Architecture challenges for intelligent autonomous machines

An industrial perspective

Sagar Behere[1], Fredrik Asplund[1], Andreas Söderberg[2], and Martin Törngren[1]

[1] KTH The Royal Institute of Technology, Stockholm, Sweden
[2] The SP Technical Research Institute, Sweden

Abstract. Machines are displaying a trend of increasing autonomy. This has a far reaching impact on the architectures of the embedded systems within the machine. The impact needs to be clearly understood and the main obstacles to autonomy need to be identified. The obstacles, especially from an industrial perspective, are not just technological but also relate to system aspects like certification, development processes and product safety. In this paper, we identify and discuss some of the main obstacles to autonomy from the viewpoint of technical specialists working on advanced industrial product development. The identified obstacles cover topics like world modeling, user interaction, complexity and system safety.

Keywords: Autonomy, Architecture, Embedded Systems

1 Introduction

Many machines which incorporate embedded systems display a trend of increasing autonomy. Autonomy, in practice, can be considered as the machine's ability to operate without constant human supervision/intervention. In this paper, the main emphasis is to identify and discuss key challenges for architectures of autonomous machines.

Although autonomy has been traditionally associated with robotics and "dirty, dull and dangerous" work loads, it is now moving to all kinds of domains, driven by environmental, efficiency and safety considerations[1]. To exemplify the trend of increasing autonomy, consider the evolution of the personal automobile. The automatic transmission was one of the early subsystems that no longer needed constant human supervision. Therefore, it may be considered autonomous by our definition. Another feature that became popular with the advent of electronic controllers within the automobile is cruise control. Cruise control removed the need for constant human supervision to maintain the speed of the automobile. The cruise control feature evolved to Adaptive Cruise Control (ACC) which not only regulates speed, but also maintains a safe distance to the vehicle ahead.

ACC systems are currently evolving toward Cooperative Adaptive Cruise Control (CACC) and Advanced Driver Assistance Systems (ADAS) which are expected to eventually deliver total propulsion autonomy, when combined with other subsystems like lane-control, traction control, pedestrian safety, active collision avoidance, etc.

One way to generate autonomous behavior in a system is to make the system intelligent. Intelligence, as defined in [2], is *"..the ability of a system to act appropriately in an uncertain environment. Appropriate action is that which increases the probability of success. Success is the achievement of behavioral subgoals that support the system's ultimate goal"*. The system's ultimate goal and the criteria of success may be defined external to the intelligent system. For example, the ultimate goal of an intelligent, autonomous car may be to transport the user from point A to point B within a city, with the criteria of success being zero accidents and strict adherence to local traffic laws. The goal and the success criteria are defined by the user of the car, not by the car itself.

It should be noted that, strictly speaking, autonomy does not imply intelligence. Intelligence may imply autonomy, but may also imply partial autonomy along with humans in the loop, for selected decisions and actions. What intelligent autonomy does imply, however, is the presence of decisional processes within the machine. The notions of deliberation, planning and decision-making are inextricably a part of intelligent machine autonomy.

The achievement of intelligent machine autonomy requires substantial technical results from primarily two areas: Algorithms and Architecture. Algorithms encode the formal, mathematical knowledge for solving specific problems in a series of computation steps. For example, a sensor fusion algorithm may be used to estimate the current acceleration in an automobile. The Architecture, on the other hand, provides all the necessary principles and infrastructure for executing the algorithms, modularizing the system and communicating data to/from the executing algorithms. A third requirement, from the perspective of industrial adoption, is the availability of cost effective sensors.

