Article:

J. Schlatow et al., "Self-awareness in autonomous automotive systems," Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, Lausanne, 2017, pp. 1050-1055.

DOI 10.23919/DATE.2017.7927145

Self-awareness in autonomous automotive systems

Johannes Schlatow, Mischa Möstl and Rolf Ernst
Institute of Computer and Network Engineering
Technische Universität Braunschweig
{schlatow,moestl,ernst}@ida.ing.tu-bs.de

Marcus Nolte, Inga Jatzkowski and Markus Maurer
Institute of Control Engineering
Technische Universität Braunschweig
{nolte,jatzkowski,maurer}@ifr.ing.tu-bs.de

Christian Herber and Andreas Herkersdorf
Institute for Integrated Systems
Technische Universität München
{christian.herber,herkersdorf}@tum.de

***Abstract*—Self-awareness has been used in many research fields in order to add autonomy to computing systems. In automotive systems, we face several system layers that must be enriched with self-awareness to build truly autonomous vehicles. This includes functional aspects like autonomous driving itself, its integration on the hardware/software platform, and among others dependability, real-time, and security aspects. However, self-awareness mechanisms of all layers must be considered in combination in order to build a coherent vehicle self-awareness that does not cause conflicting decisions or even catastrophic effects. In this paper, we summarize current approaches for establishing self-awareness on those layers and elaborate why self-awareness needs to be addressed as a cross-layer problem, which we illustrate by practical examples.**

I. Introduction

Autonomous driving is currently a very hot topic in the automotive domain. In particular, recent advances in the application of machine learning for environmental perception pave the way for fully automated vehicles. However, building truly autonomous automotive systems requires sophisticated self-assessment capabilities in order to be fail-operational (i.e. continue a safe operation) in all scenarios. Autonomous vehicles must therefore possess knowledge of (formal) boundaries of their actions to avoid catastrophic effects.

Self-awareness has already been proposed for Autonomic Computing as a means to cope with complexity [1]. Self-awareness refers to a system's capability to recognize its own state, possible actions and the result of these actions on the system itself and on its environment. This principle has been researched on different system layers as, e.g. recently presented in the context of the Internet of Things [2]. However, in order to meet the complex requirements of autonomous vehicles, self-awareness must not be taken separately on each layer (i.e. hardware/software platform, communication, sensors, driving function, etc.) but combined into a coherent vehicle self-awareness which prevents destructive behavior due to conflicting decisions. In this paper, we elaborate on our observation that self-awareness for automotive systems must be approached as a cross-layer problem.

In Section II, we present a research project into self-aware automotive platforms. Section III illustrates how virtualization technologies enable the model-based and platform-independent management of autonomous systems. Section IV summarizes the state of autonomous driving w.r.t. (functional) self-awareness capabilities of automotive vehicles. In Section V, we motivate that self-awareness must be addressed as a cross-layer mechanism before we conclude this paper in Section VI.

II. Self-awareness in Automotive Platforms

Today's automotive vehicles are typically developed following the V-model process, which particularly suits the deployment of entire, well-tested products as practiced in the automotive domain. While being well-established and efficient for safety-critical embedded systems, this process does not natively address maintenance of these systems after deployment or product variants and product lines. More precisely, every change/update must be adequately tested before it can be applied in the garage, which typically restricts modifications to a single-owner (i.e. the OEM).

The German DFG Research Unit *Controlling Concurrent Change* (CCC) (8 faculty, 6 yrs funding) addresses the topic of managing in-field changes to automotive and space vehicles by establishing an automated model-based integration process for safety-critical embedded systems that can be deployed as a self-protecting mechanism along with these systems. As a result, CCC combines a conventional lab-based design of individual functions with an automated integration process which ensures that updates are applied to an already deployed system only if the system can still adhere to the required safety and security constraints.

