Architecture support for automobile autonomy: A state of the art survey

Sagar Behere

Technical report
Department of Machine Design
KTH Royal Institute of Technology
SE-100 44 Stockholm

TRITA – MMK 2012:12
ISSN 1400-1179
ISRN/KTH/MMK/R-12/12-SE

Contents

1 Introduction

Archictectures for automobile autonomy are influenced by primarily three domains: 1) Automotive 2) Robotics and intelligent control and 3) General embedded systems. Automobile autonomy (chequered section in Figure 1) lies at

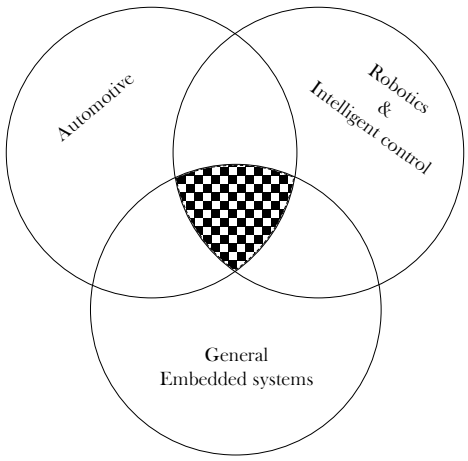

Figure 1: Automobile autonomy lies at the intersection of three research areas

the intersection of these three domains and this report therefore provides an overview of related work in all three domains. Each domain is covered in a separate section of this report. Those sections are structured as follows: First, a listing of some relevant references is made. The listing consists of introductions, overviews, surveys and states of the art. The content of these references is not elaborated much; the intention is to provide a compact set of references to background material for the interested reader. A selected set of research contributions is then described to a greater degree. Each section ends with a discussion where the author's opinions about some of the presented research are expressed. These opinions typically include an analysis of the research and potential connections to automotive architecture.

After covering the related work in the three domains mentioned above, this report presents a discussion of some aspects which affect architectures of vehicles that are intended for series production. It provides a cautionary note by highlighting the fact that architectures and technologies arising from research in domains like robotics cannot be blindly applied to automotive autonomy.

2 Automotive

The growth of electronics in vehicle systems has created new engineering opportunities and challenges. In this section, we take a look at topics related to electronics, embedded systems, software and E/E architecture, all from an

automotive specific viewpoint. We also provide references to core technologies (control systems design, wireless communication etc.) that are instrumental in enabling autonomous driving and some projects that have attempted to create autonomous driving functionality in some form or another.

An introductory overview of the expanding electronic systems in the automotive domain is given in [62], where the authors focus on in-vehicle networks and electrical power demands. A quick and general overview of software in automotive systems is provided in [70], where the differences between automotive and other types of software are highlighted, together with software processes and standardization attempts in the automotive industry. A thorough coverage of architecting and modeling automotive embedded systems is provided in [61]. A roadmap of software engineering for automotive systems is presented in [79], which also covers the salient features of the automotive domain, the consequences of each salient feature and research challenges for automotive software engineering.

A comprehensive survey of literature related to autonomous and cooperative driving is presented in section 1.3 of appended publication A. This includes research in the areas of automatic control, wireless communication and smart transport infrastructures, which covers the key knowledge and technologies that enable cooperative, autonomous driving. It also includes references to and results of large on-going or recently completed projects which aim to integrate the individual research areas and create technology demonstrators.

Some key issues affecting the development of automotive electrical and electronic (E/E) architectures are identified in [95]. An important issue that is uncovered is that architectural decisions are largely influenced by history and this is reflected in technology choices as well as the organization. The authors point out that existing automotive architectures were fundamentally designed in the mid 1990s and that there is a need to design architectures that are driven more by current needs than by legacy designs and decisions. Another highlighted issue is the lack of a clear, long term architectural strategy within an organization. A third issue underscores the fact that an established process for architecture development is missing. Some more issues are also pointed out that have an indirect bearing on architectures and the architecting process, but whose origins lie in the business processes and software tools domains.

Some characteristics and re-engineering challenges of automotive software are identified in [87]. The characteristics cover hard real-time requirements, reliability and safety requirements, limited resources, heterogenity of domain knowledge and the existence of short development cycles under time pressure. The authors also point out that programming paradigms in automotive software development are changing from using "C code in an assembly-like manner" to the use of visual programming tools with autogenerated code via a complex toolchain. They emphasize the importance of time-triggered computation models in the automotive world and suggest that techniques for understanding blackboard architectures[47] would be very useful to the automotive domain, since communication, control and dataflow in automotive subsystems is usually realized by writing/reading shared memory areas (which is akin to blackboard

architecture).

