Radboud Repository

Radboud University Nijmegen

PDF hosted at the Radboud Repository of the Radboud University Nijmegen

The following full text is a preprint version which may differ from the publisher's version.

For additional information about this publication click this link.
http://hdl.handle.net/2066/195379

Please be advised that this information was generated on 2019-04-22 and may be subject to change.

A Standard Driven Software Architecture for Fully Autonomous Vehicles

Alexandru Constantin Serban*†, Erik Poll* and Joost Visser*†
*Radboud University, Nijmegen, The Netherlands
†Software Improvement Group, Amsterdam, The Netherlands,
Email: *{a.serban, erikpoll}@cs.ru.nl, †{a.serban, j.visser}@sig.eu

***Abstract*—The goal of this paper is to design a *functional software architecture* for fully autonomous vehicles. Existing literature takes a *descriptive* approach and presents past experiments with autonomous driving or implementations specific to limited domains (e.g. winning a competition). The architectural solutions are often an after-math of building or evolving an autonomous vehicle and not the result of a clear software development life-cycle. A major issue of this approach is that requirements can not be traced with respect to functional components and several components group most functionality. Therefore, it is often difficult to adopt the proposals. In this paper we take a *prescriptive* approach starting with requirements from an automotive standard. We use a NIST reference architecture for real-time, intelligent, systems and well established architectural patterns to support the design principles. We further examine the results with respect to the automotive software development life cycle and compliance with automotive safety standards. Lastly, we compare our work with other proposals.**

***Index Terms*—Intelligent vehicles, Autonomous vehicles, Software architecture**

I. INTRODUCTION

Autonomous driving is no longer a lab experiment. As manufacturers compete to raise the level of vehicle automation, cars become highly complex systems. Driving task automation is often regarded as adding a layer of cognitive intelligence on top of basic vehicle platforms [1]. While traditional mechanical components become a commodity [2] and planning algorithms become responsible for critical decisions, software emerges as the lead innovation driver.

Since the amount of software grows, there is a need to use advanced software engineering methods and tools to handle its complexity, size and criticality. Software systems are difficult to understand because of their non-linear nature; a one bit error can bring an entire system down, or a much larger error may do nothing. Moreover, many errors come from design flaws or requirements specification [3].

Basic vehicles already run large amounts of software with tight constraints concerning real-time processing, failure rate, maintainability and safety. In order to avoid design flaws or an unbalanced representation of requirements in the final product, the software's evolution towards autonomy must be well managed. Since adding cognitive intelligence to vehicles leads to new software components deployed on existing platforms, a clear mapping between functional goals and software components is needed.

This research was funded by NWO as part of the i-CAVE project.

Software architecture was introduced as a means to manage complexity in software systems and help assess functional and non-functional attributes, before the build phase. A good architecture is known to help ensure that a system satisfies key requirements in areas such as functional suitability, performance, reliability or interoperability [4].

The goal of this paper is to design a functional software architecture for fully autonomous vehicles. Existing literature takes a descriptive approach and presents past experiments with autonomous driving or implementations specific to limited domains (e.g. winning a competition). The architectural solutions are often an after-math of building or evolving an autonomous vehicle and not the result of a clear software development life-cycle. A major issue of this approach is that requirements can not be traced with respect to functional components and several components group most functionality. Therefore, without inside knowledge, it is often not straight forward to adopt the proposals.

In this paper we take a prescriptive approach driven by standard requirements. We use requirements from the *Society of Automotive Engineers* (SAE) J3016 standard, which defines multiple levels of driving automation and includes functional definitions for each level. The goal of SAE J3016 is to provide a complete taxonomy for driving automation features and the underlying principles used to evolve from none to full driving automation. At the moment of writing this paper, it is the only standard recommended practice for building autonomous vehicles.

The term *functional architecture* is used analogously to the term *functional concept* described in the ISO 26262 automotive standard [5, 1]: a specification of intended *functions* and necessary *interactions* in order to achieve desired behaviors. Functional architecture design corresponds to the second step in the V-model [5, 6], a software development life cycle imposed by the mandatory compliance with the ISO 26262 automotive standard.

We follow the methodology described by Wieringa [7] as the design cycle; a subset of the engineering cycle which precedes the solution implementation and implementation evaluation. The design cycle includes designing and validating a solution for given requirements.

The rest of this paper develops along the following lines. In Section II we introduce background information. In Section III we present the requirements and the reasoning process

that lead to a solution domain. The functional components are introduced in Section IV, followed by component interaction patterns in Section V and a general discussion in Section VI. In Section VII we compare the proposal with related work and conclude with future research in Section VIII.

II. Background

The development of automotive systems is distributed between manufacturers – aka *Original Equipment Manufacturers* (OEM) – and component suppliers. This leads to a distributed software development life cycle, where the OEM often play the role of technology integrators. The same distributed paradigm applies to component distribution and deployment inside a vehicle: embedded systems called *Electronic Control Units* (ECU) are deployed in vehicles in order to enforce digital control of functional aspects such as steering or brakes.

