Chapter 3
System Architectures for Automated Vehicle Guidance Concepts

Felix Lotz

3.1 Introduction and Motivation

Analyzing the current landscape of automotive engineering and the respective research domains leads to the conclusion that vehicle automation is becoming a key technology for the near future. A large part of today's automotive innovation derives from advanced driver assistance systems (ADAS) which are being developed essentially to make driving safer, more comfortable and economically more efficient.

The state of technology counts about 60 available assistance systems for passenger vehicles which help to prevent traffic accidents from happening (Barrios et al. 2007, pp. 9–12). To enable this large number of assistance functionalities, modern cars contain up to 80 electronic control units (ECUs) and a variety of network platforms (Broy et al. 2006, p. V). Furthermore, when looking at an ADAS-Roadmap (Winner and Weitzel 2012, p. 666) it can be assumed that the number of assistance functionalities will increase even more in the future, and will most likely lead to more and more complex systems. The situation is becoming increasingly complicated because of the rising number of manufacturer vehicle models, platforms and powertrain concepts, including different engines and their degree of electrification, in order to fill market gaps and satisfy customer demands for individualization and individual mobilization.

Faced with this variety and complexity in automotive systems design, the engineer and system architect have to deal with challenging problems which cannot be solved through the linear addition of functionalities and control units into already existing architectures, and this not only because of hardware packaging problems (Reichart and Bielefeld 2012, p. 84). The functional variation and diversity mean that the time and effort of system application as well as the costs of corresponding testing and validation will probably increase and can lead to an uneconomical development

F. Lotz (🖂)
Institute of Automotive Engineering, Technische Universität Darmstadt, Petersenstraße 30, 64287 Darmstadt, Germany
e-mail: lotz@fzd.tu-darmstadt.de

process. Another aspect is the risk of the visual and mental overload of the driver, who has to interact with and know the system boundaries of the individual assistance systems (Kauer et al. 2010, p. 1214).

From a functional viewpoint, a higher grade of vehicle automation opens the opportunity to incorporate existing systems into an integrated, functionally combined assistance approach and hence provides a possible solution to the problems described above. Presently, many research projects initiated by industry, academia and also the military concern the development of semi- and fully automated driving (cf. Sect. 3.3.1). The latter, often also referred to as autonomous driving, has to address extensive technical and social requirements. Besides the challenge of machine-based perceiving and understanding of a highly complex traffic environment like inner-city driving, to date there is still no valid metric to give a proof of safety, which is required for legal registration and in order to resolve the issue of manufacturer liability (Winner and Weitzel 2012, p. 661). A recent report by the German Federal Highway Research Institute (BASt) comes to the conclusion that highly and fully automated driving, in which the driver has not to control the automated system permanently, is inconsistent with today's German regulatory law because the driver is obligated to show permanent caution in road traffic (Gasser et al. 2012, p. 3).

A legally acceptable intermediate solution for automation concepts could be the vehicle guidance paradigm of "cooperative automation". Cooperative automation can be characterized as an intensive, cooperative interaction between the driver and the automated system based on mutual information, recommendation and approval in order to encounter the driving task more effectively than without one another (Löper et al. 2008; Hakuli et al. 2012, p. 641). For example, in the cooperative assistance concept "Conduct-by-Wire", the system relieves the driver of the vehicle stabilization task and yet enables him/her to stay in the control loop by communicating with the car at maneuver level. Therefore, the driver is still holding responsibility as demanded by law (Hakuli et al. 2011, p. 221; Geyer 2013).

Besides the approach of combining assistance functionalities by an integrated vehicle automation concept, a very important tool to manage the overall system complexity is the system architecture. Not only does it bridge the gap between requirements analysis and implementation by defining the structural layout of the automotive system, but it also accounts for an efficient development process allowing risks to be identified at an early stage of development, enabling the division of labor within a project group and promoting a mutual understanding between all stakeholders (Posch et al. 2007, pp. 14–15).

However, a well-designed system architecture does not by itself generate a successful overall development process. Similar to the functional integration already described above, the upcoming challenges in automotive engineering also require an integrated development process which addresses requirements engineering, the design of test and validation strategies and also tool development. Independent of specific applications, in this chapter such an integrated development process is referred to as 'automotive systems engineering'.

In the following sections we will specify the role of the system, and particularly the software architecture within an integrated development process based on the interdisciplinary research project PRORETA 3. This is the latest project within the PRORETA long-term research initiative between the TU Darmstadt and Continental AG. The objective is to design a virtual safety corridor that prevents accidents without limiting the application to specific use cases. Furthermore, a semi-automated and maneuver-based vehicle guidance concept is being developed in order to relieve the driver from the task of vehicle stabilization. Hereby, emphasis is set on an intuitive close-to-production HMI-solution[1] and the "Safety Corridor" as a solution to combine exclusively intervening and semi-automated assistance systems. In contrast to other research projects, a concept is derived on how this collision mitigation function can be efficiently integrated in synergy into a semi-automated driving concept (Bauer et al. 2012).

One outcome of the project is to implement the PRORETA 3 assistance concept in a test vehicle in order to verify the system performance. The purpose is on the one hand to analyze the warning and intervention strategy within the "Safety Corridor" mode which aims to prevent accidents and critical situations in a 360-degree field of view, and on the other to investigate the vehicle behavior with respect to the maneuver-based, cooperative driving mode, including driver interaction. In this mode, the driver is given the opportunity to assign a vehicle maneuver, like turning left or right at an intersection or going through a roundabout, and then supervise the assistance system that automatically accomplishes the selected maneuver.

In order to achieve the functionalities described above, the test vehicle is equipped with environment sensors as shown in Fig. 3.1. Besides a GPS-receiver, attached radar sensors and their opening angle are indicated in light and dark blue, the stereo camera is indicated in green. Furthermore, the vehicle comes with an active force-feedback gas pedal, controllable electric power steering and a controllable brake booster.

The sensor- and actuator-configuration is an important part of the hardware architecture. Nevertheless, in the following section, focus is set on the software architecture since it offers greater degrees of freedom within the corresponding design process.

An overview over the state of technology for automated vehicle concepts and the respective architectures is given in Sect. 3.3.

3.2 Software Architecture Design

3.2.1 *Definitions*

According to Vogel et al. (2005, p. 46), there are many definitions for the term "Software Architecture". However, a common definition is the following (Bass et al. 2003, p. 3):

[1] **HMI**: Human-Machine Interface.

"*The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.*"

In analogy to the architecture of a building or a house, the architecture is usually displayed in form of "views", comparable to a construction plan. Similar to a building, different views of the architecture can be obtained, for example a blueprint, a plan of electricity or a plan of statics. In software architecture, the four most common (Darms 2007, p. 3) views (or viewpoints) are called "context view", "runtime view", "deployment view" and "module view" (IEEE 2000), while every view is important for a different group of stakeholders (Starke 2009, p. 15).

For the architecture design process, the most important view (and the one most published in literature) is the "module view", which represents the static structure in terms of modules, subsystems, components and the interfaces among them. It is possible to represent the module view in different levels of abstraction, whereas the most abstract view would be the context view (which, for example, shows the interaction of a user with the software) or the most detailed view, which would be the software source code itself (Starke 2009, p. 79).

In the definition of software architecture and in most publications concerning architecture design, the term "system" is often used, yet mostly without a definition. However, in order to understand the difference between the architecture and the system itself, the term "system" has to be defined. In system theory, the following definition can be found (Vogel et al. 2005, p. 52):

"*A system is an entity which maintains its existence through the mutual interaction of its parts*."