In this paper, we focus exclusively on architectures for intelligent autonomous systems. In particular, we present some key challenges for these architectures, from the perspective of industrial product development. This perspective is concerned with the development of products that may be mass produced and sold to end users. Therefore, it has to emphasize topics like safety, certification, development methodology, standards compliance, reliability and cost effectiveness. The challenges presented in this paper were partly elicited from a workshop[3] conducted at KTH The Royal Institute of Technology in Stockholm, Sweden. The workshop covered topics like bottlenecks to autonomy, safety/certification and verification of autonomous products. It was attended by about 65 participants, most of whom were senior system architects from a broad cross-section of Swedish industry, including automotive, aviation, telecom and industrial automation.

The rest of this paper is organized as follows: section 2 provides a brief description of the term 'system architecture' and its various aspects, in order to

provide an unambiguous context for the entire paper. Section 3 presents some of the bottlenecks to achieving system autonomy, from an architectural perspective. It includes, among other things, a discussion of user interaction, system complexity, safety and safety standards. Finally, Section 4 has a brief discussion on the way forward towards autonomy.

2 Overview of System Architecture

The term 'architecture' is described by ISO42010:2011 as *"fundamental concepts or properties of a system in its environment, embodied in its elements, relationships and principles of its design and evolution."*. In practice, one way to think of system architecture is to decompose it into its conceptual and technical design aspects[4, 5], as shown in Figure 1.

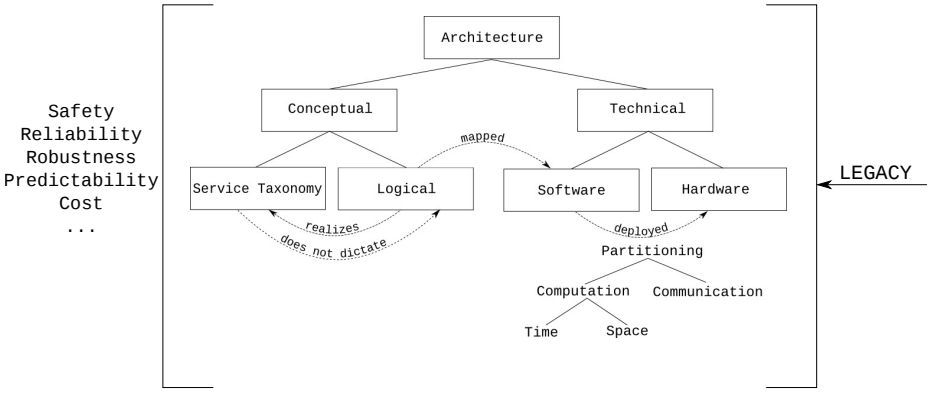

Fig. 1. An overview of system architecture

The content of Figure 1 as well as the description of the content that follows, should by no means be construed as comprehensive. However, it provides an overview with sufficient detail to begin reasoning about system intelligence and autonomy.

Service Taxonomy describes the hierarchy of services/features as seen by the **user** of the system. It describes the externally visible (sub-)behaviors of the system without any reference to how they are internally realized.

Logical architecture shows the logical decomposition of the system into its components[6] and sub-components as well as the data-flows between them. It is commonly referred to as the 'system block diagram' and it *realizes* the behavior/features defined by the service taxonomy. However, no mention is made of how the system components are actually implemented in terms of software and hardware.

Software architecture describes all the software code (components, services, layers, middleware, ...) within the system. Relevant blocks of the logical architecture are *mapped* onto individual pieces of software code which realizes the functionality of the blocks.

Hardware architecture describes the individual computer processing units and the communication channels between them. The software architecture is *deployed* on the hardware architecture.

Early system architectures consisted of the independent development of features in self-contained hardware platforms connected to a common communication bus. Such architectures are known as **federated** architectures[7]. In recent times, the trend is to permit a single hardware computer unit to execute multiple software tasks each of which may realize different system features and may have different criticality. This is achieved by means of robust mechanisms that partition and allocate resources like processor cycles, memory, i/o, communication channels etc. in space and time among the individual software tasks. For example, one processor may execute a set of 5 tasks, where each task executes for a fixed period of time and then gets replaced by the next task in the set. Such architectures which provide shared and partitioned access to resources are termed **integrated** architectures[8]. Integrated architectures offer substantial benefits in terms of cost, resource utilization and weight saving. On the other hand, they also necessitate careful management of resources and make it more difficult to analyze and assure overall system behavior.