To understand the vehicle automation process, we first introduce the most important terms as defined in SAE J3016 [8]:

- *Dynamic Driving Task* (DDT) - real-time operational and tactical functions required to operate a vehicle, excluding strategic functions such as trip scheduling or route planning. DDT is analogous to all processes needed to drive a car on a given route.
- Driving automation system - hardware and software systems collectively capable of performing part or all of the DDT on a sustained basis. Driving automation systems are usually composed of design-specific functionality called *features*.
- *Operational Design Domains* (ODD) - the specific conditions under which a given driving automation system or feature is designed to function.
- DDT feature - a design-specific functionality at a specific level of driving automation with a particular ODD.

Besides hardware constraints, full vehicle automation involves DDT automation in all ODDs, by developing a driving automation system. Recursively, driving automation systems are composed of design-specific features. In this sense, complete vehicle automation is regarded as developing, deploying and orchestrating enough DDT features in order to satisfy all conditions (ODD) in which a human driver can operate a vehicle.

The transfer of total control from humans to machines is classified by the SAE as a stepwise process on a scale from 0 to 5, where 0 involves no automation and 5 means full-time performance by an automated driving system of all driving aspects, under all roadway and environmental conditions [8].

The classification is meant to clarify the role of a human driver, if any, during vehicle operation. The first discriminant condition is the environmental monitoring agent. In the case of no automation up to partial automation (levels 0-2), the environment is monitored by a human driver, while for higher degrees of automation (levels 3-5), the vehicle becomes responsible for environmental monitoring. Another discriminant criteria is the responsibility for DDT fall-back mechanisms. For low levels of automation (0-3) a human driver needs to take control in case of a system fall, while for levels 4-5 a human is no longer needed.

III. Requirements and Rationale

The process of functional architecture design starts by developing a list of functional components and their dependencies [3]. Towards this end, SAE J3016 defines three classes of components:

- Operational - basic vehicle control,
- Tactical - planning and execution for event or object avoidance and expedited route following, and
- Strategic - destination and general route planning.

Each class of components has an incremental role in a hierarchical control structure which starts from low level control, through the operational class and finishes with a high level overview through the strategic class of components. In between, the tactical components handle trajectory planning and response to traffic events.

Later, the SAE definition for DDT specifies, for each class, the functionality that must be automated in order to reach full autonomy (level 5):

- Lateral vehicle motion control via steering (operational).
- Longitudinal vehicle control via acceleration and deceleration (operational).
- Monitoring of the driving environment via object and event detection, recognition, classification and response preparation (operational and tactical).
- Object and event response execution (operational and tactical).
- Maneuver planning (tactical).
- Enhanced conspicuity via lighting, signaling and gesturing, etc. (tactical).

Moreover, an autonomous vehicle must ensure DDT fall-back and must implement strategic functions, not specified in the DDT definition. The latter consists of destination planning between two points provided by a human user.

The automation of a task resembles a control loop which receives input from sensors, performs some reasoning and controls vehicle behavior through actuators [9]. The automation of complex tasks requires a deeper (semantic) understanding of sensor data in order to generate higher order decisions or plans. However, the loop behavior is preserved. The analogy holds for the SAE classification of functional components, where components falling in the operational class require less complicated semantics to reason upon sensor data and perform vehicle control, while tactical and strategic components require in depth understanding of the vehicle surroundings through object and event recognition in order to generate high level plans and decisions.

Architecture design for autonomous vehicles is analogous to the design of a real-time, intelligent, control system. There is considerably literature from the fields of robotics and artificial intelligence which puts forward reference architectures for such systems [10, 11, 12, 13, 14, 15, 16]. The proposals revolve around *behavior* or *knowledge* based systems, where

knowledge-based systems maintain an internal state of the environment and behavior-based systems do not [10].

On the same path, the literature distinguishes between *deliberative* and *reactive* systems, where deliberative systems *reason* upon an internal representation of the environment and *reactive* system fulfill goals through *reflexive* reactions to environment changes [12, 13, 14, 15].

However, an architecture for autonomous vehicles must be capable to represent both reactive and deliberative components in a single artifact, so it can both plan for the future and react, in the least of time, to unexpected events. In both cases, the use of internal state must be minimized as much as possible, towards an increase in performance and efficiency. One of the most popular reference architectures to propose a balance between reactive and deliberative components was introduced by Gat et al. [12]. Here, the functional components are classified based on their memory and knowledge about the internal state in: *no knowledge*, knowledge of the *past*, or knowledge of the *future*. However, the model does not specify how, or if, the knowledge can be shared between components and if one component can hold knowledge about both the past and the future.