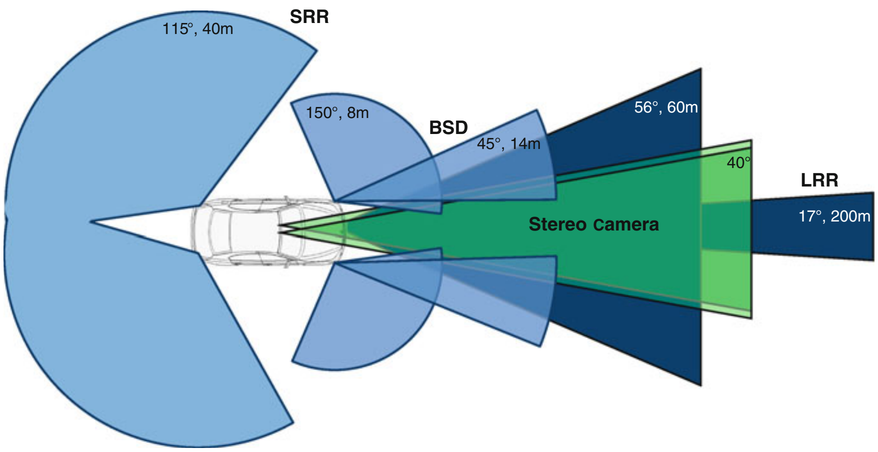

Fig. 3.1 Sensor configuration in the PRORETA 3 test vehicle (unscaled) (Bauer et al. 2012). **LRR**: Lone Range Radar Sensor; **SRR**: Short Range Radar Sensor; **BSD**: Blind Spot Detection Radar Sensor.

According to this definition, a system consists of interacting parts (or modules) and has a system border. It can also be a module for a superior system. Systems exist in order to achieve an objective and therefore exchange information with their environment (open systems) or at least stand in an energetic relation with it (closed systems) (Vogel et al. 2005, p. 53). Hence, the system architecture cannot be seen as the system itself, but as a description of the structure of the system.

Another important finding of system theory is the so-called "emergence", which means that the system possesses characteristics that are more than just the sum of its modules' characteristics. These "emergent" characteristics come into existence by module interaction (Vogel et al. 2005, p. 54). Since the architecture only describes the structure and interfaces between them, by itself it can neither describe the "holistic" system characteristics, nor verify the overall system requirements.

A driver assistance system can be associated with the definitions mentioned above. Its objective is to assist the driver of a vehicle and to this end, exchanges information with its environment, e.g. with HMIs or environmental sensors. While the assistance system consists of interacting software and hardware-modules, their structure and interfaces are described by the system architecture. The emergent characteristics of the overall assistance system have, therefore, to meet the product requirements and are an outcome of the module interaction. Concerning the test- and validation strategy, this means that it is not sufficient simply to evaluate the correct module characteristics and functionalities (which is called reductionism) but also the overall, holistic system characteristics have to be included.

3.2.2 The Role of the Architecture

The significance of the architecture design within the overall development process of a driver assistance system can best be described by the well-known V-Model as shown in Fig. 3.2. The V-Model has proven itself for the top-down and structured development of complex technical systems.

On the left-hand side, the specification branch is depicted, which ranges from the overall product requirements down to detailed software components. The specification and design of the logical, technical and software architecture take place within these development milestones. Hence, as already mentioned in Sect. 3.1, the architecture design is the connecting process between requirements definition and software implementation. A characteristic of the V-Model is that for every specification step a suitable testing strategy is defined, which forms the basis for the component and system validation branch on the right hand side (Fig. 3.2).

The meaning of the architecture for the development team involved has already been described in Sect. 3.1. For the system architect and the software programmer it has another important function: It serves as a "skeleton system" within the implementation phase (Vogel et al. 2005, p. 284). This means that first of all, "empty" software modules and their interfaces are implemented while the module functionalities are extended in later steps. This approach makes it possible to first implement

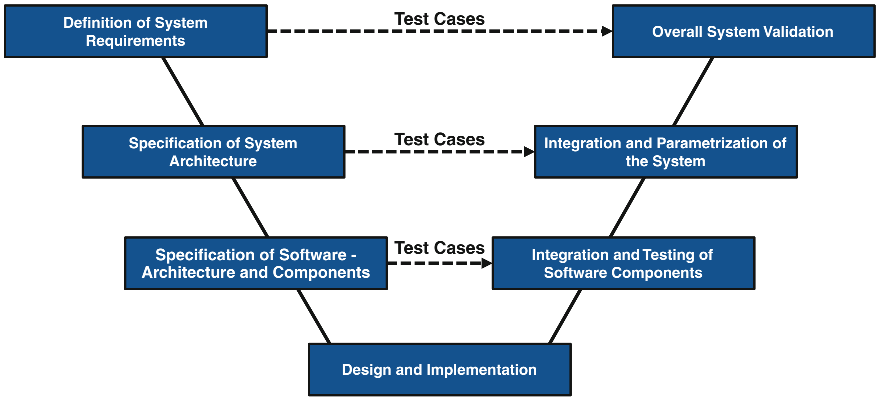
Definition of System Requirements
Test Cases
Overall System Validation
Specification of System Architecture
Test Cases
Integration and Parametrization of the System
Specification of Software - Architecture and Components
Test Cases
Integration and Testing of Software Components
Design and Implementation

Fig. 3.2 Basic structure of the V-Model, see Maurer (2012, p. 45)

the full functionality of one specific use case (the so-called “cut”) and then add the functions needed for further use cases gradually, which allows risks to be identified and simultaneously obtain a functioning system in an early development phase.

Having explained the importance of the architecture design within the development process, the following section will now focus on the procedure of architecture design.

3.2.3 *The Architecture Design Process*

Figure 3.3 shows the architecture design process, derived from Starke (2009, p. 33) and Posch et al. (2007, p. 57). The first step is the system requirements analysis which is a fundamental process not only for architecture design, but also for the whole development process (cf. V-Model in Fig. 3.2). Requirements can be divided into functional requirements, e.g. specific use cases for the assistance system, and non-functional requirements, e.g. safety-related requirements like a backup strategy during a sensor breakdown, the real-time capability or testability of the system.

In the context of the requirements analysis for driver assistance systems, a scenario-based, use-case-driven approach is proposed as successfully used within the PRORETA 3 project (step 1). The use cases are derived based on the assistance target of the system.

As an example for the cooperative automation mode (cf. Sect. 3.1) use cases are structured by means of basic driving maneuvers for vehicle automation (Tölle 1996). Also, scenario features are specified such as road geometry and the description of the behavior of other vehicles. By describing the desired system behavior within all use case scenarios, a detailed list of requirements can be derived.[2] Other advantages of such a scenario-based use-case description are the clear communication and discus-

[2] See also Vogel et al. (2005, p. 278).

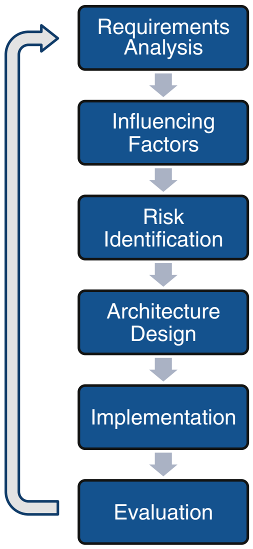

Fig. 3.3 Proposed architecture design process

sion of the system requirements within the project team, the possibility to analyze the scenarios in order to identify the specific system modules and functionalities (e.g. a traffic light detection module) and to use it as a foundation for the later testing strategy ("test cases").

