulfills all constraints, there is no change to the overall system behavior.

On the topic of learning, any improvements to the cognitive driving intelligence need to be categorized, for example, as rule based, knowledge based, and skill based [21]. Improvements in some specific categories are likely to be incorporated far more rapidly than in others. An example of this is where the vehicle relies on constantly updating maps supplied from off-board sources. The map may be considered a knowledge source and any updates to the map that contain instances of more and detailed information, in previously known categories, would help the same vehicle software to make potentially better decisions. In contrast, improvements to a pedestrian detecting algorithm would be considered a 'skill based' improvement and would need to be validated far more thoroughly than the map case.

6. Conclusion and future work

The functional architecture described in this paper has been refined and applied to three different vehicle categories, in different projects spanning a 5 year period. The architecture helped each project in meeting its goals. At the same time, we acknowledge that there are no uniquely and definitively correct solutions in architecting and at this point it is difficult to say anything beyond, "These ideas are derived from the state of art and practice and they have worked well for us." We believe that patterns exist for autonomous driving architectures and these patterns ought to be documented and debated. In this paper, we have contributed to such efforts by describing the principal functional components needed for autonomous driving, together with some reasoning regarding their distribution across the architecture. A specific architecture incorporating the ideas has also been presented.

Future work in this area could be in the refinement of the conceptual architecture, as well as documentation and elaboration on the model based development processes used to refine the conceptual architecture and develop vehicles based on it. In particular, we would like to address the following topics in future publications

- A deeper safety centric analysis in the form of functional safety concepts as per ISO26262
- A comprehensive definition of component interfaces, requirements, and computation characteristics
- Description of a technical architecture, based on the proposed conceptual architecture
- Methods of testing, verification, and validation of autonomous driving systems
- Coverage of systems beyond propulsion control, like driver interfaces and integration with in-vehicle infotainment.

Acknowledgments

The authors would like to gratefully acknowledge funding provided by the VINNOVA projects FUSE no. 2013-02650 and ARCHER no. 2014-06260, as well as the support provided by the Volvo Car Corporation and Scania CV AB.

References

[1] European Commission, G. Technology Readiness Levels (TRL), HORIZON 2020—WORK PROGRAMME 2014–2015 General Annexes, Extract from Part 19—Commission Decision C(2014)4995. http://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415-annex-g-trl_en.pdf.
[2] ISO 26262:2011 Road Vehicles—Functional Safety, 2011.
[3] ISO 42010Systems and Software Engineering—Recommended Practice for Architectural Description of Software-intensive Systems, 2011.
[4] T. Ferris, On the methods of research for systems engineering, in: Seventh Annual Conference on Systems Engineering Research, 2009 (April).
[5] T. Ferris, Engineering design as research, in: M. Mora, O. Gelman, A.L. Steenkamp, M. Raisinghani (Eds.), Research Methodologies, Innovations and Philosophies in Software Systems Engineering and Information Systems, IGI Global, 2012, doi:10.4018/978-1-4666-0179-6.
[6] J. Mårtensson, A. Alam, S. Behere, The development of a cooperative heavy-duty vehicle for the GCDC 2011: team scoop, IEEE Trans. Intell. Transp. Syst. 13 (3) (2012) 1033–1049, doi:10.1109/TITS.2012.2204876.
[7] S. Behere, M. Törngren, D.-J. Chen, A reference architecture for cooperative driving, J. Syst. Archit. 59 (10, Part C) (2013) 1095–1112, doi:10.1016/j.sysarc.2013.05.014. Embedded Systems Software Architecture.
[8] The FUSE Project. Functional Safety and Evolvable Architectures for Autonomy. http://www.fuse-project.se/.
[9] O. Wallmark, et al., Design and implementation of an experimental research and concept demonstration vehicle, in: 2014 IEEE Vehicle Power and Propulsion Conference (VPPC), 2014, pp. 1–6, doi:10.1109/VPPC.2014.7007042.
[10] S. Behere, Architecting Autonomous Automotive Systems: With an Emphasis on Cooperative Driving (Licentiate thesis), School of Industrial Technology and Management, KTH The Royal Institute of Technology, 2013.
[11] S. ETSITR 102 863 V1.1.1 Local Dynamic Map (LDM)—Rational for and Guidance on Standardization. http://www.etsi.org/deliver/etsi_tr/102800_102899/102863/01.01.01_60/tr_102863v010101p.pdf, 2011.
[12] Z. Papp, C. Brown, C. Bartels, World modeling for cooperative intelligent vehicles, in: 2008 IEEE Intelligent Vehicles Symposium, 2008, pp. 1050–1055, doi:10.1109/IVS.2008.4621272.
[13] H. Kopetz, The complexity challenge in embedded system design, in: 2008 11th IEEE International Symposium on Object and Component-oriented Real-time Distributed Computing (ISORC), IEEE, 2008, pp. 3–12, doi:10.1109/ISORC.2008.14.
[14] T. Fong, C. Thorpe, Vehicle teleoperation interfaces, Auton. Robots 11 (1) (2001) 9–18.
[15] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, S. Thrun, Junior: the Stanford entry in the urban challenge, in: M. Buehler, K. Iagnemma, S. Singh (Eds.), The DARPA Urban Challenge, Springer Tracts in Advanced Robotics, vol. 56, Springer, Berlin, Heidelberg, 2009, pp. 91–123, doi:10.1007/978-3-642-03991-1_3.
[16] The HAVE-it EU Project. Deliverable D12.1 Architecture Document. http://haveit-eu.org/LH2Uploads/ItemsContent/24/HAVEit_212154_D12.1_Public.pdf.
[17] J. Ziegler, et al., Making Bertha drive: an autonomous journey on a historic route, IEEE Intell. Transp. Syst. Mag. 6 (2) (2014) 8–20, doi:10.1109/MITS.2014.2306552.
[18] J. Boyd. The essence of winning and losing. Unpublished lecture notes, 1996.
[19] J. Kephart, D. Chess, The vision of autonomic computing, Computer 36 (1) (2003) 41–50, doi:10.1109/MC.2003.1160055.
[20] M.C. Huebscher, J.A. McCann, A survey of autonomic computing—degrees, models, and applications, ACM Comput. Surv. 40 (3) (2008) 7:1–7:28, doi:10.1145/1380584.1380585.
[21] J. Rasmussen, Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models, IEEE Trans. Syst. Man Cybern. SMC-13 (3) (1983) 257–266, doi:10.1109/TSMC.1983.6313160.