A better proposal, that bridges the gap between reactive and deliberative components, is the NIST *Real Time Control Systems* (RCS) reference architecture by Albus [11]. This architecture does not separate components based on memory, but builds a hierarchy based on semantical knowledge. Thus, components lower in the hierarchy have limited semantic understanding and can generate inputs for higher components, deepening their semantic knowledge and understanding. This makes RCS a better mapping on the SAE classification of functional components discussed in Section III. Moreover, RCS has no temporal limitations for a component's knowledge. One can hold static or dynamic information about past, present or future. We select this reference architecture as a baseline for our proposal because of its ability to describe both the autonomous vehicle as a whole and the individual driving automation features.

At the heart of the RCS control loop is a representation of the external world, the world model, which provides a site for data fusion, acts as a buffer between perception and behavior, and supports both sensory processing and behavior generation. Sensory processing performs the functions of windowing, grouping, computation, estimation, and classification on input from sensors. World modeling maintains knowledge in the form of images, maps, events or relationships between them. Value judgment provides criteria for decision making. Behavior generation is responsible for planning and execution of behaviors [11].

Albus proposed the design for a node in a hierarchical control structure, where perception at lower level nodes generates inputs for higher level nodes, thus increasing the level of abstraction and cognition. However, in the automotive context it, is not always a requirement for lower node's output to be input of higher nodes. It may be that nodes at different hierarchical levels process the same sensor data. Moreover, low level control loops such as longitudinal or lateral control, which need basic sensor input to execute their tasks, need no world modeling capacity. Their structure is thus reduced to flat, simple, control loops with multiple processing steps (e.g. noise filtering, data comparison).

From an architectural point of view, a flat stream of data which passes through different, individual, processing steps can be represented as a *pipe-and-filter* pattern [17]. The pattern divides a process into several sequential steps, connected by the data flow - the output data of a step is the input to the subsequent step. Each processing step is implemented by a *filter* component.

In its pure form, the pipe-and-filter pattern can only represent sequential processes. For hierarchical structures, a variant of the pattern, called *tee-and-join pipeline*, is used [17]. In this paradigm, the input from sensors is passed either to a low level pipeline corresponding to a low level control loop, to a higher level pipeline, or both.

An alternative to the chosen architectural style is a *component-based* architecture. This style specifies that all components can be interchangeable and independent of each other. While it is a popular choice for functional software architecture description [3], it reveals no information about functional hierarchies. Another alternative, the *layered* architectural style, would limit component communication because the commands have to flow strictly from the higher layer to the lower one, thus grouping all decisions at a layer.

IV. Functional Components

We start by introducing the functional components and, in Section V, discuss interaction patterns. Figure 1 depicts the functional components that satisfy SAE J3016 requirements for fully autonomous vehicles. The data flows from left to right; from the sensors abstraction to actuator interfaces, simulating a closed control loop. The figure represents three types of entities: functional components (blue boxes), classes of components (light gray boxes), and sub-classes of components (dark gray boxes).

The proposal maps onto the SAE classification of functional components, introduced in Section II, in the following way: *vehicle control* and *actuators interface* class of components correspond to SAE operational functions, the *planning* class of components corresponds to SAE tactical functions, and the *behavior generation* class maps to both strategic and planning class of functions.

It is an instantiation of the RCS reference architecture where the *sensors abstraction* and *sensor fusion* classes map to RCS sensor processing, the *world modeling* contains the real time knowledge data analogous to the world modeling component in RCS and the *behavior generation*, *planning*, *vehicle control* and *actuators interface* classes map to the behavior generation module in RCS.

Two orthogonal classes, corresponding to *data management* and *system and safety management*, are depicted because they represent cross-cutting concerns: data management components implement long term data storage and retrieval, while

system and safety management components act in parallel of normal control loops and represent DDT fall-back mechanisms or other safety concerns.

In the following subsections each class of filters is discussed together with its components. The last sub-section discusses the relation with middle-ware solutions and AUTOSAR.

A. Sensor Abstractions

Sensor abstractions provide software interfaces to hardware sensors, possible adapters and conversion functionality needed to interpret the data. We distinguish two classes of sensors and, respectively, of abstractions: (1) sensors that monitor the internal vehicle state or dynamic attributes (inertial measurements, speed, etc.) and (2) sensors that monitor the external environment as required by the SAE requirements.

Environmental monitoring can be based on RADAR, LIDAR and camera technologies. In the case of cooperative driving, communication with other traffic participants is realized through *vehicle-to-everything* (V2X). Global positioning (GPS) is required to localize the vehicle in a map environment or to generate global routes and is therefore represented as a separated functional component.

All abstractions concerning the internal vehicle state are grouped into one functional component, because the choice is specific to each OEM.

B. Sensor Fusion

Multi-sensor environments generate large volumes of data with different resolutions. These are often corrupted by a variety of noise and clutter conditions which continually change because of temporal changes in the environment. Sensor fusion combines data from different, heterogeneous, sources to increase accuracy of measurements.