Step two of the architecture design process is to identify the factors influencing the architecture. Organizational factors are, for example, the project budget, time schedule or given programming tools and frameworks (Posch et al. 2007, p. 75). Also technical influencing factors like given hardware components have a big impact on the later system architecture. Limiting factors often come from political issues and policies within the project group or project clients, not necessarily from technical limitations (Starke 2009, p. 48). As an example, an OEM supplier is more eager to set up an ADAS testing vehicle using its own brand of sensors instead of using other sensor concepts from a competitor, even if such concepts would better suit the system in development.

Further along the architecture design process (step 3), potential risks are identified by comparing the system requirements and the limiting influencing factors. As a result, strategies then have to be derived to minimize the impact of the influencing factors. Example negative influencing factors could be the limited availability of important resources like suitable development and testing tools or a strategy to set up an interface to external systems that is insensitive to software versions.

In step four, the actual software architecture is derived. For this, a literature research is necessary in order to identify the state-of-the-art of architecture design for the system in development and to pinpoint similar architecture solutions, architecture patterns or reference architectures. Section 3.3.1 describes the outcome of the research regarding vehicle automation architectures.

Architectural heuristics play an important role within the design process, since they are tools to reduce the complexity of systems with wide degrees of conceptual freedom. The term "heuristics" is used in a variety of meanings throughout architectural literature, e.g. in synonym with the terms "rule", "pattern" or "principle" (Starke 2009, p. 136). In this chapter, the term heuristics is understood as a "code of practice" obtained from the experience of system architects in the past. One common and important heuristic is modularity. The principle of modularity indicates an architecture in which the single module has a distinct and closed functionality and well-defined interfaces, so that it can be exchanged and reused arbitrarily. Modularity is of such importance because it incorporates the also imminent principles of "abstraction", "separation of concerns" and "information hiding" (Vogel et al. 2005, pp. 129–130). A good overview of architectural heuristics can be found in Vogel et al. (2005, p. 111ff.) and Starke (2009, p. 135ff).

Architectural patterns, in contrast, are defined as solutions for recurring problems in software design. In literature, they are described as a practical approach with a distinct solution, often documented in form of UML[3]-diagrams or even code fragments (Vogel et al. 2005, p. 175) and hence are on a more concrete level compared to heuristics. Example patterns are "adapters", "proxies" or "observers" (Starke 2009, p. 167ff).

Given the many examples of architecture heuristics and architecture patterns, it becomes obvious that a thorough literature research on similar systems can pay off quickly in terms of project resources. After the communication and implementation process step of architecture design (cf. the "skeleton system" in Sect. 3.2.2), the architecture design has to be constantly evaluated because requirements as well as influencing factors can change during the development phase, which is therefore characterized as an iterative process (Starke 2009, p. 25).

The evaluation process of architecture design has to answer the question of whether the designed architecture is "good enough". That raises another question: How is a good architecture characterized and is there a suitable criterion or quantity to be measured for its evaluation?

According to Starke (Starke 2009, p. 301), there are two ways to evaluate software projects. The first is to rate organizational aspects of the development process, like the amount of resources needed. However, this kind of evaluation does not provide evidence about the quality of the software product. The second way is to analyze so-called "artifacts" coming from the development process, i.e. requirement lists, architecture views or source code. Only a few of those artifacts are suitable for evaluation using a quantitative criterion, e.g. the number of lines of source code, the amount of memory capacity needed on the computer, the number of test cases needed or, for processes for example, the number of implemented features per time unit.

Therefore, an architecture itself cannot be measured quantitatively. It has to be subject of a qualitative evaluation, that means in respect of its composition and suitability (Starke 2009, p. 302). In detail, the evaluation focuses on how well a designed

[3] **UML**: Unified Modeling Language.

architecture is likely to meet the functional and non-functional requirements derived during the earlier requirement analysis process (Posch et al. 2007, pp. 12–13). Therefore, the requirements in terms of quality criteria are prioritized as to their importance for the assistance system. This results in a "utility tree" (Starke 2009, p. 310). With the help of specific use case analyses, it is then possible to evaluate in a structured way the suitability of the architecture to meet the quality criteria. This "Architecture Tradeoff Analysis Method" (ATAM) has a lot in common with the "house of quality" methodology (Bohn 2012), well-known from product development science.

Although a lot of effort can be put into the evaluation process, it cannot describe the quality of the architecture to the full extent. The missing piece of architectural quality is a phenomenon known in software science under the term "quality without a name" (QWAN). Lacking a precise scientific definition, it can best be described as the unmistakable esthetics, structure and beauty of an architecture (Vogel et al. 2005, p. 176). Although this phenomenon has not been analyzed appropriately, it can be stated that most systems that are not well-structured and are hard to understand tend to be problematic in terms of the fulfillment of their requirements. Hence, the influence of an architecture's esthetics should not be underestimated within the architecture design and evaluation process.

Based on this description of the architectural design process and associated challenges, the following section summarizes the functional and non-functional requirements for the assistance system and sums up important quality criteria to satisfy the requirements for the architecture itself within the PRORETA 3 project.

3.2.4 Requirements for Software Architectures

To be able to conduct a preliminary evaluation of the software architectures identified in the literature research in Sect. 3.3.1 and to structure important quality criteria for the PRORETA 3 software architecture, it is necessary to define a list of the most important architectural requirements. These are:

- Modularity in terms of
  - exchangeability of software modules
  - re-usability of single software modules for other ADAS
  - expendability of system functionalities
  - changeability in terms of robustness against changes and software versions
- Testability in terms of functional testing and debugging that lead to a preferably easy architecture verification and system- or component validation

The requirements above demonstrate that an important goal of the PRORETA 3 project is to develop an ADAS software architecture that is arbitrarily scalable in its functionality in respect to the level of vehicle automation, similar to the work of Darms (2007), who proposed a reference architecture for sensor data fusion for advanced driver assistance systems and Maurer (2000), who introduced a flexible architecture in terms of the degree of automation for vehicles with dynamic machine vision. The issue at hand is how such an architecture has to be composed in order to satisfy the principle of being "as substantial as possible while just as complex as necessary".

3.3 Architectures for Vehicle Automation

The previous section pointed out the need for a literature research to be carried out in order to find out how other ADAS software architectures concerning vehicle automation are structured, which modules are then necessary and how the developers proceeded within other projects.

If the research is limited to the field of driver assistance systems, the software architecture will only play a minor role in most publications.[4] Usually, the architecture itself is depicted as a low detail module view (cf. Sect. 3.2.1) and with regards to content is dwarfed by the systems' functionalities. Unfortunately, little or no explanation is given of how the architecture was derived or what the advantages of the architectural structure are. This stands in a noticeable contrast to publications concerning robotics, where architectural structures and reference architectures seem to be a keen discussion topic.

However, advanced driver assistance systems in terms of "intelligent vehicles" are not in contrast to the robotics discipline but can rather be seen as an application of robotics. This is why the outline given below is structured similar to Kortenkamp and Simmons (2008, p. 187ff), who give a good overview over robotic architectures.

3.3.1 State of Technology

3.3.1.1 Sense-Plan-Act

The sense-plan-act (SPA) paradigm was one of the first architectures applied in robotics. It was employed in mobile robots that used sensors (e.g. cameras) to perceive and build an internal model of their environment, used a planner module to generate a plan in form of a sequence of actions and then executed this plan with the help of its actuators (Kortenkamp and Simmons 2008, p. 189). This linear sequence of events has the disadvantage that the robot is not able to react to unforeseen occurrences like an oncoming obstacle, since it does not use its sensors while moving. Obviously, therefore, this sequential architecture is not suitable for a dynamic environment like road traffic.

[4] This finding is also described in Maurer (Maurer 2000, p. 15).