This becomes particularly challenging as the target platform is shared by multiple functions with different criticality. All side-effects must therefore be anticipated and either be bounded or mitigated in order to ensure safe operation of critical functions at all times. For this purpose, all requirements and constraints must be explicitly specified in the input models. Another challenge consists in finding (and specifying) appropriate abstractions that guide the decisions which must be made during such a model-based integration process as these are usually based on experience and expert knowledge, which is only implicitly available.

In CCC, a system's architecture is divided into a model domain and an execution domain as illustrated in Fig. 1. As in any conventional system architecture, the execution domain provides the run-time environment (RTE) including the operating system (OS) required for hosting multiple application components. The execution domain is augmented by application and platform monitoring capabilities to close the gap between model assumptions and the actual behavior of the execution domain. The model domain employs formal

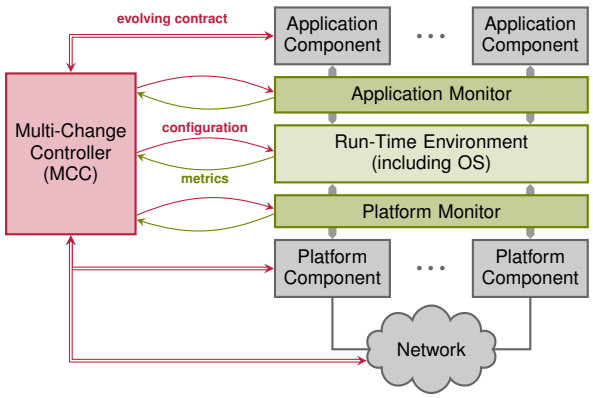

Fig. 1. CCC architecture comprising a model domain (red), an execution domain (green) as well as changing application/platform components (gray).

methods to control changes of all platform components such as CPUs or networks. In conjunction, the methods and mechanisms developed in both domains equip a CCC platform with safe change capabilities. In the next sections, we summarize the essential methods and mechanisms of both domains.

A. Model Domain

In the CCC architecture, a so-called *Multi-Change Controller (MCC)* takes full control over the system and platform configuration by implementing the methods developed in the model domain. It performs the integration process and ensures that a new configuration passes all necessary acceptance and conformance tests. Similar to the conventional V-model development process, the MCC gradually refines the model representation of the new system configuration during the integration process. It further introduces additional layers that model certain aspects of the system in order to represent particular viewpoints such as safety, availability or security. The requirements for these viewpoints – e.g. a safety-level requirement or a real-time constraint – are collected for each component in a so-called contracting language, which serves as an input to the MCC. Viewpoint-specific analyses can be implemented as separate entities in the MCC, e.g. to resolve run-time dependencies between software components [3] or to build a security threat model for vehicular systems [4].

A change to a system can be the addition of a new functionality that is modeled in a logical or functional system architecture in a platform-independent way. The integration process first involves fitting this functionality to the target platform that typically consist of multiple processing resources and networks. Secondly, the resulting technical architecture is transformed and mapped to a model of its implementation (i.e. software components and their interconnection) such that undesired dependencies/interference is avoided. This process is assisted by formal analyses that a) can guide the (mapping) decisions and b) work as acceptance tests. For instance, a worst-case response time analysis can check real-time constraints based on a timing model of the system.

B. Execution Domain

In order to apply the model-based methods implemented in the MCC, the execution domain must be able to enforce the modeled behavior where necessary. In the scope of CCC, we base the execution domain on microkernel-based operating systems, which already provide a strong isolation of application components and fine-grained access control that allows to follow the principle of least privilege while being dynamically configured at run time. The software-component model used in the model domain is based on the component-based semantics of these systems in which so-called *micro servers* provide services that can be granted to other components that require these services. Note that, however, the methods developed in the model domain are not restricted to these semantics but can be adapted to different implementation models.

As mentioned above, the execution domain is augmented by run-time monitoring capabilities that a) enforce certain model assumptions or b) extract run-time metrics that can be fed back into the model domain for optimization. As the MCC relies on the adherence of implementations to their modeled behavior, it can configure the monitoring facilities to enforce, e.g., the access policy to network resources [5] or real-time behavior [6] where necessary. On the other hand, supervising certain run-time properties, such as execution times, access patterns, or sensor values, equips the system with self-awareness capabilities w.r.t. certain performance metrics and is actually implemented with very little interference on the actual functionality. This enables the model domain to detect deviations from the nominal behavior, refine its models, anticipate changes, and adapt the system configuration accordingly.