The functional components are chosen with respect to SAE requirements for object and event detection, recognition, and classification. We distinguish between *static and dynamic objects* (e.g. a barrier, pedestrians) and *road objects* (e.g. traffic lights) because they have different semantic meaning. Moreover, *local positioning* is needed to position the vehicle relative to the identified objects and *global positioning* is needed for strategic functionality.

Through sensor fusion, a processing pipeline gathering information from various sensors such as RADAR and camera can be used to classify an object, determine its speed, and add other properties to its description. The distinction between the external environment and the internal state of a vehicle is preserved in Figure 1: the first four components process data related to the external environment, while the internal state is represented by the last functional component.

C. World Model

The world model represents the complete picture of the external environment as perceived by the vehicle, together with its internal state. Data coming from sensor fusion is used together with stored data (e.g. maps) in order to create a complete representation of the world.

As in RCS architecture, the world model acts as a buffer between sensor processing and behavior generation. Components in this class maintain knowledge about images, maps, entities and events, but also relationships between them. World modeling stores and uses historical information (from past processing loops) and provides interfaces to query and filter its content for other components. These interfaces, called *data sinks*, filter content or group data for different consumers in order to reveal different insights. One example heavily used in the automotive industry is the bird's eye view. However, the deployed data sinks remain OEM-specific.

D. Behavior Generation

Behavior generation is the highest cognitive class of functions in the architecture. Here, the system generates predictions about the environment and the vehicle's behavior. According to the vehicle's goals, it develops multiple behavior options (through *behavior generation*) and selects the best one (*behavior selection*). Often, the vehicle's behavior is analogously to a *Finite State Machine* (FSM). The behavior generation module develops a number of possible state sequences from the current state and the behavior reasoning module selects the best alternative. Complex algorithms from *Reinforcement Learning* (RL) use driving policies stored in the knowledge database to reason and generate a sequence of future states. Nevertheless, the functional loop is consistent: at first a number of alternative behaviors are generated, then one is selected using an objective function (or policy).

A vehicle's goal is to reach a given destination without any incident. When the destination changes (through a *Human Machine Interface* (HMI) input), the *global routing* component will change the goal and trigger new behavior generation. These components correspond to the SAE strategic functions.

E. Planning

The planning class determines each maneuver an autonomous vehicle must execute in order to satisfy a chosen behavior. The *path planning and monitoring* component generates an obstacle free trajectory and composes the trajectory implementation plan from *composition functions* deployed on the vehicle. It acts like a supervisor which decomposes tasks, chooses alternative methods for achieving them, and monitors the execution. The need to re-use components across vehicles or outsource the development leads the path to compositional functions. Examples of such functions are: lane keeping systems or automated parking systems (all, commercial of-the-shelf deployed products). Compositional functions represent an instantiation of the RCS architecture; they receive data input from sensor fusion or world modeling through data sinks, judge its value and act accordingly, sending the outputs to vehicle control. Path planning and monitoring act as orchestrators which decide what functions are needed to complete the trajectory and coordinate them until the goal is fulfilled or a new objective arrives. For vehicles up and including level 4, which cannot satisfy full autonomous driving in all driving conditions, the control of the vehicle must be handed to a