3.3.1.2 Reactive Architectures

Reactive Architectures emerged after the sense-plan-act paradigm reached its limits; they are characterized by quick, “reactive planning”. Reactive architectures exist of different interacting finite state machines which are called “behaviors” (Kortenkamp and Simmons 2008, p. 190). This is why this kind of architecture has embossed the term “behavioral robotics”. Behaviors access sensors and actuators during their runtime so they can react to changes in their environment quickly, just like a “reflex”. This is also why robots based on this architecture are said to have an “insect-like” behavior. An example of a mobile robot is a small robotic vacuum cleaner with the behaviors “wander around” and “collide”. Every time the robot collides with a wall, the “wander” behavior is subsumed by the “collide” behavior and the heading angle is changed randomly. After that, the “wander” behavior is active again. A well-known type of reactive architectures is the “subsumption architecture” of Brooks (1986). In this architecture, it is possible to just add more and more behaviors in order to create more complex robots.

However, this also brings disadvantages: Adding more behaviors leads to a complex state machine interaction and, since only a few behaviors can be active at the same time, results in an arbitration problem (Kortenkamp and Simmons 2008, p. 190). An easy way to deal with multiple behaviors is to categorize low- and high-level behaviors, like a hierarchy. In the vacuum cleaner example, the “collide” behavior is superior to the “wander around” behavior. Another way to arbitrate different behaviors was proposed by Arkin (1987) with the “motor-schema” based on human perception and action. In it, every behavior creates a potential field as an overlay over the current environment. The robot’s path then is derived from the sum of all of those potential fields. In order to also fulfill more complex tasks, a high-level planning module was implemented that consists of a mission planner, a navigator that calculates waypoints and a pilot that controls the different behaviors.

A further way to deal with the arbitration problem was proposed by Rosenblatt (1997) with the “Distributed Architecture for Mobile Navigation” (DAMN). It also consists of different behaviors, however, an arbitration module on top assigns different priorities to them, depending on the current situation. The behaviors then each vote for a path curvature they want the vehicle to execute in the future while being weighted with the corresponding priorities. Instead of using a mean curvature of all votes, the arbiter, in contrast to Arkins’ scheme, chooses the path with the most votes (cf. Fig. 3.4). This way, inherent problems of averaging commands, that can lead to unsatisfactory results like logical minima on potential fields, are avoided (Rosenblatt 1997, p. 343).

Another advantage of this architecture design is the directly resulting vehicle trajectory, since the curvature and therefore the maximum safe vehicle velocity is a direct outcome of the arbitration process. As an example, a successful implementation of this architecture approach was conducted by the German team “Caroline” in the DARPA urban challenge. The autonomous vehicle concept used the DAMN arbitration concept and combined it with an interrupt function to be able to also react to road intersections, roadblocks and other events (Rauskolb et al. 2008, p. 701).

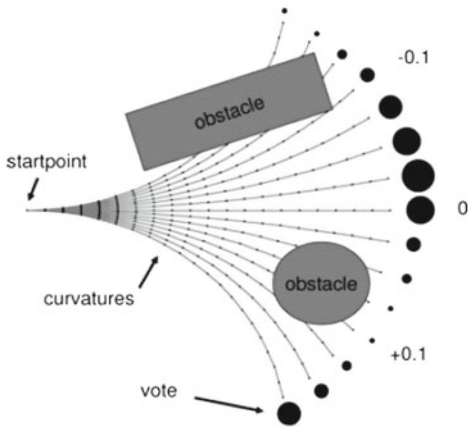

Fig. 3.4 Result of the DAMN arbitration process (Rauskolb et al. 2008, p. 701)

To sum up, reactive architectures have the advantage that they are able to react quickly to unforeseen events and can be easily expanded in terms of the system's functionality. However, this architectural style reaches its limits in complex environments like road traffic since the interaction of the behaviors becomes hard to arbitrate and to optimize. Also, no long-term planning is available such as the planning to drive a complex route. The examples of the Arkin scheme as well as the DAMN architecture show that there has to be a superior module to coordinate the individual behaviors. This circumstance leads to the so-called "layered architectures" explained below.

3.3.1.3 Layered Architectures

Layered architectures emerged in order to combine the advantage of reactivity with deliberate planning. The most common type of layered architectures has three hierarchical layers, as depicted in Fig. 3.5. Just like reactive architectures, in the lowest layer a set of behaviors is implemented, which have the closest connection to sensors and actuators. These behaviors are either active all the time and have to be arbitrated or are triggered specifically by the higher-level executive layer, for example when two behaviors compete for the same resource (e.g. an actuator) (Kortenkamp and Simmons 2008, p. 196). The task of the executive layer is to decompose a high-level plan coming from the planning layer into a sequence of low-level behaviors. Therefore, it has to also set temporal constraints if behaviors are conducted in a sequence or are active concurrently. Usually, the executive layer is implemented as a hierarchical finite state controller which also has the function to monitor the behavior execution and to handle exceptions within a routine (Kortenkamp and Simmons 2008, p. 198).

The intention of the high-level planning layer is to generate long-term plans to reach the system goal. Also, it has to re-plan in case a situation changes, e.g. if a road is blocked. To do this, planners usually can be seen as schedulers, which lay out

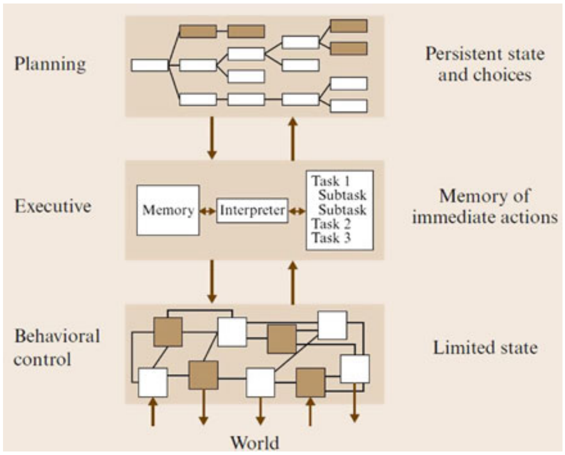

Fig. 3.5 Exemplary structure of layered architectures (Kortenkamp and Simmons 2008, p. 191)

tasks for the executive layer on a time line (Kortenkamp and Simmons 2008, p. 199). They can be implemented in that the executive layer requests a task at a given time or that the planner is always active and hands out tasks to the executive layer.

Maurer (2000, pp. 32–34) summarizes three different types of theoretical hierarchical decompositions. One possibility is to use a "hierarchy of description", in which a complex problem can be described in various levels of detail. In it, a high level layer describes the problem's meaning in order to get a deeper understanding of it and low-level layers give a more detailed explanation. Another type is the "hierarchy of decisional layers" in which the solution to a complex problem can be substituted with a sequence of low order problems that have to be solved. A third way of hierarchical decomposition is the "hierarchy of organization" for multi-goal systems. This kind of architecture is characterized by layers which do not control the lower-order layers completely but coordinate them in such a way that they have a specified freedom of action. The advantage of this type of hierarchy is a more efficient decision process, since the higher-order layers only have to intervene in case a conflict of goals occurs. Hence, such a hierarchy supports the ability to introduce specialist modules or agents into the architecture.

Besides combining reactivity and deliberate planning, layered architectures have the advantage that each layer can be developed, implemented and tested independently of each other. For example, in an early stage of development, behaviors can be triggered manually and tested without the need for an executive layer, or an executive layer state machine can be checked by the trigger outputs in certain situations coming from a planning task without the vehicle or robot needing to actually conduct the behavior.