A system architecture is additionally characterized by the so called 'extra-functional' properties like system safety, reliability, maintainability, evolvability, etc. These extra-functional properties are applicable across the entire system design and are influenced by decisions made throughout the conceptual and technical architecture. Furthermore, modern industrial product architectures are rarely 'clean-sheet' designs and therefore a significant concern in systems architecting is the integration of legacy designs into the architecture. An example of architecture legacy is that in many automobiles, the cruise control subsystem is still a part of the engine management system for historic reasons.

Industrial exploitation of architecture depends on the simultaneous availability of *principles, technologies and tools*. Academic research on system architectures usually emphasizes the principles, whereas practicing engineers need reliable tools to implement the principles and technologies.

Given the above understanding of system architecture, a relevant question is: Where to start with system autonomy? The subsequent sections present some of the identified challenges to system autonomy, from the architectural perspective.

3 Autonomy bottlenecks

The bottlenecks identified in this section are not exclusive to autonomous systems alone, but they are especially pertinent to safety critical autonomous systems.

3.1 Constructing and maintaining the world model

Intelligent autonomous systems need a model of their internal and external environments to correctly reason about action choices and make decisions. At all times, the model needs to reflect the real environment, typically via strong statistical correlations. For example, the correctness of decisions and actions made by a self-driving car depends heavily on the accuracy of the road objects (lane markers, traffic lights etc.), other vehicles and pedestrians populating its world model at every time instant. The world model of any moderately complex system is usually maintained by fusing redundant or partially overlapping sensor data. The self-driving car for instance, may rely on a combination of vision, radar, lidar and/or laser sensor data. Each sensor operates optimally under specific conditions and it is a challenge to ensure that the set of sensors as a whole can lead to a high-confidence world model under all conceivable operating conditions. Furthermore, sensors can fail and the systems needs to know the degradation of the world model to continue making correct decisions.

Although the fusing of sensor data to construct accurate and reliable world models is largely a matter for algorithms, there are several architecture issues surrounding the topic. Subsystems may require partial world models containing only the information necessary for the subsystem operation. The same information may be needed by several subsystems, but in differing representations. Simultaneously, it may be desirable to have a global, common world model to prevent unnecessary, replicated model building activities within individual subsystems and to have a single source of data. But the subsystems may require the model information with differing Quality of Service (QoS) requirements. For example, an active safety system needs to know information about detected oncoming vehicles with greater accuracy and frequency than a Human Machine Interface (HMI). Some subsystems may require historical information. Should the subsystem maintain that itself or should it be part of a common global world model? (The answer to that depends, among other things, on the number of current and future subsystems that need historical information.) Should the partial word models within individual subsystem be gathered together to form a common global model? Or should relevant parts of the global world model be "projected" into individual subsystems? Should a global world model be **implemented** in a distributed, redundant way? What about security and access control to the model information? Who are the readers, who are the writers, how to manage simultaneous conflicting writes... these are all questions relevant to the system architecture.

From the perspective of autonomous systems, it therefore becomes relevant to explore the design patterns, technologies and tools to build such world models.

3.2 User interaction

An autonomous system must necessarily reduce the user engagement required for system operation. If the required user engagement is maintained or higher, autonomy provides no immediate benefit. At the same time, the system needs

to conduct its operations with maximum transparency, so that the user is aware of what is going on. Unfortunately, it is not always clear what 'transparency' implies, nor do established norms exist to achieve it. An example of this can be taken from the aviation domain (which is particularly rich in such examples), with regards to the differences in throttle lever movement between modern Boeing and Airbus aircraft. The throttle levers in Boeing aircraft move during autopilot operation, providing a ready visual indication of autopilot activity. Airbus aircraft (the A320+ series) however have throttle levers which can be set to one of several 'gated' positions and do not move, regardless of autopilot activity. After two decades of debate, there is no clear consensus on which design is 'clearly better'.