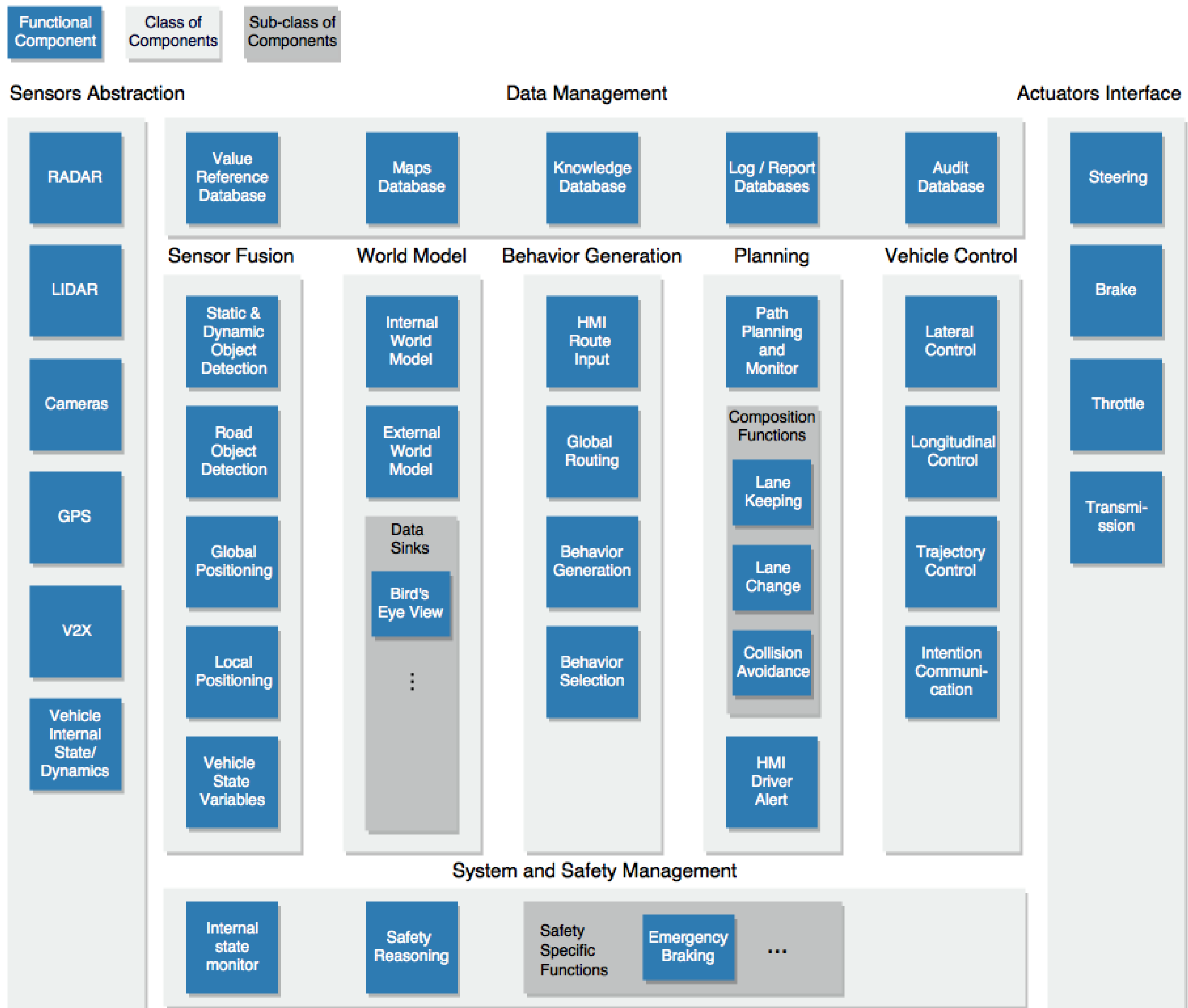

Fig. 1: Proposed functional architecture, part I: functional components.

trained driver in case a goal can not be fulfilled. Therefore, this class includes a *driving alert* HMI component.

F. Vehicle Control

Vehicle control is essential for guiding a car along the planned trajectory. The general control task is divided into lateral and longitudinal control, reflecting the SAE requirements for operational functions. This allows the control system to independently deal with vehicle characteristics (such as maximum allowable tire forces, maximum steering angles, etc.) and with safety-comfort trade-off analysis. The *trajectory control* block takes a trajectory (generated at the path planning and monitoring level) as input and controls the lateral and longitudinal modules. The trajectory represents a future desired state given by one of the path planning compositional functions. For example, if a lane-change is needed, the trajectory will represent the desired position in terms of coordinates and orientation, without any information about how the acceleration, steering or braking will be performed. The *longitudinal control* algorithm receives the target longitudinal state (such as brake until 40 km/h) and decides if the action will be performed by accelerating, braking, reducing throttle, or using the transmission module (i.e. engine braking). The *lateral control* algorithm computes the target steering angle given the dynamic properties of a vehicle and the target trajectory. If the trajectory includes a change that requires signaling, the communication mechanisms will be triggered through the *intention communication* module.

G. Actuator Interfaces

The actuator interface modules transform data coming from vehicle control layer to actuator commands. The blocks in Figure 1 represent the basic interfaces for longitudinal and lateral control.

H. Data Management

Autonomous vehicles handle huge amounts of data. In spite of the fact that most data requires real-time processing, persistence is also needed. These concerns are represented using the data management class of components. Global localization features require internal *maps storage*; intelligent decision and pattern recognition algorithms require trained models (*knowledge database*); internal state reporting requires advanced logging mechanisms (*logging database*). The *logging-report* databases are also used to store data needed to later tune and improve intelligent algorithms. Moreover, an *audit database* keeps authoritative logs (similar to black boxes in planes) that can be used to solve liability issues. In order to allow dynamic deployable configurations and any change in reference variables (e.g. a calibration or a decisional variable) a *value reference* database is included.

I. System and Safety Management

The system and safety management block handles functional safety mechanisms (fault detection and management) and traffic safety concerns. It is an instantiation of the separated safety pattern [18] where the division criteria split the control system from the safety operations. Figure 1 only depicts components that spot malfunctions and trigger safety behavior (*internal state monitor*, equivalent to a watch dog), but not redundancy mechanisms. The latter implement partial or full replication of functional components. Moreover, *safety specific functions* deployed by the OEM to increase traffic safety are distinctly represented. At this moment they are an independent choice of each OEM.