An important example of a layered architecture is the "Real-Time Control System Architecture" (RCS) which was derived from the US National Institute of Standards and Technology as an advancement of the "NASA Standard Reference Model Architecture" (NASREM) used in various robots, autonomous vehicles and in space travel

(Albus et al. 1994; Albus 1997; Albus 2000). The RCS architecture is a multi-layered architecture in which each layer has a node consisting of a world modeling module that creates maps, events and other state variables concerning its environment and is serviced by a perception module. Also, a behavior generation module exists, which receives a commanded goal from a superior architectural layer, creates a number of plans to achieve it and then gets an expected result of the plan from the world model, including a cost/benefit analysis (Albus 2000, p. 3262). For each plan and a corresponding "executor", a new node of the same kind as just described is used. The layers themselves are arranged hierarchically in terms of spatial range and resolution, whereas each layer differs to the factor 10 from the others. As an example, the top level could be a navigation task that uses a 10 min horizon and a spatial range of about 5 km, the lowest level could be the actuator level that uses a 20 ms time horizon with a spatial range of centimeters. A successful implementation of this reference architecture in the field of ADAS is described by Häring et al. (2009), who proposes the function of an autonomous emergency brake using the RCS paradigm.

A milestone in respect to autonomous vehicle concepts was achieved within the PROMETHEUS project, which introduced the spatio-temporal "4-D" approach (3D space plus time) for dynamic machine vision (Dickmanns et al. 1994). The structure of the hierarchical layers within the 4-D architecture design was inspired by the well-known Rasmussen model of human action (Rasmussen 1983) which comprises "knowledge-based", "rule-based" and "skill-based" behaviors as a base for hierarchical decomposition (Maurer 2000, p. 27). In a subsequent version of the 4-D approach, Maurer (2000) expands the 4-D architecture in a way that various levels of automation could be achieved. Therefore, a key requirement is that the automated system is aware of its own capabilities and is able to activate matching vehicle behaviors during the system's runtime. It works in such a way, that the driver-requested level of automation is permanently compared with correspondent and therefore required quality criteria that have to be fulfilled by the automated system. In case the defined criteria are not fulfilled, for example the dead time of the vehicle control loop exceeds a required threshold, a flexible behavior decision module, which consists of hierarchical ordered state machines, is able to activate appropriate automated driving functions for a lower level of automation. The capability of the 4-D concept was shown by the implementation into the test vehicle "VaMP" which travelled more than 1,600 km in an automated driving mode on public roads (Maurer 2000, p. 114).

The success of both approaches described above (RCS and 4-D) led to a combined architecture which shows the compatibility of both designs, the "4-D/RCS architecture" (Albus 2000). Maurer (Maurer 2000, p. 38) however concludes that a combination of both paradigms results in conceptual disadvantages and hence they should rather be used in parallel, depending on the task to be achieved.

Most recent projects regarding vehicle automation are, for example, the "autonomous car project" of Google or the German team "AutoNOMOS Labs" with the prototype vehicle "MadeInGermany", which has gathered a lot of media attention. Unfortunately, only few details are known so far in terms of the software architecture design of these projects. At least for the "MadeInGermany" project it can be stated that also a layered architecture design was chosen that consists of two main layers, the "behavior layer", itself divided into a high-level "strategy module" which processes data from digital maps and a lower-level "tactics" module which calculates an appropriate vehicle trajectory, and the "controller layer" which controls the given trajectory.[5]

Other examples of layered architectures are widespread and very similar. For further reference see Baker et al. (2008) who used a three-layered architecture for the winning autonomous vehicle "BOSS" in the DAPRA Urban Challenge or Payton (1986), Simmons (1994), Laugier et al. (1998), Miura et al. (1997) and Nelson (1999).

All of the mentioned architectures have been designed for autonomous or highly-automated vehicle guidance. However, since the PRORETA 3 project's intention is to design a vehicle for semi-automated, maneuver-based vehicle guidance, a lacking feature of all of these architectures is that a driver interaction is not provided or not sufficiently described. The state of technology so far regarding automated vehicle guidance in combination with a systematic driver interaction concept consists of the research projects "Conduct by Wire" (CbW), "H-Mode" and "HAVEit".

The software architecture of CbW has a lot in common with the layered architectures described above (Geyer 2013). It is hierarchically structured according to the three-level hierarchy of the driving task of Donges (2012, p. 16), who divides the driving task into the navigation level, the trajectory-based guidance level and the vehicle stabilization level. The architecture consists of a maneuver management module that assigns and enables a set of driving functions, which can be seen as vehicle behaviors, in accordance with the driver's desired maneuver. These maneuvers are obtained via an HMI called maneuver interface. To sum up, in CbW the driver is able to interact with the automation concept via an HMI that allows him/her to assign maneuvers, parameterize trajectories and also manually control the vehicle in an unstructured environment like a parking lot. These interactions then influence the automation in respect to the activation of vehicle behaviors.

In contrast to CbW, the research projects "H-Mode" and "HAVEit" propose a variable level of vehicle automation, ranging from manual driving through to autonomous driving (Löper et al. 2008). The architecture itself consists of a driver interface module, which assesses the driver's state, a command layer to define maneuvers and vehicle trajectories dependent on the automation level and the execution layer which controls the vehicle actuators (Zeng 2010, p. 1667). The driver interaction is arbitrated via a selection unit, in which a quantitative measure, the "valential", decides over the future trajectory. The magnitude of this unit is influenced by a negotiation process between driver and automation based on haptic feedback, e.g. the driver's torque of the steering wheel (Löper et al. 2008, p. 16).

A further publication on driver-vehicle interaction was proposed by Bayouth et al. (1997), who introduced the idea of different automation levels and maneuver based driving already in the year 1997.

[5] Lecture slides and personal question to Professor Raùl Rojas at his speech at FZD, TU Darmstadt on 9 July 2012.

3.3.2 Summary and Discussion

The above excerpt of the state of technology on software architectures of vehicle automation concepts aims to find out which architectural patterns have proved themselves within the field of ADAS and robotics and which system modules are therefore necessary.

It can be stated that the paradigm of behavior-based, layered architectures have proved themselves to be a beneficial architectural pattern for vehicle automation concepts since they have been successfully used in key research projects within the last two decades, right up to the most recent prototypes like "MadeInGermany". Important advantages are the inherent structured arrangement based on a hierarchical and modular task composition, the ability to develop and test the different architectural layers independently of each other and the fact that this type of architecture is capable of deliberate, long-term planning.

An interesting finding is that most layered architectures for vehicle automation consist of three architectural main layers. As Maurer (2000, p. 34) summarizes, three layers have become established in this field of robotics although there is not necessarily a reason for that from the viewpoint of functional architecture. However, we assume that this number comes from common agreed-upon models that help the architect to decompose the driving task in a structured way, like the Rasmussen model of human activities or the Donges model for the driving task, which each contain three main layers (Rasmussen 1983; Donges 2012).

Important modules, in addition to a sensory processing module, that can be found in most automation concepts are a world model that gathers information about the systems' environment, a high-level planning module to be able to achieve long-term goals like vehicle navigation, an executive module that coordinates low-level vehicle behaviors and reacts to exceptions, a set of behaviors and—for cooperative automation concepts—a suitable HMI.

The three projects on cooperative vehicle automation described in Sect. 3.3.1.3 have in common, that no direct mechanical connection exists between the driver and the vehicle because "by-wire" interfaces are being used for the steering as well as the brake and gas pedals. Within the PRORETA 3 project, however, the test vehicle is equipped with regular controllable power steering as well as a brake pedal connected to the main braking cylinder by the electro-pneumatically actuated booster. This leads to the question of how the driver interaction influences the software architecture of automated vehicle guidance concepts when the driver input comes from the hardware architecture.