An excellent introduction to the engineering of automotive software is provided in [27]. The authors begin by pointing out that more than 80% of the innovation in modern cars is realized via software. Then a characterization of automotive software engineering is provided by taking into consideration the idiosyncrasies of the automotive domain. This includes the market, interplay of OEMs and suppliers, heterogenity of software involved and the multi-disciplinary nature of the field. The non-technical features of automotive software which are highlighted include division of labor, long product lifetimes despite short innovation cycles and the presence of a large number of product variants. From a more technical perspective, the authors make the claim that the goal of automotive software engineering should be to differentiate between the various software domains in a vehicle (infotainment, driving functionality etc.) and offer the proper reliability, safety and security for both the software and its development processes. After a further description of the complexity of technical architectures in vehicles, the authors then describe the trends and challenges that occur precisely due to the identified characteristics. The identified future trends in functionality include crash prevention and safety, advanced energy management and advanced driver assistance systems. Another trend is the presence of integrated, comprehensive data models. This implies the presence of a vehicle-wide, distributed database, instead of the current situation where each ECU keeps local copies of data and the local copies within different ECUs may contain conflicting data about the same information. A sophisticated structural view of modeling automotive architecture is then described that encompasses different levels of abstraction ranging from user-level views down to the hardware architecture. The trends are wrapped up with a discussion of model based development, model based middleware, tool support and improvements to reliability and safety.

The state of practice in automotive architectures is to isolate functionality in independent ECUs, that are connected to a common communication bus. This is termed *federated architecture* in [33], wherein the authors argue that the problems of increasing functional complexity and cost are pushing automotive architectures towards a new paradigm, the so-called *integrated architecture.* An integrated architecture is one where a single ECU can support multiple functions, and a single function can be distributed over multiple ECUs. The integrated architecture concept is inspired by the *Integrated Modular Avionics(IMA)*[96] architecture in the avionics domain, where a similar transition from federated architectures has been initiated[97].

The design and development of component based embedded systems for automotive applications is covered in [72]. The authors have divided this paper into three categories

- Challenges to the adoption of model based technologies: The identified challenges include shortcomings of most modern tools for model-based design. For example, lack of separation between functional and architectural models, insufficient support for specification of timing constraints

and attributes, and a lack of support for the analysis of scheduling related delays. The authors also describe issues in model-to-model transformation and translation, giving an example of a model made in Simulink, UML and AUTOSAR where the execution semantics are found to differ for each case.

- A review of recent advances in component based technologies: The review focuses on timing predictability, timing isolation and the role played by standards like CAN, FlexRAY and OSEK for priority based scheduling.

- Results of a methodology for architecture exploration that is based on the concept of virtual platforms and timing analysis. The concept basically involves the optimal mapping of a system model onto the candidate execution platform instances. The optimality is based on goodness-of-fit to certain constraints and the paper focus on timing constraints and metrics.

Further exposition of the virtual platform concept from this paper is provided in [82], which includes a discussion of communication, distributed systems, composability and compositionality, especially in the context of AUTOSAR.

AUTOSAR (AUTomotive Open System ARchitecture) [15, 39] is a worldwide development partnership of car manufacturers, suppliers and other companies from the electronics, semiconductor and software industry. It not only provides a technical middleware/platform for automotive ECU development, but also includes a development methodology for the same. AUTOSAR is rapidly becoming the de-facto implementation method in the automotive industry.

Beyond AUTOSAR, engineering support for automotive embedded systems comes in the form of integrated architecture description languages (ADLs)[63], specific to the automotive domain[31]. In particular, EAST-ADL2[29], is an ADL for automotive safety and architecture modeling that supports safety requirements, faults/failures, hazards and safety constraints in the context of the ISO/DIS 26262 reference safety lifecycle.

In [35], the authors describe an experience of introducing a reference architecture in the development of automotive electronic systems. Their findings emphasize the importance of centralized development processes and the need for a unifying vehicle architectural platform, rather than having individual architectures for each vehicle project.

The DySCAS project[81, 23, 30, 77] looked at issues, architecture and middleware for dynamically self-reconfigurable embedded systems in the automotive domain. It developed a reconfigurable, adaptable, component based middleware for distributed automotive architectures. The project also produced a formalism based on timed automata for modeling resource management, including quality of service. Finally, a hierarchical reference architecture framework was presented, which uses the publish-subscribe message passing communication model.

2.1 Discussion

The issues highlighted above from [95] make a significant point: Companies must have a long term architectural strategy, which is driven by current needs rather than legacy. Legacy is also one of the drivers of the bottom up style of development processes prevalent in the automotive industry today. A bottom up process leads to the development of locally optimized solutions that necessitate late refactoring of the architecture. The future of automotive E/E architectures will be driven by progress in three main areas: Principled top down design, implementation technologies and supporting tools and techniques for model based development. It is worth noting that a principled top down architecture is unlikely to receive a clean, new implementation and therefore ways must be found to migrate legacy architectures towards the new ones. Keeping this fact in mind will probably have an influence on how the top down architecture is designed.

AUTOSAR as an implementation platform has enjoyed a certain degree of success, but it still needs more work in order to cover the needs of upcoming architectures and their description. In particular, as mentioned in [72], the AUTOSAR metamodel lacks clear and unambiguous communication and synchronization semantics and a complete timing model. This adversely affects the design time verification of component properties and prevents prediction of behavior and properties of composed components. The AUTOSAR metamodel is fairly mature in its static/structural part, but needs more support for behavioral descriptions. These will enable better component reuse and composition.