As the level of automation increases, it is necessary to take complex safety decisions. Starting with level 3, the vehicle becomes fully responsible for traffic safety. Therefore, algorithms capable of full safety reasoning and casualty minimization are expected to be deployed. While it is not yet clear how *safety reasoning* will be standardized and implemented in future vehicles, such components will soon be mandatory [19]. An overview of future safety challenges autonomous vehicles face is illustrated in [20]. With respect to the separated safety pattern, in Figure 1 safety reasoning components are separated from behavior generation.

J. AUTOSAR *(AUTOSAR) Context*

AUTOSAR is a consortium between OEMs and component suppliers which supports standardization of the software infrastructure needed to integrate and run vehicle's software. This paper does not advocate for or against AUTOSAR. The adoption and use of AUTOSAR is OEM-specific. In the AUTOSAR context, the functional components in Figure 1 represent AUTOSAR *software components*. The interfaces between components can be specified through AUTOSAR's standardized interface definitions.

V. Interaction between Components

As mentioned in Section III, the components in Figure 1 act as a hierarchical control structure, where the level of abstraction and cognition increases with the hierarchical level, mapping on the SAE classification of functional components. Components lower in the hierarchy handle less complex tasks such as operational functions, while higher components handle planning and strategic objectives (e.g. global routing or trajectory planning).

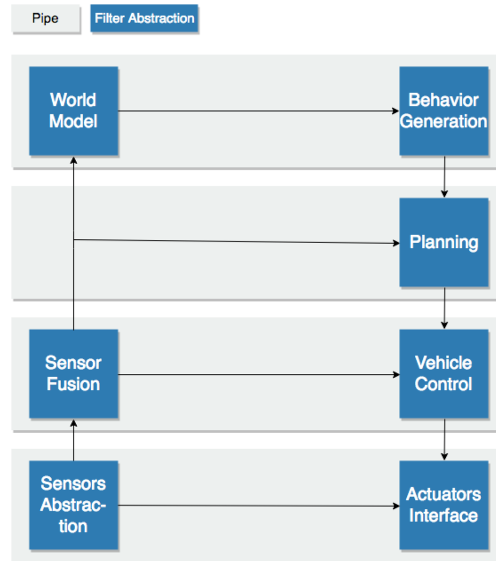

Fig. 2: Proposed functional architecture, part II: hierarchical control structure using tee-and-join pipelines pattern.

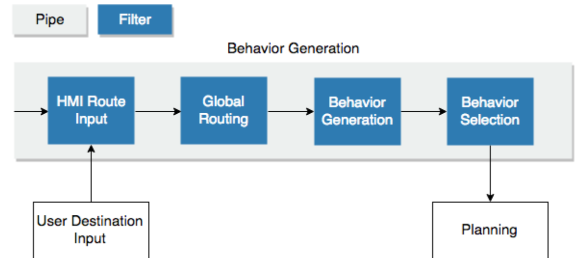

Fig. 3: Proposed functional architecture, part III: component interaction at class level. The behavior generation process.

We propose the use of pipe-and-filter pattern for component interactions in flat control structures (same hierarchical level) and the use of tee-and-join pipelines to represent the hierarchy. In a hierarchical design, lower level components offer services to adjacent upper level components. However, the data inputs are often the same. A high level representation of the system, through the tee-and-join pipelines pattern is illustrated in Figure 2. The gray boxes represent processing pipelines and the blue ones represent components classes.

For each component class, a process is analogous to a pipeline. As example, once a user communicates a final destination, the behavior generation process starts. This example is illustrated in Figure 3, where upon receiving a destination at the *HMI input filter*, the *global routing* filter forwards a route to the *behavior generation* filter. This filter breaks down the route in actions that have to be taken by the vehicle in order

to reach the destination. The actions are analogous to states in a FSM. Often, there are several paths between two states. Further on, the *behavior selection* component will select the best path between two states and forward it to the *planning* process.

Moreover, messages received at component level have to be prioritised. For example, an actuator interface can receive commands from the longitudinal control component or a safety specific function (e.g. emergency braking). In order to decide which command to execute first, the messages must contain a priority flag. Since this functionality is dependent on the component's interface and specific to OEM, this discussion is limited.

VI. Discussion

Software architecture evaluation is an expert-driven process, based on scenario evaluation and requirements tracing with respect to stakeholder concerns [21]. This process evaluates both the functional suitability of an architecture and non functional properties such as performance or maintainability [22]. In this paper we are only interested in functional suitability and completeness with respect to SAE J3016 requirements. However, we further discuss two important aspects: the position of the proposed architecture with respect to (1) the automotive software development life cycle and (2) the ISO 26262 standard that regulates functional safety. Later, in Section VII, we provide a comparison with existing literature. Expert based validation is the subject of our future study and is meant to reflect non functional properties of the proposal.

A. Incremental development and component reuse

The SAE classification presented in Section II shows an incremental transition from partially automated to fully autonomous vehicles. The functional division of software components should respect this incremental transition. Moreover, the OEM software development life-cycle and preference for outsourcing must be taken into account.