III. Virtualization as Enabler

Virtualization techniques are effective means to achieve temporal and spatial segregation among mixed-criticality applications running on shared multicore computing platforms using on- and off-chip interconnect and memory/storage resources. This ability to isolate, i.e. guarantee freedom of interference between, applications and provision them with their own logical resources is a powerful enabling technology to support environments witnessing continuous change. Modifications made on one virtual machine (VM) will not affect other VMs. Such hypervisor- or VMM-based process virtualization [7], interconnect and memory virtualization methods are layered underneath the MCC services introduced above. The concisely defined allocation and mapping between application tasks and virtualized resources is also helpful in context of system self-awareness. Autonomous automotive systems have extremely high, variable and domain-specific demands for computing, communication and memory requirements. In the sequel, we describe an architecture for automotive CAN controller virtualization for multicore processors. Performance bounds and cost impacts of the virtualization wrapper are assessed experimentally and via synthesis.

In the context of autonomous automotive systems, industry-specific protocols and properties like real-time capability and

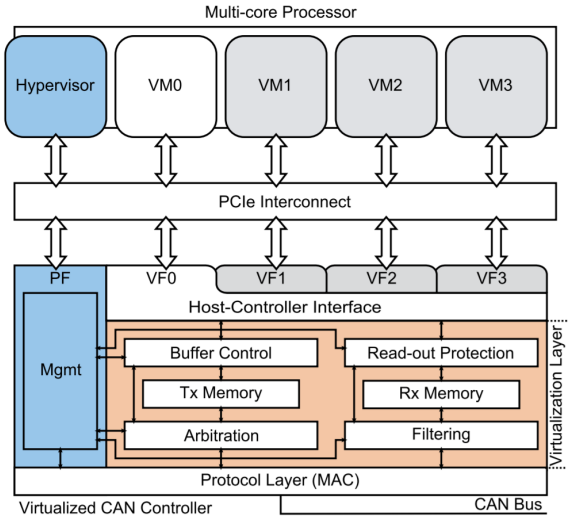

Fig. 2. System Architecture using a virtualized CAN controller [8].

safety have to be considered. Our solution follows a layered architecture (Fig. 2), which extends a traditional CAN controller (referred to as protocol layer) by a virtualization layer. The additional hardware virtualization layer ensures that CAN messages from multiple VMs are properly isolated and transmitted with respect to their bus priority in real-time. On the receiving end, messages are filtered towards the VMs.

An important aspect of the architecture is the division of the virtualized CAN controller into a physical function (PF) and virtual functions (VFs). The VFs provide data path functionality only, while the PF is able to perform privileged operations, e.g. modifying the bus speed or managing the VF resources. The PF shall only be accessible to privileged SW components, e.g. the hypervisor running an MCC. The VMs host multiple concurrent execution domains.

We quantified the performance of the solution in an experimental system setting [8], using an Intel i7-3770T CPU and an FPGA prototype based on Xilinx Virtex-7. The results show that near-native transmit and receive performance can be achieved, with an added latency around 7-11 μs for a round-trip. In terms of FPGA resources, the virtualized solution breaks even with multiple stand-alone controllers at for VMs.

The applicability of this solution to automotive scenarios allows highly flexible but robust systems at the same time. Platform virtualization in combination with proper (potentially HW-assisted) virtualization of critical peripheral components can thus be used as enablement technology in scenarios using an MCC and multiple execution domains on a single chip.