Extensive discussions on introducing autonomy, whether the introduced autonomy should be high or low, and the exact role of autonomy are especially common in the aviation domain, within the larger discourse on automation[9]. Several perspectives on how to approach automation continue to coexist. These perspectives range from the long-held belief that automation should be matched to functions unsuitable for human control[10] to the more recent view that automated systems may be analyzed as team members[9]. Each perspective highlights different user interaction problems which are very much relevant to the discussion of autonomy, such as *mode confusion*[11, 12], moving interaction to times of high workload, failure after reaching or nearing the point of no return, *automation bias*, etc. While the discussions so far have provided balanced views on automation(autonomy), clear consensus on user interaction remains elusive.

Automation and autonomy, though intended to enhance human capabilities, often degrade them. For example, a recent Federal Aviation Administration (FAA) report[13] concludes that for pilots, in addition to degradation of basic 'stick-and-rudder' flying skills, autoflight systems could lead to degradation of the pilot's ability to quickly recover the aircraft from an undesired state. Worse still, lack of consensus on user interaction leads to functionally similar, yet subtly different autonomous systems. This makes it problematic to rely on past experience when human operators migrate between similar systems. Relying on faulty automation might easily be framed as over-reliance and misuse, while relying on correct automation that still allows for mistakes might be framed as under-reliance or disuse[9]. Focusing primarily on the human-in-the-loop as the problem is too frequently a "convenient solution".

From the perspective of autonomous systems, it therefore becomes relevant to explore flexible architectures that allow for different user interaction paradigms while maintaining maximum transparency and a meaningful exposure of error handling strategies such as graceful degradation.

3.3 Complexity and Feature Interaction

The critical concerns of world modeling and user interaction highlighted above, pinpoint another problem area. Federated and integrated architectures are meant to decentralize the logical architecture and allow for treating different functionality in isolation. However, autonomy seems to be pushing in the other direction,

requiring an increased communication between different subsystems. The result is an increased architectural complexity, where the architecture is required to simultaneously isolate and bring together different parts of the system. This, coupled with the increasing number of (often conflicting) goals within an autonomous system, significantly raises the *cognitive complexity*[14] of the system. Increasing complexity has a direct and negative impact on test case coverage, verification and validation of the system.

One of the consequences of complexity is *feature interaction*. A feature interaction is said to occur when the operation of a subsystem/feature interferes with the operation of another subsystem/feature leading to unexpected and undesirable system level behavior[15, 16]. An example of feature interaction in the automotive domain would be the simultaneous activation of the brakes and throttle. For cyber-physical systems in particular, the external physical environment often 'completes the loop'[17]. For the brake and throttle example just mentioned, the two subsystems may be completely independent in their operation, but the physical laws of the external world dictate that their operation affects one and the same variable viz. the vehicle's speed.

The feature interaction problem needs to be jointly considered by both algorithms and architecture. From an architectural perspective, it can be addressed by two complementary approaches[18]. The first approach is 'correctness by construction' which refers to the principles and mechanisms of composing systems out of subsystems in a way that there are no unexpected side effects or emergent behavior. The complementing approach is to formally represent both the architecture and a feature interaction and use model checking and verification methods to search for the feature interaction in the architecture. Once found, the interaction can be eliminated by a variety of problem specific approaches. Both the approaches above seek to detect and eliminate feature interactions during the design phase. However, all feature interactions may not be detected or eliminated and therefore intelligent autonomous systems need mechanisms to resolve feature interactions during system operation (a.k.a 'runtime'). Several ways have been proposed to detect and resolve feature interactions at runtime, mostly within networking and telecom domains(example: [19, 20]), however their applicability to industrial embedded systems has been limited by the required guarantees for robustness and safety.