As mentioned in Section II, DDT automation is analogous to deploying and orchestrating enough driving automation features in order to satisfy all driving conditions (ODD) in which a human can drive. This assumption employs two development paths:

1) the deployment of new software components specific to new DDT features or
2) updating a driving feature with enhanced functionality.

In Figure 1, new DDT features represent new compositional functions specific to path planning. The use of composition functions enables incremental and distributed development at the cost of increased complexity for *path planning and monitor*. These components can be commercial-of-the-shelf products that can easily be outsourced to tier one suppliers.

Behavior generation improvements are solved through knowledge database updates. The V2X component interfaces with the external world, therefore, updates can be pushed through this component. In most cases, the updates will target the knowledge or value reference databases.

B. Functional safety

The automotive industry has high *functional* safety constraints imposed by the mandatory adherence to ISO 26262 [5]. The objective of functional safety is to avoid any injuries or malfunctions of components in response to inputs, hardware or environmental changes. Error detection and duplication of safety critical components are mechanisms suggested by ISO 26262. In this proposal, we represent the functional component specific to error detection, however, omit to represent any redundancy or duplicated components.

We also aim to fulfill a gap in the ISO 26262 standard, with regards to autonomous vehicles: safety reasoning [20]. To this moment it is not clear how autonomous vehicles will behave in case an accident can not be avoided and which risk to minimize. However, it is expected for future safety standards to include specification for safety behavior. Towards this end, the proposed architecture features a safety reasoning component.

VII. Related work

We focus on literature proposing functional and reference architectures starting with level 3, since level 2 vehicles only automate lateral and longitudinal control. A historical review of level 2 systems is presented in [23].

The only reference architecture for fully autonomous (level 5) vehicles was introduced by Behere et al. [1]. In this proposal, the authors make a clear distinction between cognitive and vehicle platform functionality, similar to the classification in tactical and operational SAE classes. The decision and control block [1] is responsible for trajectory reasoning, selection and implementation, equivalent to the behavior generation and planning class of components from Figure 1. Yet it is not clear how this block handles all functionality, leading to a rough representation of functional classes. It is also interesting to observe that HMI components are ignored.

Other work in this field focused primarily on systems developed for autonomous driving competitions or other constrained experiments. Introducing a project which attended one of the first competitions organized by DARPA, Montmerlo et al. [24] show a layer-based architectural model based on sensor interface, perception, navigation, user and vehicle interfaces. In this model localization features are embedded in perception together with laser object detection. No object recognition or classification was needed in this competition. The model represents operational and tactical functions through navigation components, but excludes strategic functions.

Jo et al. [25, 26] present their experience from an autonomous vehicle competition held in Korea. The proposal comes one step closer to a general architecture, given broader competition goals. The model contains sensor abstractions, fusion, behavior and path planning, vehicle control and actuator interfaces. In this regard, it represents similar concerns to Figure 1, without world modeling and HMI route inputs. Instead, the behavior planning component integrates data coming from sensors in order to generate an execution plan. Since the goal of the competition was limited, both localization and behavior

reasoning components are restricted (a finite state machine with only 8 possible states that can stop for a barrier, detect split road, etc.). The artifact successfully represents operational and tactical functions. Moreover, Jo et al. divide, for the first time, the concerns from behavior and from path planning, thus obtaining several levels of cognition and control. The study also reveals important details for in-vehicle ECU deployment and a mapping to AUTOSAR.

An important contribution from industry research is the work of Ziegler et al. [27] at Mercedes Benz. Although its purpose is not to introduce a general functional architecture, the system overview reveals similar functional requirements. It features object recognition, localization, motion planning and vehicle control, analogous to behavior generation, planning and vehicle control in Figure 1. Once again, the concerns for behavior generation are separated from path and trajectory planning, and grouped under motion planning. Another important contribution is the representation of data storage functionality for digital maps and reactive components such as emergency braking.

Overall, we observe two approaches in the literature: (1) a high level overview of system components or (2) proofs-of-concept from experiments with autonomous features or competition with limited operational domain. This paper takes one step further and considers a fine-grained functional decomposition with respect to the automotive software development life cycle. Moreover, data concerns are central to the proposal, both for long term storage and fast update of reasoning and cognitive models. We advocate advanced logging mechanisms with specific architectures and data models that can help in fast identification of malfunctions and could be later used to improve learning algorithms.

VIII. Conclusions and Future Research

We have presented a functional software architecture for fully autonomous vehicles. Since the automotive industry is highly standardized, we follow the functional requirements from an automotive standard which defines multiple levels of driving automation and includes functional definitions for each level. During the architecture design, we aim to respect the incremental development process of autonomous vehicles and the distributed software development process specific to the automotive industry. The final artifact represents an automotive specific instantiation of the NIST RCS reference architecture for real-time, intelligent, control systems. We use the pipe-and-filter architectural pattern for component interaction and the tee-and-join pipeline pattern to represent a hierarchical control structure. Software architecture evaluation is often an expert-driven process. One downside of our methodology is that we do not call for expert validation. Future work includes, at first, refinement through expert opinion. Later steps consider component interface design, a choice for hardware architecture, functional component distribution across ECUs, component distribution inside local networks in order to satisfy security requirements and an instantiation in the i-CAVE project.