IV. Self-Awareness in Autonomous Driving

Recent advances in the field of automated road vehicles, have shifted the focus of the research community from (advanced) driver assistance systems (SAE levels 1&2 [9]) towards fully automated vehicles (SAE level 5 [9]). With increasing automation, the driver's task shifts from performing the actual driving task (levels 1&2) to system monitoring (levels 3&4). Eventually, when considering fully automated vehicles (level 5), the driver has to be assumed completely out of the control loop. In this case, the vehicle must be able to act autonomously also in critical situations. Thus the vehicle must remain fail-operational at least until a safe stop is reached without constituting a safety risk for other traffic participants.

In order to being able to maintain a fail-operational state, the vehicle must be *self-aware* at the functional level. This means, that each function must be able to assess its current performance and be able to autonomously isolate faults in order to being able to engage appropriate counter measures. In addition it must also be aware of possible redundant functional modules which can replace faulty modules, if degrading performance is detected. From a functional safety perspective, with self-monitoring of the system and the knowledge of the vehicle's current capabilities it is possible to maintain a safe vehicle state at all times [10]. This knowledge of the vehicle's current capabilities can additionally be employed to influence driving decisions of the automated vehicle and adapt these to the current performance capabilities.

Considering the state of the art in series vehicles, current monitoring strategies only take small sets of system states into account, such as tire pressure or battery charge. Simple faults of single sensors or actuators, particularly in the drive train, can then be detected using on-board diagnosis (OBD). For fully automated vehicles, these self-diagnostic capabilities need to be extended towards the data quality assessment for environmental sensors (e.g. cameras, LiDAR-, RADAR-sensors) and, as mentioned above, the quality assessment of functions (e.g. object tracking, control algorithms).

First ideas towards self-awareness at a functional level for autonomous systems have already been stated in [11] and [12], without specifying detailed concepts. More concrete concepts for the explicit internal representation of an autonomous vehicle's skills and abilities have been developed by [13] and detailed at the example of monitoring the performance of control algorithms. These concepts have been extended by [14] and [15] to serve as an input for decision making.

Within the RACE project [16], a flexible software and partially redundant hardware architecture is proposed as a solution to the challenge of creating highly available and dependable automated vehicle systems. However, detection of sensor failures is limited to performing a set of boundary checks for the respective sensors. With SAFER, a flexible framework was proposed in [17], which uses hot and cold stand-by nodes in the system as an effort to increase the dependability of the system. Any degradation strategy is only activated if the heartbeat of a sensor goes missing [17], a more detailed monitoring of sensor data quality is not considered by the authors. Both approaches do not use the gathered information in a wider context to create a detailed representation of the current system performance as it would be required for a self-aware vehicle.

In the context of the *Stadtpilot* project, a surveillance and safety subsystem for monitoring the current system operation state of an automated vehicle was proposed [18]. It states the necessity of being able to detect failures of software and

hardware modules, and have the system react to these failures immediately. Performance criteria for the system are identified and functional degradation strategies are proposed and applied to the research vehicle Leonie.

Bergmiller [19] suggests a skill network for monitoring and fault-detection in full-by-wire vehicles. In the scope of the above mentioned research unit CCC, we use the experimental x-by-wire vehicle *MOBILE* which is contributed by the Institute of Control Engineering. For this vehicle, we evaluate use cases including control applications and automated driving functions on top of the CCC architecture presented in Section II. The integration of the CCC architecture in MOBILE for these use cases has been specified in [20]. In addition, we develop monitoring mechanisms for self-protection against harmful (function) updates and for self-awareness of control applications [21]. The latter is of particular importance as it allows to react to decreased control performance due to operating conditions that have not been anticipated. By combining a model of the vehicle with a performance model this self-representation allows to monitor the physical state of the vehicle and to assess the current performance of each of the vehicle's skills. Following this concept and the ideas of [14] and [15], Reschka [22] proposes the concept of ability and skill graphs as an holistic approach for modeling the vehicle's capabilities considering self-perception and self-representation not only for runtime monitoring, but also as a modeling tool in a development process. A skill can be understood as an abstract representation of the driving task including the conditions necessary to provide it while an ability is derived from an abstract skill by instantiation and including information about the ability's current performance. Skill graphs can be used in the development phase of a vehicle guidance system to model the required abilities for the application domain of the system. With a variable degree of detail, skill graphs may guide the development process by revealing necessary redundancies in the system to achieve identified safety goals. It can also be employed to visualize error propagation and performance degradation in the system. Within the implemented system ability graphs are used during operation of the vehicle to monitor the current system performance. The ability level of the vehicle can then guide decision making and the vehicle's behavior execution.