3.4 'Extra-functional' properties

Extra-functional properties are those characteristics of the system which are not directly related to the functionality offered by the system to its user. They are often related to the quality attributes of the system. Examples of extra-functional properties are safety, reliability, maintainability, flexibility etc. Autonomy has a significant impact on the machine's architecture and consequently the extra-functional properties are also affected. As an example, safety (defined as freedom from unacceptable risk[21]), is profoundly impacted by the bottlenecks discussed so far in this paper. This subsection presents a quick overview of selected extra-functional properties that affect autonomous architectures.

Redundancy Safety engineers frequently rely on the availability of a human operator to take control, in case the system is unable to cope with its operational environment. The user interaction problems associated with autonomy, along with the fact that a human may simply not be available to take control, means that autonomous systems need to be designed to a greater degree of robustness. Such robustness is often added by means of providing redundancy for critical sensors, actuators as well as communication and computation facilities.

Adding redundancy to existing systems brings up sensitive issues related to cost. Also, existing machinery often simply lacks the physical space to accommodate redundant units (geometry constraints). Therefore, application of traditional redundancy measures is often not feasible for evolving autonomous architectures. New thinking is needed to add redundancy to the architecture while avoiding mere replication of the subsystems under question. Research related to dynamically reconfigurable embedded systems may yield promising results for this area. This would make it possible, for example, to migrate at runtime, software executing on a failing hardware unit to another hardware unit which was hitherto in 'hot standby' mode. Architectures like IMA-2G within the avionics domain already have limited reconfigurability as one of the development goals[22].

Architectures can also be setup to exploit the *inherent redundancy* manifested by many machines. For example, the engine in modern heavy vehicles is already used for braking; in a multi-engine aircraft, differential thrust can be used for directional control.

Determinism/Predictability Safety critical systems are usually required to demonstrate deterministic operation. For autonomous systems, it is a matter of debate and semantics whether autonomy involves a tradeoff with determinism. It is clear, however, that the issue of complexity discussed in subsection 3.3 will make it more difficult to establish determinism.

Furthermore, while the overall behavior of an autonomous system may remain predictable within certain boundaries, the system may not be absolutely deterministic within those bounds. For example, it may be assured that the collision avoidance function of an automotive active safety system will make a best effort to avoid a head-on collision with an oncoming vehicle. However, the exact trajectory which the vehicle will undertake depends on many factors and cannot be deterministically calculated in advance.

If the behavior of an individual machine can not be deterministically predicted, it is unlikely that the interactions of that machine with a heterogeneous mix of other autonomous, manual and remotely operated machines will be deterministic. Such interaction scenarios are already being envisioned in, for example, upcoming Intelligent Transport Systems (ITS) where (semi-)autonomous vehicles will coexist with those having human drivers.

This leaves designers and architects with an open question: What is the extent of permissible unpredictability within the system? This question is not explicitly answered by standards and certification requirements.

Safety Standards for autonomous systems Safety-related concerns are likely to result in requirements for new ways to develop and assure the safety of intelligent, autonomous systems. This might require updates to safety standards, such as:

- IEC 61508[21]. This standard is applicable to *most* safety-related control systems. It specifies different sets of requirements on the design and development of end products based on the required Risk Reduction Level (RRL), i.e. based on the perceived criticality of the end product. These levels are called safety integrity levels (SIL) with SIL 1 providing the lowest risk reduction and SIL 4 the highest.
- ISO 26262[23]. This standard is a specialized version of IEC 61508, adopted to the automotive domain. In this standard the RRLs are called Automotive SIL (ASIL), with ASIL A providing the lowest risk reduction and ASIL D the highest.
- ISO 13849-1[24]. This standard is only applicable on machinery, as defined in 2006/42/EC (the European machinery directive). In this standard the RRLs are called Performance Levels (PL), with PL A providing the lowest risk reduction and PL E the highest.