As a conclusion drawn from the literature research, it can be stated—in accordance with Dickmanns (2005, p. 224) and Gasser et al. (2012, p. 25)—that there is a significant demand for research into the interaction of the human being with automated vehicle concepts. Especially the driver interaction as a problem of command arbitration and its impact on the software architecture and dependency on the hardware architecture is to be investigated.

Based on the information given in this section, an initial design of the PRORETA 3 software architecture is derived in Sect. 3.4.

3.4 Software Architecture for Vehicle Automation

Based on the findings from literature research, the functional requirements of PRORETA 3 and the requirements for the software architecture itself, an architecture design for automated vehicle concepts is derived. In contrast to the robotic architecture patterns for autonomous vehicles, focus is also placed on the human-machine interaction.

A layered architecture is believed to be most appropriate as a design basis for PRORETA 3, in line with the "modularity" and "testability" requirements (cf. Sect. 3.2.4), the inherently well-structured and functionally scalable approach as well as the ability to cope with complex situations such as road traffic. Based on this decision, the question has to be answered of how many layers are, therefore, necessary and how the complex problem of vehicle automation can be decomposed into them.

As shown in Maurer (2000), the model of human activity according to Rasmussen (1983) has proven to be well-suited for an example decomposition, especially for behavior-based layered architectures. This comes from the fact that in behavioral robotics, behaviors can be seen as basic state machines containing mostly simple transfer functions that describe the robot's desired movement and have direct access to sensors and actuators (cf. Sect. 3.3.1.2). Accordingly, Rasmussen (1983) describes the "skill-based behavior" level of human action as a set of automated sensorimotor patterns that process sensory input features into direct actions, which is a very similar procedure. The next higher layer in Rasmussen's model is described as the "rule-based behavior", in which the human recognizes specific "signs" from the perceived sensory features, associates an appropriate task and chooses between a set of stored rules to solve this task. This can be seen as a feedforward control for the sensorimotor patterns (Maurer 2000, p. 29). As an equivalent, most architecture designs use an "executive" or "coordination" layer in which a superior module controls the behaviors in a way that they trigger specific behaviors to solve a recognized task. The similarity of Rasmussen's model to layered architectures also applies for the top-layer. In both paradigms, a planning entity decides between tasks in order to reach its overall goal concerning specific boundary conditions. To sum up, both paradigms consist of hierarchically structured layers that can be understood as layered control loops with an increasing time constant, in which higher layers provide a feedforward control to the next lower layers. Hence, the Rasmussen model serves as a good basis for hierarchical decomposition and is considered in the following design.

After having defined the basic architecture layout and an approach how to decompose it into specific layers, the modules within the layers have to be determined. Tasks that are necessary for vehicle automation are, independent of their exact specification or name, the perception of the vehicle's environment, and an interpretation of the perception that serves as a basis for a behavioral decision which itself feeds a behavior generation (or action) module. A task neglected in most publications on architecture is a description of how to embed the driver into a vehicle automation concept. Therefore, a suitable HMI concept is needed and should be considered in the design.

Based on the above considerations, the actual architecture design process results in a top module view that describes the static architecture layout as shown in Fig. 3.6.

As already described, the architecture is based on three horizontal main layers within the application range (Bauer et al. 2012), which are structured according to their level of abstraction in the Rasmussen model. Vertically, there are also three columns. A world model column has the task of providing all necessary information coming from the vehicle's environment to the planning modules in order to make situation-appropriate decisions. The world model itself is fed by a solution-independent **sensorics- and sensordata-fusion** module which provides information about the existence of perceived objects and the vehicle's surroundings in a detailed but mostly not situational-interpreted way. Approaches of Darms (2007) are suitable for an implementation, however, in PRORETA 3 a geometric grid-map-based description of the static environment as well as an object list for the dynamic environment was chosen. According to the Rasmussen model, most information is used by the lowest "behavior" layer, but also more abstract information via digital maps or Car2X can be provided that a human would not be able to perceive him/herself. From this, the sensorics- and sensordata-fusion module spreads over all layers. On

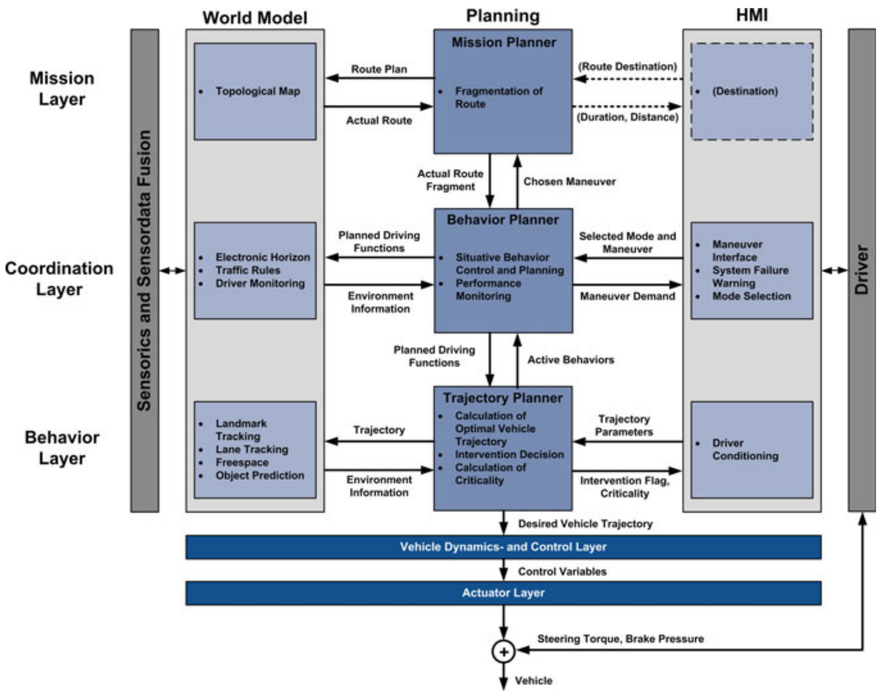

Fig. 3.6 Proposed software architecture for vehicle automation concepts (cf. (Bauer et al. 2012))

this basis, the **world model** is able to interpret the given data in order to analyze and understand a traffic situation. Therefore, specialist modules are needed. In the behavior layer, a landmark tracking module detects localization-relevant features like road junctions or stop lines, a lane tracking module interprets video data for a lane model, a drivable freespace module determines valid maneuvering space (Schreier and Willert 2012) and an object prediction module calculates future positions of relevant entities. For the "coordination layer" more abstract information is needed. Corresponding modules are an electronic horizon based on geometric and digital maps with free-configurable attributes, a module to determine the traffic rules such as the right of way on intersections and the driver monitoring to check whether the driver is able to stay in the overall control loop. For the "mission layer", a graph-based topological map is proposed.

The **planning modules,** which are explained below in this section, receive high-order tasks and split them up into lower-order tasks. A novel element within the proposed behavior-based, layered architecture is the **HMI** column that is needed for driver interaction within a cooperative automation concept. Specialist modules are the driver conditioning which helps to support and warn the driver in critical traffic situations and, for the coordination layer, a maneuver interface to offer the driver possible situation-dependent vehicle maneuvers as well as the possibility to choose a specific mode of automation. A possible module on mission layer level could be the input of a destination, which, however, is not implemented within PRORETA 3.