A skill graph is a directed acyclic graph (DAG) that consists of skill nodes, data sink nodes, data source nodes, and dependency relations between the nodes. A path in this DAG, starting with a main skill and ending at a data source or data sink, represents a chain of dependencies between abilities. To illustrate the construction of a skill graph we take the example of Adaptive Cruise Control (ACC) as the driving task that shall be modeled. In this case, *ACC driving* will make up the main skill or root of the DAG. The main skill is gradually refined to an arbitrary degree of detail until the data sources and data sinks are reached. More precisely, for realizing *ACC driving*, the abilities to *control distance*, to *control speed* and to *keep the vehicle controllable for the driver* are required. To keep the vehicle controllable for the driver it is necessary to *estimate the driver's intent* and to be able to *decelerate* the vehicle if required. To control the distance to the preceding vehicle and to control the speed of the ego vehicle the skill to *select a target object* is needed. Both the aforementioned abilities are also dependent on the skill to *estimate the driver's intent* and the skill to *accelerate* and *decelerate*. For the selection of a target object, the system has to be able to *perceive and track dynamic objects* which itself depends on *environment sensors* as data sources. To estimate the driver's intent, a form of *HMI* is required as a data source. Acceleration and deceleration both require the *powertrain system* as a data sink while deceleration also requires the *braking system* as a data sink. A graphical representation of this skill graph can be found in [22]. To use this graph for monitoring it has to be transformed into an ability graph by choosing a suitable implementation.

Given appropriate metrics for the ability graph, this approach allows for error detection within the different levels of detail in the graph. In case of a reduced ability level it is possible for the system to apply graceful degradation tactics, e.g. by switching to different software modules or by performing self-reconfiguration. The development of appropriate metrics, aggregated measures and models for performance propagation is subject to ongoing research.

V. Cross-Layer Aspects of Self-Awareness

A general challenge for self-aware autonomous systems is the fact that they are operated in an environment that allows only limited predictability. In general, not all the effects that impact the system can be fully anticipated. This targets functional as well as non-functional properties of a system in the same manner. While certain effects can be modeled probabilistically to allow a limited predictability of the influenced properties, other properties are influenced by effects that lack such descriptions. For instance, while platform reliability (property) is typically approached with probabilistic modeling of hardware failures (effect), data integrity is challenging to guarantee since the absence of security threats (as modeled in [4]) cannot be trivially proved due to a lacking model for the influencing effects. Instead, the system must be able to assess such uncertainty in a *self-aware* manner, meaning that its goal in case of an attack is to find suitable countermeasures.

However, finding a suitable countermeasure requires the capability to identify the leak and isolate it on the appropriate layer. For instance, if only a single IP-based service is affected by a security leak, it will be more appropriate to contain this service than to terminate all network connections on the Ethernet layer. The security leak in this case is the environment that excretes an influence on the system, forcing it to react. A self-aware system is then able to identify the most appropriate layer to respond to detected anomalies without the need to anticipate the exact situation at design time.