None of these standards are immediately applicable to autonomous systems. The main reason for this is that they rely on techniques for hazard and risk analysis in which human involvement is an important factor in the RRL estimation. ISO2626 for instance directly uses the anticipated capability of a driver to control unexpected vehicle behavior caused by major system failure as a major input to calculating the required RRL (ASIL) on the system. This reasoning is not applicable for autonomous and intelligent functions. It is not reasonable to assume that the driver has his total attention fixed on the current traffic situation if he is traveling in an autonomous car. On the other hand, increased intelligence could be made to partially replicate the actions provided by an expert human operator. However, this would require changes to the safety standards in terms of new concepts of controllability where the machine intelligence replaces the human being.

Another reason is the complexity of the environment in which many autonomous functions will operate or are expected to operate in within the immediate future. An important example of such an environment is the Intelligent Transportation System, which is expected to be deployed in the EU during the upcoming decade. With the standards above calling for a worst-case analysis centered on each separate vehicle many of the envisioned benefits in regard to traffic efficiency might be lost, for instance by enforcing safety margins with unnecessarily low speed or large distance restrictions between vehicles. Such kinds of safety margins can also lead to conditions where the transport system as a whole is prone to unacceptable high risks, mainly due to the fact that a violation of such safety margins is often not really risky in many practical situations. If the drivers of individual vehicles can bypass the functions implementing such safety margins to the benefits of the practical situations, they are likely to start doing so.

A third reason is that architectures for safety critical systems strongly emphasize static specification. Usually, system components and their inter-connections, communication channels and the data exchanged over those channels are all specified prior to system execution and cannot be changed at runtime. Autonomous systems on the other hand can benefit from dynamic, self-adapting and self-evolving architectures that respond to the operational situation and environmental inputs. Components can be started and shut down as the system functions, the inter-component connections can be modified at runtime, data-flows can be rerouted and executing code could be migrated from one component to another. Such dynamism is not permitted by existing safety standards and certification processes, mostly because of the impact on determinism/predictability.

4 Discussion

In order to satisfactorily address the autonomy challenge, it is necessary to investigate the implications of autonomy on different aspects of the system architecture. Where do crucial shortcomings exist? What are the immediate bottlenecks? From an industrial perspective, which solutions will yield the greatest usable results? Within a specified industrial domain (e.g. Automotive) is there an identified prioritization of issues obstructing autonomous behavior? These were some of the questions posed during our workshop on Autonomy. This paper has provided an initial overview of the pertinent challenges for industrial autonomous machines.

Autonomy requirements drive the adoption of new design principles and technologies, and often necessitate restructuring of the hierarchies and organization of existing systems. In this scenario, a significant challenge is that of *legacy integration*. Most of the systems that are evolving towards autonomous operation already have existing subsystems that fulfill their specific roles. These subsystems in turn may exhibit differing levels of autonomy (need for human supervision). When architecting for overall system autonomy, it may not be possible to shuffle around the contents of these subsystems in the overall logical architecture, or to modify their software/hardware implementations.

As architectures evolve towards autonomy, their information management aspect will gain greater importance. Traditionally, architectures have focused on partitioning and minimizing the data flow across partitions. Autonomy may well be viewed as a multitude of tasks cooperating closely by exchanging data. This data needs to be filtered, abstracted and approximated as it flows around within the system. Communication of high resolution information is architecturally expensive and so it may need to be meaningfully degraded as it flows away from core components, as a tradeoff with better Quality of Service. It would fall on the architecture to realize any such data-centric designs and to manage the information entropy within the system.

In section 3 we identified the following topics as currently representing bottlenecks: world model, user interaction, complexity, feature interaction, redundancy and safety. Several of these topics point towards the need for suitable reference

models, patterns and architectures, that could provide guidelines that address many of these challenges. A definition of various functionalities and their interfaces related to intelligent autonomous systems, would facilitate the creation of components, systems integration, prototypes, experiments, and may indeed, in the end, pave way for a new market for various AI-related functionalities.