References

[1] S. Behere and M. Törngren, "A functional reference architecture for autonomous driving," *Information and Software Technology*, vol. 73, pp. 136–150, 2016.

[2] M. Broy, "Challenges in automotive software engineering," in *International Conference on Software Engineering (ICSE'06)*, pp. 33–42, ACM, 2006.

[3] M. Staron, *Automotive Software Architectures*, vol. 1. Springer International Publishing, 2017.

[4] D. Garlan, "Software architecture: a roadmap," in *Conference on The Future of Software Engineering (ICSE'00)*, pp. 91–101, ACM, 2000.

[5] International Organization for Standardization (ISO), "ISO standard 26262:2011 Road vehicles - Functional safety," 2011.

[6] N. B. Ruparelia, "Software development lifecycle models," *ACM SIGSOFT Software Enginering Notes*, vol. 35, no. 3, pp. 8–13, 2010.

[7] R. Wieringa, "Design science methodology: principles and practice," in *International Conference on Software Engineering (ICSE'10)*, pp. 493–494, ACM, 2010.

[8] Society of Automotive Engineers (SAE), "J3016," *SAE international taxonomy and definitions for terms related to on-road motor vehicle automated driving systems, levels of driving automation*, 2014.

[9] R. Horowitz and P. Varaiya, "Control design of an automated highway system," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 913–925, 2000.

[10] P. Maes, "Behavior-based artificial intelligence," in *Proceedings of the Fifteenth Annual Meeting of the Cognitive Science Society*, pp. 74–83, 1993.

[11] J. S. Albus, "The NIST real-time control system (RCS): an approach to intelligent systems research," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 9, no. 2-3, pp. 157–174, 1997.

[12] E. Gat and R. P. Bonnasso, "On three-layer architectures," *Artificial intelligence and mobile robots*, vol. 195, p. 210, 1998.

[13] N. Muscettola, G. A. Dorais, C. Fry, R. Levinson, and C. Plaunt, "Idea: Planning at the core of autonomous reactive agents," in *NASA Workshop on Planning and Scheduling for Space*, 2002.

[14] K. Konolige, K. Myers, E. Ruspini, and A. Saffiotti, "The Saphira architecture: A design for autonomy," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 9, no. 2-3, pp. 215–235, 1997.

[15] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, "The CLARAty architecture for robotic autonomy," in *IEEE Aerospace Conference*, vol. 1, pp. 121–132, IEEE, 2001.

[16] "An architectural blueprint for autonomic computing," tech. rep., IBM, 2006. White paper. Fourth edition.

[17] F. Buschmann, K. Henney, and D. C. Schmidt, *Pattern-oriented Software Architecture*, vol. 5. John Wiley & Sons, 2007.

[18] J. Rauhamäki, T. Vepsäläinen, and S. Kuikka, "Functional safety system patterns," in *Proceedings of VikingPLoP*, Tampere University of Technology, 2012.

[19] J.-F. Bonnefon, A. Shariff, and I. Rahwan, "The social dilemma of autonomous vehicles," *Science*, vol. 352, no. 6293, 2016.

[20] A. Serban, E. Poll, and J. Visser, "Tactical safety reasoning. a case for autonomous vehicles.," *Proceedings of Ca2V Workshop*, 2018.

[21] R. Kazman, G. Abowd, L. Bass, and P. Clements, "Scenario-based analysis of software architecture," *IEEE Software*, vol. 13, no. 6, pp. 47–55, 1996.

[22] L. Dobrica and E. Niemela, "A survey on software architecture analysis methods," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 638–653, 2002.

[23] A. Khodayari, A. Ghaffari, S. Ameli, and J. Flahatgar, "A historical review on lateral and longitudinal control of autonomous vehicle motions," *ICMET*, pp. 421–429, 2010.

[24] M. Montemerlo, J. Becker, S. Bhat, *et al.*, "Junior: The Stanford entry in the urban challenge," *Journal of Field Robotics*, vol. 25, no. 9, pp. 569–597, 2008.

[25] K. Jo, J. Kim, D. Kim, C. Jang, and M. Sunwoo, "Development of autonomous car - part I," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 12, pp. 7131–7140, 2014.

[26] K. Jo, J. Kim, D. Kim, C. Jang, and M. Sunwoo, "Development of autonomous car - part II," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 8, pp. 5119–5132, 2015.

[27] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, T. Dang, U. Franke, N. Appenrodt, C. G. Keller, *et al.*, "Making Bertha drive - an autonomous journey on a historic route," *IEEE ITS Magazine*, vol. 6, no. 2, pp. 8–20, 2014.