As mentioned above, self-awareness of an automated vehicle is crucial for guaranteeing its safe operation. When considering level 5 (*fully automated* according to [9]) systems, in which the driver is not in the loop to take over control, the system needs to provide mechanisms for resilient operation

because the vehicle must remain fail-operational at least until a safe stop in a safe place is reached. Classical fault tolerance is not sufficient, here, as errors may have many causes including conceptual and implementation faults, and interference effects in function and platform. The task of detecting and handling such effects cannot be addressed on a single layer, i.e. representation of the system, alone. Knowledge of critical cross-layer dependencies and their effects is an essential part of self-awareness. In traditional design, such dependencies are identified with semiformal methods, such as a *Failure Mode and Effects Analysis* (FMEA). In CCC, such dependency analysis is automated to derive cross-layer dependency models describing the effect of change and actions on the overall system [23], [24]. In future vehicles, cross-layer dependencies are likely to grow as hardware component and power management become more sophisticated and stateful. Self-awareness and learning algorithms will also be helpful at this level, but will contribute to a richer system state [25]. Because environmental effects (e.g. security leaks, component failures) and their impact on the system can never be fully anticipated at design time, an automated vehicle must be able to engage countermeasures against complete system failure at run time.

An example of possible environment interference is the temperature regime within and around the considered system. Ambient temperatures are a source of common cause faults that can affect the operation of a system in many ways. On the one hand, temperature can alter the physical properties of the system such that the anticipated plant models for control software no longer apply. On the other hand, it can cause performance degradation of the (hardware) platform, which, in a self-aware system, may influence the error model and/or require voltage or frequency scaling to prevent permanent damage. This alone, however, does not fully contain the fault as the deteriorated hardware performance can still cause deadline misses and other, functional, faults.

Many more challenges that manifest with different effects on distinct layers exist, ranging from other platform (hardware) aspects over safety and security side-channels to security flaws of the employed hardware. All of them have in common that a deviation of the anticipated/expected behavior must be detectable by a system as a prerequisite to become self-aware. This can be facilitated by monitoring capabilities that allow the system to capture its current status and compare it to an expected state, or sequence of expected states. Generally speaking, monitoring is also performed based on models and metrics extracted from individual layers. Yet in order to achieve a meaningful self-awareness, the overall monitoring concept must ensure that metrics from different layers can be aggregated to a consistent self-representation of the system.

Furthermore, a system must have capabilities to counteract the experienced effects on each layer in order to properly handle them. However, this does not imply that every consequence of an observed event must be handled locally on each layer. Instead, resolving the situation is a primal example where cross-layer solutions are necessary in order to avoid that complexity gets out of hands on individual layers.

Thus, in order to provide the required level of resilience, we propose to combine a self-aware middleware (cf. Section II) also with self-awareness at a functional level (cf. Section IV). In this way, the system can be equipped with multiple problem solving strategies, which shall be illustrated by the example of an intrusion detection system.

By monitoring communication behavior, the system itself is capable of detecting components or subsystems affected by a security leak [5]. In this example, we assume a security flaw in the software component governing rear braking. The only viable option for the system is often to shut down the affected component, however, this can happen in two fundamentally different ways. On the one hand, the effect of the suddenly switched off component can be treated as a component failure on the safety layer, where this effect must have been anticipated as part of the safety design. For instance, a safe-guard such as a redundancy concept is in place and allows to continue the service even in the presence of the faulty or switched off component. Also, recovery mechanisms such as restarting the service with a different software setup may count as a countermeasure on the safety layer.

On the other hand, the fact that the rear braking capability is compromised and must be shut off can be propagated to the ability layer within the functional level to reassess available skills. The layer can then search for solutions how the system can keep up desired functionalities that require rear-braking. One solution to this could be that the objective of driving can be kept operational although the ability to brake is only partially available by reducing the maximum speed and generating additional brake torque from the drive train in order to stay in safe margins. On the contrary, the ability layer can forward the search for solutions to the objective layer and alter the driving objective of the system. An option would be to transition the system into a safe state, i.e. stop driving and, only afterwards, deactivate the affected component and subsystems. In conclusion, the cross-layer approach to the problem allows quite different solution spaces for detected problems while the system is in operation, forcing the system to be aware of the consequences of the chosen solution. Hence, as the system can propagate detected problems through the layers, it must ensure that these also cooperate and avoid situations in which the problem is forwarded ad infinitum. Therefore, the correct degree of cooperation between the layers must be found such that the system is able to protect itself from the impact of the environment effects.