Solutions in the form of domain specific reference architectures would be one way to guide the standardization and development of autonomous systems.

4.1 Safety aspects of autonomous systems

Current engineering techniques enable safety critical systems to be developed and deployed, albeit at a very high cost. Moreover, the acceptance for accidents caused by intelligent autonomous machines will most likely be much much lower compared to accidents directly attributable to humans. There are thus strong needs to consider system designs that can be formally verified and proved safe. This drives the verification aspect of architectures. At the same time, as architectures explode in complexity, *post hoc* verification becomes increasingly difficult. Therefore, methods for 'correctness by construction' will play a more significant role. This includes, among others, model based system development.

In regard to the weaknesses of the current safety standards, one way forward is to call for new hazard and risk analysis methods and the inclusion of detailed guidelines on how to systematically motivate restrictions in the degree of freedom of the autonomous system functionality with respect to different situations. The possibility of this happening is unclear, however, since these standards mainly capture and rank best practices in the industry. Manufacturing safe, intelligent autonomous systems will require these techniques to be introduced in a stepwise fashion, not through a slow process of being accepted by more and more industrial players. A more feasible way forward might be to push for acceptance via the large automotive consortia.

4.2 The need for multidisciplinary and cross-domain efforts

At the Stockholm workshop, the following shared challenges and gaps became apparent.

- The workshop was attended by several industrial companies representing developers of cars, trucks, heavy machinery, aircraft and military systems. It was interesting to note that telecommunication companies were also present. They expressed interest in architectures for autonomous systems - where they perceived that challenges in developing such architectures were shared with other domains. The take away was that the architecting challenges have much in common. Networking across domains as a way of learning was perceived highly beneficial by the participants.
- Development of intelligent autonomous systems will clearly require the use of results from several academic disciplines. The academic disciplines are unfortunately largely fragmented, and disciplines such as AI, control, software,

sensing and signal processing rarely integrate across their specific theories, frameworks, tools and prototypes. Successful and more optimal design of intelligent autonomous systems will require a closer collaboration and interactions between the disciplines. This insight is not new, and is for example one of the key tenets of the Cyber-Physical Systems initiative in the U.S.A[25]. Developing demonstrators and industrial collaboration provide important means for integration as illustrated by the DARPA grand challenge and the GCDC[26] contest. Such collaboration can also help to reduce the often perceived gap between industry and academia.

5 Acknowledgments

This research was conducted within the VINNOVA (Swedish Governmental Agency for Innovation Systems) funded FUSE project. FUSE conducts research on functional safety and evolvable architectures for autonomy. The autonomy workshop from which several of this paper's insights were drawn was conducted under the aegis of the Innovative Center for Embedded Systems (ICES) at KTH, Stockholm.

References

1. Takayama, L., Ju, W., Nass, C.: Beyond dirty, dangerous and dull: what everyday people think robots should do. In: 3rd ACM/IEEE International Conference on Human-Robot Interaction. (2008) 25–32
2. Albus, J.: Outline for a theory of intelligence. Systems, Man and Cybernetics, IEEE Transactions **21**(3) (1991) 473–509
3. : ICES Workshop on Architectures for Autonomous Automotive Systems. `http://www.ices.kth.se/events.aspx?pid=3&evtKeyId=ab27fb03b44a4dd79536cd4b048d6a7b` (2014) [Online; accessed 14-February-2014].
4. Broy, M.: Model-driven architecture-centric engineering of (embedded) software intensive systems: modeling theories and architectural milestones. Innovations in Systems and Software Engineering **3**(1) (November 2006) 75–102
5. Broy, M.: Two Sides of Structuring Multi-Functional Software Systems: Function Hierarchy and Component Architecture. 5th ACIS International Conference on Software Engineering Research, Management & Applications (SERA 2007) (August 2007) 3–12
6. Bruyninckx, H., Klotzbücher, M., Hochgeschwender, N., Kraetzschmar, G., Gherardi, L., Brugali, D.: The BRICS component model. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing - SAC '13, New York, New York, USA, ACM Press (2013) 1758
7. Di Natale, M., Sangiovanni-Vincentelli, A.: Moving From Federated to Integrated Architectures in Automotive: The Role of Standards, Methods and Tools. Proceedings of the IEEE **98**(4) (April 2010) 603–620
8. Watkins, C.B., Walter, R.: Transitioning from federated avionics architectures to Integrated Modular Avionics. In: 2007 IEEE/AIAA 26th Digital Avionics Systems Conference, IEEE (October 2007) 2.A.1–1–2.A.1–10