In the following, the layers and their functionality are described. The layer with the highest hierarchy is called **"mission layer"**. In it, the mission planner module splits the current vehicle route into route fragments and delegates a matching task to the secondary behavior planner module, e.g. "handle intersection" or "follow road". The mission planner module receives its information from a topological map, in which intersections are represented as knots and roads as conjunctive edges. Additionally, corresponding characteristics, e.g. the number of outgoing lanes at an intersection, are deposited. In line with the Rasmussen model, the mission planning module should be able to detect and react to unknown situations in which no rules for the behavior planning module are available. However, in most architecture designs, the task of the top-level layer is limited to the navigation task since—in contrast to human behavior—it is very easy to implement a large number of rules for behavior planning (Maurer 2000, p. 46). For autonomous vehicle concepts, a vehicle destination comes from the HMI module, however, this is not necessary for the cooperative PRORETA 3 concept (dashed lines in Fig. 3.6).

The second layer is called **"coordination layer"** and has the job of controlling the underlying vehicle behaviors according to its current task. The main module within this layer is the behavior planner, which triggers appropriate vehicle behaviors in a timely manner and hence is the central behavior decision unit within the architecture. A common implementation for this kind of planning module is one or more state machines. As an interface towards the behavior layer, disjunctive longitudinal and lateral driving functions, e.g. "lane change" or "follow road" are a possible solution (Maurer 2000, p. 52; Hakuli et al. 2010). As an example of the behavior planner module, the task "handle intersection" with the attribute "traffic light" may result in reducing the vehicles' speed in advance, requesting a desired maneuver from the driver and stopping the vehicle in case of a red traffic light. Therefore, it needs information from the world model, like the right of way or the geometric relation between the car and the scene, e.g. the number of lanes on a road or the distance to the next intersection. Another major task of the behavior planning module is to monitor the system's performance in terms of important resources and modules that are needed for a given task. In case one of those fails or necessary information cannot be provided, it has to transfer the system into a safe state.

The third layer is called **"behavior layer"**, and is where the different vehicle behaviors are implemented. The main trajectory planner module has the function of processing a safe vehicle trajectory in respect to the given driving functions that have been triggered. Therefore, it is possible to implement the behaviors independently, e.g. a sigmoid function for a lane change or a braking intervention in terms of a high negative acceleration, or, to have more complex behaviors like the safety corridor function, that cannot be implemented as just a transfer function. In the trajectory planning module, an emergency trajectory is determined with the use of a potential-field based approach. In order to perform this task, the world model has to provide much more specific information in a spatial and timely manner coming from the specialist modules described above. The HMI on the behavior level is called "driver conditioning" and has the task of embedding the driver not only via information and warning but also in an accident-preventive way.

In the lower hierarchy of the proposed architecture two more layers exist outside of the functional range. In the **vehicle dynamics and control layer**, the given vehicle trajectory is processed into the vehicle dynamics controllers. This layer then generates commands for the actuators like a steering torque or a brake pressure which are then processed by the **actuators** themselves.

The importance of these two layers should not be underestimated since the driver is able to directly influence the actuators depending on the hardware concept of the car. This may not be problematic within an autonomous concept using by-wire interfaces for human-machine interaction or also for single ADAS functionalities like adaptive cruise control. However, it is problematic within the full range of automation functionalities in-between, especially for systems that utilize standard HMIs like a steering wheel and a brake pedal. As an example, in the case of an emergency maneuver, the driver wants to evade to the right, the automation to the left. The question then is, is the driver treated as a disturbance quantity which the automation wants to compensate? If not, how is the force-transition process back to the driver to be defined? So far, no scientific investigation has been conducted on how to integrate the drivers' intention in that direct mechanical access to the actuators has to be integrated within the overall arbitration strategy. This reveals a clear demand for research in the future.

In the architecture design for vehicle automation concepts proposed above, the Rasmussen (1983) model for human action was used as a basis for hierarchical decomposition. A novel element within behavior-based layered architectures is the integration and description of a suitable driver interface into the design, which is especially important for the cooperative automation concepts explained in Sect. 3.1.

Details of the necessary specialist modules for situation interpretation were presented. The proposed design makes some exceptions with regard to the Rasmussen model, e.g. the sensory input and the knowledge-based behavior. However, this is because vehicle behavior cannot be completely subsumed by human behavior (Maurer 2000, pp. 30–31). Decomposition paradigms need to serve as a flexible helper, not as a rigid burden for the system architect.

3.5 Summary and Outlook

This chapter proposes a systematic and top-down development methodology for automated vehicle guidance concepts and their corresponding system- and software architecture.

To this end, a software architecture design process was introduced in Sect. 3.2, which bridges the gap between requirements analysis and implementation within the overall development process according to the well-known V-Model. Explanations were given on the importance of knowing limiting influencing factors, ways of evaluating architecture design and on the conduct of a literature research on architectural patterns.

The state-of-technology identified was that most vehicle automation concepts utilize behavior-based, hierarchically layered architectures since they are well-structured in an inherent way and hence facilitate a modular design. This allows a division of labor and an efficient design-, implementation and validation process. However, most architectural patterns have been designed for autonomous vehicle guidance and do not support a human-machine interaction on a cooperative basis. This is why the architecture proposed in Sect. 3.4 facilitates a new human-machine interface column. From a first evaluation, the architecture meets the functional and architectural requirements mentioned in Sect. 3.2.4 thanks to its modular approach. In future, the proposed architecture has to prove itself within the further development process of the PRORETA 3 project.

A clear demand for research has been identified in terms of the interaction and arbitration metric between the driver and the automation with regard to the system's hardware architecture.

In the automotive systems engineering discipline and the associated scientific community, it would be desirable for a lively discussion to arise about efficient development processes and structures of system architectures, just like that underway in the example discipline of robotics.

Acknowledgments We thank Continental AG for kindly funding this work within the PRORETA 3 cooperation, which aims to develop future concepts for integrated driver assistance systems.

References

Albus, J., Lumia, R., Fiala, J.: NASREM—The NASA/NBS Standard Reference Model for Telerobot Control System Architecture, discussion paper. National Institute of Standards and Technology, Gaithersburg (1994)

Albus, J.: The NIST real time control system (RCS): an approach to intelligent systems research. J. Exp. Theore. Artif. Intell. **9**(2–3), 157–174 (1997)

Albus, J.: 4-D/RCS reference model architecture for unmanned ground vehicles. In: IEEE International Conference on Robotics and Automation. San Francisco, USA (2000)

Arkin, R.: Motor Schema based Navigation for a Mobile Robot. In: IEEE International Conference on Robotics and Automation. Amherst, Massachusetts (1987)

Baker, C., Ferguson, D., Dolan, J.: Robust Mission Execution for Autonomous Urban Driving, Carnegie Mellon Research Showcase,Robotics Institute, Paper 178 (2008)

Barrios, J., Aparicio, A., Dundar, S., Schoinas, D.: Common Database of Existing Safety Functions and Corresponding System Platforms, Report of Deliverable 6.1, Trace Project (2007). http://www.trace-project.org (September 17th, 2012)

Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 2nd edn. Addison-Wesley, Boston, USA (2003)

Bauer, E., Lotz, F., Pfromm, M., Schreier, M., Abendroth, B., Cieler, S., Eckert, E., Hohm, A., Lüke, S., Rieth, P., Willert, V., Adamy, J., Bruder, R., Konigorski, U., Winner, H.: PRORETA 3: An Integrated Approach to Collision Avoidance and Vehicle Automation. In: at-Automatisierungstechnik, 12/2012. Oldenbourg Wissenschaftsverlag (2012)

Bayouth, M., Nourbakhsh, I., Thorpe, C.: A hybrid human-computer autonomous vehicle architecture. In: Third ECPD International Conference on Advanced Robotics, Intelligent Automation and Control, Belgrade, Serbia (1997)

Bohn, A.: Produktinnovation, Lecture Notes. Department of Mechanical Engineering, TU Darmstadt (2012)