Obviously, the environment does not only constitute of physical effects on the system but also of other participants in networks to which the system is connected, and other traffic participants. This requires awareness of the system that any reaction it takes might require cooperation with others and even delegation, raising issues of trust and self-protection against other malicious neighbors. Future self-aware vehicles may cooperate to share information or even to agree on collective behavior as needed in platooning scenarios. Building a platoon with other vehicles can be beneficial in scenarios where the vehicles are differently suited for driving in certain

weather conditions. For instance, driving in dense fog with inappropriate or broken sensors will not be possible by a single autonomous vehicle. Nevertheless, building a platoon with better equipped vehicles could still be a viable option, which, however, raises the issue of trustworthiness and uncertainty. In this case, agreeing on a common velocity or a minimum distance between vehicles in a platoon is an essential but non-trivial problem as the communication to or the platform of another vehicle might not be fully trustworthy or even compromised. While, in this scenario, this can be addressed by agreement or consensus protocols, there is also information that inherently contains uncertainty such as weather forecasts. Typically, the latter does not dramatically influence safe operation of a vehicle. However, if the system was aware, that its systems may degrade on a certain route due to possible weather influences, it could plan alternative routes which avoid weather-related degradation. In this case, a self-aware vehicle could determine whether it plans a (possibly shorter) route across an alpine pass in winter or whether it is advantageous to take a longer detour without risking degraded performance. For the realization of such strategies advanced reasoning under uncertainty will be required at the functional level.

In summary, an autonomous vehicle must be self-aware across all layers in order to assess the above mentioned scenarios adequately.

VI. Conclusion

Autonomous automotive systems are not only expected to meet certain dependability and security requirements but are also exposed to a very heterogeneous and dynamic environment that cannot be fully anticipated at design time. Due to this fact, such systems must also be able to autonomously detect and react to unexpected situations on several layers, such as degrading hardware components, functional deficiency, or a changing physical environment. In this paper, we presented self-awareness as a key property on multiple layers in automotive systems. In order to detect and adapt to unanticipated effects such as thermal stress or security leaks, self-awareness must be introduced on every layer. In particular, we summarized our approaches to this on the hardware/software platform and the autonomous driving function. More importantly, however, we illustrated by several examples that self-awareness is actually a cross-layer property particularly when considered in the complex and safety-critical automotive context. Otherwise, conflicting decisions between multiple layers of self-awareness could lead to catastrophic effects. Building self-aware automotive systems therefore requires new methods – such as a cross-layer dependency analysis or cross-layer modeling for in-field integration – to be developed and applied to address this property.

Acknowledgement

This work was funded within the DFG Research Unit Controlling Concurrent Change (FOR 1800), and partially within the project ARAMiS by the German Federal Ministry for Education and Research with the funding IDs 01|S11035.

References

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, Jan. 2003.

[2] M. Möstl, J. Schlatow, R. Ernst, H. Hoffmann, A. Merchant, and A. Shraer, "Self-aware systems for the internet-of-things," in *Intern. Conf. on Hardware/Software Codesign and System Synthesis*, 2016.

[3] J. Schlatow, M. Moestl, and R. Ernst, "An Extensible Autonomous Reconfiguration Framework for Complex Component-Based Embedded Systems," in *Intern. Conf. on Autonomic Computing (ICAC)*, 2015.

[4] M. Hamad, M. Nolte, and V. Prevelakis, "Towards Comprehensive Threat Modeling for Vehicles," in *Workshop on Security and Dependability of Critical Embedded Real-Time Systems (CERTS)*, 2016.

[5] M. Hamad, J. Schlatow, V. Prevelakis, and R. Ernst, "A communication framework for distributed access control in microkernel-based systems," in *Operating Sys. Platforms f. Embedded Real-Time App.*, 2016.

[6] M. Neukirchner, K. Lampka, S. Quinton, and R. Ernst, "Multi-Mode Monitoring for Mixed-Criticality Real-time Systems," in *Intern. Conf. on Hardware/Software Codesign and System Synthesis*, 2013.