9. Pritchett, A.R.: Aviation automation: General perspectives and specific guidance for the design of modes and alerts. Reviews of Human Factors and Ergonomics **5**(1) (2009) 82–113
10. Fitts, P.M., Viteles, M.S., Barr, N.L., Brimhall, D.R., Finch, G., Gardner, E., Grether, W.F., Kellum, W.E., Stevens, S.S.: Human engineering for an effective air navigation and traffic control system. Technical Report ADB815893, Ohio State University Research Foundation (1951)
11. Bredereke, J., Lankenau, A.: A rigorous view of mode confusion. Computer Safety, Reliability and Security **2434** (2002) 19–31
12. Rushby, J.: Modeling the Human in Human Factors Extended Abstract. In: Computer Safety, Reliability and Security. Volume 2187. (2001) 86–91
13. The PARC/CAST Flight Deck Automation WG: Operational use of flight path management systems. Technical report (2013)
14. Kopetz, H.: The Complexity Challenge in Embedded System Design. In: 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), IEEE (May 2008) 3–12
15. Cameron, E., Griffeth, N., Lin, Y.J., Nilson, M., Schnure, W., Velthuijsen, H.: A feature-interaction benchmark for IN and beyond. IEEE Communications Magazine **31**(3) (March 1993) 64–69
16. Metzger, A.: Feature interactions in embedded control systems. Computer Networks **45**(5) (August 2004) 625–644
17. Juarez Dominguez, A.L.: Feature Interaction Detection in the Automotive Domain. In: 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, IEEE (September 2008) 521–524
18. Hay, J., Atlee, J.: Composing features and resolving interactions. ACM SIGSOFT Software Engineering Notes **25**(6) (November 2000) 110–119
19. Tsang, S., Magill, E.: Learning to detect and avoid run-time feature interactions in intelligent networks. IEEE Transactions on Software Engineering **24**(10) (1998) 818–830
20. Velthuijsen, H.: Distributed artificial intelligence for runtime feature-interaction resolution. Computer (1993)
21. : IEC 61508:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems (2010)
22. Bieber, P., Boniol, F., Boyer, M., Noulard, E., Pagetti, C.: New Challenges for Future Avionic Architectures. Aerospace Lab (4) (2012) 1–10
23. : ISO 26262:2011 Road vehicles — Functional safety (2011)
24. : SS-EN ISO 13849-1:2008 Safety of machinery – Safety-related parts of control systems – Part 1: General principles for design (ISO 13849-1:2006) (2008)
25. : Designing a digital future: Federally funded research and development in networking and information technology. Report to the president and Congress. `http://www.whitehouse.gov/sites/default/files/microsites/ostp/pcast-nitrd-report-2010.pdf` (2010) [Online report; accessed 14-February-2014].
26. Ploeg, J., Shladover, S., Nijmeijer, H., van de Wouw, N.: Introduction to the Special Issue on the 2011 Grand Cooperative Driving Challenge. IEEE Transactions on Intelligent Transportation Systems **13**(3) (September 2012) 989–993