Brooks, R.: A robust layered control system for a mobile robot. IEEE J. Robot. Autom. **2**(1), 14–23 (1986)

Broy, M., Krüger, I., Meisinger, M.: Automotive Software—Connected Services in Mobile Networks. Springer, Berlin (2006)

Darms, M.: Eine Basis-Systemarchitektur zur Sensordatenfusion von Umfeldsensoren für Fahrerassistenzsysteme. VDI, Düsseldorf (2007)

Dickmanns, E.D., Behringer, R., Dickmanns, D., Hildebrandt, T., Maurer, M., Thomanek, F., Schiehlen, J.: The seeing passenger car 'VaMoRs-P'. In: IEEE Intelligent Vehicles Symposium, Paris (1994)

Dickmanns, E.D.: Vision: Von Assistenz zum Autonomen Fahren. In: Maurer, M., Stiller, C. (eds.) Fahrerassistenzsysteme mit maschineller Wahrnehmung. Springer, Berlin (2005)

Donges, E.: Fahrerverhaltensmodelle. In: Winner, H., Hakuli, S., Wolf, G. (eds.) Handbuch Fahrerassistenzsysteme. 2nd edn. Vieweg/Teubner, Wiesbaden (2012)

Gasser, T., Arzt, C., Ayoubi, M., Bartels, A., Bürkle, L., Eier, J., Flemisch, F., Häcker, D., Hesse, T., Huber, W., Lotz, C., Maurer, M., Schumacher, S.-R., Schwarz, J., Vogt, W.: Rechtsfolgen zunehmender Fahrzeugautomatisierung. Wirtschaftsverlag NW, Bremerhaven (2012)

Geyer, S.: Maneuver-based vehicle guidance based on the Conduct-by-Wire principle. In: Maurer, M., Winner, H. (eds.) Automotive Systems Engineering. Springer-Verlag Berlin, Heidelberg (2013)

Häring, J., Wilhelm, U., Sailer, U.: Systemarchitektur des Predictive Safety Systems, in ATZ Elektronik, 3/2009. Springer, Heidelberg (2009)

Hakuli, S., Kluin, M., Geyer, S., Winner, H.: Development and Validation of Manoeuvre-Based Driver Assistance Functions for Conduct-by-Wire with IPG CarMaker, FISITA 2010 World Automotive Congress. Budapest, Hungary (2010)

Hakuli, S., Geyer, S., Winner, H., Henning, J.: Integriertes Konzept für die manöverbasierte Fahrerassistenz, in ATZ Automobiltechnische Zeitschrift, 3/2011. Springer, Heidelberg (2011)

Hakuli, S., Bruder, R., Flemisch, F., Löper, C., Rausch, H., Schreiber, M., Winner, H.: Kooperative Automation. In: Winner, H., Hakuli, S., Wolf, G. (eds.) Handbuch Fahrerassistenzsysteme. 2nd edn. Vieweg/Teubner, Wiesbaden (2012)

IEEE Standard 1471–2000: Recommended practice for architectural description of software-intensive systems, Institute of Electrical and Electronics Engineers/Circuit Theory Group (2000)

Kauer, M., Schreiber, M., Bruder, R.: How to conduct a car? In: IEEE Intelligent Vehicles Symposium 2010. San Diego (2010)

Laugier, C., Fraichard, T., Paromtchik, I.E., Garnier, P.: Sensor-Based Control Architecture for a Car-Like Vehicle, IEEE International Conference on Intelligent Robots and Systems, Victoria (1998)

Löper, C., Kelsch, J., Flemisch, F.: Kooperative, manöverbasierte Automation und Arbitrierung als Bausteine für hochautomatisiertes Fahren, in Automatisierungs-. Assistenzsysteme und eingebettete Systeme für Transportmittel, Gesamtzentrum für Verkehr, Braunschweig (2008)

Maurer, M.: Flexible Automatisierung von Straßenfahrzeugen mit Rechnersehen, VDI Fortschrittsberichte Reihe 12 Nr. 443, VDI, Düsseldorf (2000)

Maurer, M.: Entwurf und Test von Fahrerassistenzsystemen. In: Winner, H., Hakuli, S., Wolf, G. (eds.) Handbuch Fahrerassistenzsysteme. 2nd edn. Vieweg/Teubner, Wiesbaden (2012)

Miura, J., Ito, M., Shirai, Y.: A three-level control architecture for autonomous vehicle driving in a dynamic and uncertain traffic environment. In: IEEE Conference on Intelligent Transportation System, Boston (1997)

Nelson, M.: A design pattern for autonomous vehicle software control architecture. In: IEEE International Computer Software and Applications Conference, Phoenix (1999)

Payton, D.: An architecture for reflexive autonomous vehicle control. In: IEEE International Conference on Robotics and Automation, San Francisco (1986)

Posch, T., Birken, K., Gerdom, M.: Basiswissen Softwarearchitektur, 2. edn. dpunkt, Heidelberg (2007)

Rasmussen, J.: Skills, rules and knowledge; signals, signs and symbols, and other distinctions in human performance models. IEEE Trans. Syst. Man Cybern. **13**(03), 257–266 (1983)

Rauskolb, F., Berger, K., Lipski, C., Magnor, M., Cornelsen, K., Effertz, J., Form, T., Graefe F., Ohl, S., Schumacher, W., Wille, J.-M., Hecker, P., Nothdurft, T., Doering, M., Homeier, K., Morgenroth, J., Wolf, L., Basarke, C., Berger, C., Gülke, T., Klose, F., Rumpe, B.: Caroline: an autonomously driving vehicle for urban environments. J. Field Robot. **25**(9), 674–724 (2008). Wiley InterScience

Reichart, G., Bielefeld, J.: Einflüsse von Fahrerassistenzsystemen auf die Systemarchitektur im Kraftfahrzeug. In: Winner, H., Hakuli, S., Wolf, G. (eds.) Handbuch Fahrerassistenzsysteme. 2nd edn. Vieweg/Teubner, Wiesbaden (2012)

Rosenblatt, J.: DAMN: a distributed architecture for mobile navigation. J. Exp. Theor. Artif. Intell. **9**(2–3), 339–360 (1997)

Kortenkamp, D., Simmons, R.: Robotic system architectures and programming. In: Siciliano, B., Khatib, O. (eds.) Springer Handbook of Robotics, Springer, Berlin (2008)

Schreier, M., Willert, V.: Robust free space detection in occupancy grid maps by methods of image analysis and dynamic B-Spline contour tracking. In: IEEE Conference on Intelligent Transportation Systems, Anchorage (2012)

Simmons, R.: Structured control for autonomous robots. In: IEEE Transactions on Robotics and Automation, pp. 10–1 (1994)

Starke, G.: Effektive Software-Architekturen, 4th edn. Hanser, München (2009)

Tölle, W.: Ein Fahrmanöverkonzept für einen maschinellen Kopiloten. VDI, Düsseldorf (1996)

Vogel, O., Mehling, U., Neumann, T., Thomas, A., Chughtai, A., Völter, M., Zdun, U.: Software-Architektur. Spektrum Akademischer, Heidelberg (2005)

Winner, H., Weitzel, A.: Quo Vadis, FAS?. In: Winner, H., Hakuli, S., Wolf, G. (eds.) Handbuch Fahrerassistenzsysteme. 2nd edn. Vieweg/Teubner, Wiesbaden (2012)

Zeng, H.: HAVEit—A Driver Centric Vehicle Automation System with a Scalable and Flexible Architecture, 19. Aachener Kolloquium Fahrzeug- und Motorentechnik, Aachen (2010)