[7] I. Pratt, K. Fraser, S. Hand, C. Limpach, A. Warfield, D. Magenheimer, J. Nakajima, and A. Mallick, "Xen 3.0 and the art of virtualization," in *Linux symposium*, vol. 2. Ottawa, Ontario, Canada, 2005.

[8] C. Herber, D. Reinhardt, A. Richter, and A. Herkersdorf, "HW/SW trade-offs in I/O virtualization for Controller Area Network," in *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015.

[9] SAE International, "Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems," Tech. Rep., 2014.

[10] A. Reschka and M. Maurer, "Conditions for a safe state of automated road vehicles," *it - Information Technology*, vol. 57, no. 4, Jan. 2015.

[11] E. D. Dickmanns, "Multisensorielle Fahrzeugführung," 1987.

[12] K. M. Passino and P. J. Antsaklis, "Modeling and analysis of artificially intelligent planning systems," P. J. Antsaklis and K. M. Passino, Eds. Norwell, MA, USA: Kluwer Academic Publishers, 1993.

[13] M. Maurer, "Flexible Automatisierung von Straßenfahrzeugen mit Rechnersehen," Ph.D. dissertation, Univ. der Bundeswehr München, 2000.

[14] K.-H. Siedersberger, "Komponenten zur automatischen Fahrzeugführung in sehenden (semi-)autonomen Fahrzeugen," Ph.D. dissertation, Univ. der Bundeswehr München, 2003.

[15] M. Pellkofer, "Verhaltensentscheidung für autonome Fahrzeuge mit Blickrichtungssteuerung," Ph.D. dissertation, Univ. der Bundeswehr München, 2003.

[16] A. Scholz, S. Sommer, A. Kemper, A. Knoll, C. Buckl, G. Kainz, J. Heuer, and A. Schmitt, "Towards an adaptive execution of applications in heterogeneous embedded networks," in *ICSE Workshop on Software Engineering for Sensor Network Applications*, 2010.

[17] J. Kim, G. Bhatia, R. Rajkumar, and M. Jochim, "SAFER: System-level architecture for failure evasion in real-time applications," in *Real-Time Systems Symposium (RTSS)*, 2012.

[18] A. Reschka, J. R. Böhmer, T. Nothdurft, P. Hecker, B. Lichte, and M. Maurer, "A surveillance and safety system based on performance criteria and functional degradation for an autonomous vehicle," in *Intern. Conf. on Intelligent Transportation Systems*, 2012.

[19] P. Bergmiller, *Towards Functional Safety in Drive-by-Wire Vehicles*. Springer, 2015.

[20] A. Reschka, M. Nolte, T. Stolte, J. Schlatow, R. Ernst, and M. Maurer, "Specifying a middleware for distributed embedded vehicle control systems," in *Intern. Conf. on Vehicular Electronics and Safety (ICVES)*, Hyderabad, India, Dec. 2014.

[21] M. Möstl, J. Schlatow, M. Nolte, M. Maurer, and R. Ernst, "Automating Future (Function-)Updates," in *Tagungsband ELIV-Marketplace: E/E im Pkw*, Düsseldorf, Germany, 2016, ISBN: 978-3-945435-05-2.

[22] A. Reschka, G. Bagschik, S. Ulbrich, M. Nolte, and M. Maurer, "Ability and skill graphs for system modeling, online monitoring, and decision support for vehicle guidance systems," in *Intelligent Vehicles Symposium (IV)*, 2015.

[23] M. Moestl and R. Ernst, "Handling complex dependencies in system design," in *Design, Automation Test in Europe (DATE)*, 2016.

[24] M. Moestl and R. Ernst, "Cross-layer dependency analysis for safety-critical systems design," in *Architecture of Computing Systems*, 2015.

[25] N. Dutt, F. J. Kurdahi, R. Ernst, and A. Herkersdorf, "Conquering MP-SoC complexity with principles of a self-aware information processing factory," in *Intern. Conf. on Hardware/Software Codesign and System Synthesis*, 2016.