3 Intelligent control and robotics architectures

Architectures in the areas of intelligent autonomy and robotics can be broadly split into two categories[44, 78]. The first category is that of cognitive architectures, which explore issues of general intelligence. Their primary concern is the reproduction of human-like characteristics of information processing, reasoning and decision making. The second category of architectures is designed explicitly for the control of physical, embedded systems that need to operate reliably and robustly in an uncertain environment. The primary concern of this second category of architectures is with topics of real-time control, sensor fusion, error recovery etc. The two categories have a degree of overlap, which is mostly in their underlying theory of hierarchical systems. The overlap occurs because both categories use hierarchies to represent information at different abstraction levels. For example, some architectures for real-time control, like RCS[25], are organized as hierarchical graphs in which nodes at the higher levels have broader scope, longer time constants and less detail[19]. The theory of hierarchical control is well-explained in [64]. It presents some of the fundamental concepts for intelligent control, covers the abstraction of models at different control levels and presents a theory for coordination of different subsystems that are under the command of an intelligent controller. The concept of intelligent control has also evolved in the domain of classical control systems. In this domain, intelli-

gence is added as a hierarchical layer on top of a traditional control loop. The application of the control loop is not necessarily for robotics.

This section is organized into subsections for intelligent control, cognitive architectures and real-time control architectures. In the last subsection, some characteristics of these architectures and possible relationships to automotive architectures are discussed.

3.1 Intelligent control

A sketch of the theory behind intelligent control, together with some of its specific traits in outlined in [68]. A general examination of architectures for intelligent control systems is made in [67].

An excellent literature overview of intelligent autonomous control is presented in [24]. In addition to the overview, a brief history of the development of control systems is presented to motivate the necessity of autonomous controllers. Next, desirable functions, characteristics and behaviors of intelligent control systems are outlined. For example, it is stated that the control architecture should be functionally hierarchical. Highest authority should lie with the machine's operator and lower level subsystems should require a clearance from higher authority levels before executing their actions. At the same time, lowest level subsystems that monitor and reconfigure for failures should be capable of acting autonomously to enhance system safety. The paper then introduces an three layer autonomous control architecture for space vehicles. The authors also identify a number of fundamental characteristics of autonomous control theory. For example, the successive delegation of duties from higher to lower hierarchical levels results in an increasing number of distinct tasks down the hierarchy. The higher hierarchical levels are concerned with longer time horizons than lower levels and incorporate models with higher levels of abstraction. The paper is concluded with an approach to a quantitative and systematic modeling and analysis of autonomous controllers. The approach includes both differential equations as well as symbolic formalisms like finite automata.

Some definitions and structures of intelligent control systems are presented in [85]. It is postulated that the presence of high intelligence lowers the demand on precision and vice versa, and a multi-level structure representing a hierarchy in the distribution of intelligence is also presented. An integrated theory for intelligent machines is presented in [83] by the same author, where control performance of a feedback control system is expressed analogously to entropy in thermodynamics. This facilitates the treatment of all levels of an intelligent, hierarchical control system by attempting to minimize the sum of entropies at individual levels. An example of applying the theory of intelligent control to robotic manipulator is given in [84].

An outline of a general theory of intelligence is given in [17]. This is one of the seminal works in the field and evolved into the engineering of the mind[18] that intended to facilitate the development of scientific models of the mind. More practically, it resulted in the creation of a reference model architecture for intelligent systems design[21], called Real-time Control System (RCS)[25].

RCS evolved through at least four versions, and RCS-3 was adapted for the NASA/NBS Standard Reference Model Telerobot Contros System Architecture (NASREM)[20], which was developed for applications in space telerobotics.

3.2 Cognitive architectures

Surveys of artifical cognitive systems and cognitive architectures can be found in [93, 34].

The Guardian architecture[46] is a blackboard architecture[47] for controlling embedded agents. A blackboard architecture consists of a common knowledge pool, which is shared among and updated by a diverse group of agents. Initially, the problem specification is written onto the "blackboard" and then agents can iteratively post partial solutions, until eventually the whole solution is obtained. The process is similar to a group of specialists clustered around a physical blackboard in order to solve a problem. The approach enables generating a whole solution incrementally, despite the possibility that no individual agent has sufficient knowledge to solve the problem. The Guardian architecture builds upon the blackboard concept and consists of a perception/action component, which is controlled by a cognitive component. One of the architectural highlights is the ability of the cognitive component to reason about the current situation and migrate decision making to the relatively faster perception/action component.

The SOAR architecture[59] enables a system to switch between deliberative and reactive modes of reasoning. While originally a pure cognitive architecture, it has been extended with a perceptual motor interface that allows some interaction with the physical world. The Adaptive Control of Thought-Rational (ACT-R) [22] is a cognitive architecture that attempts to explain and offer insights into how all the components of mind work together to produce coherent cognition. It is more of a psychological model, which could nevertheless find interesting applications in future embedded systems design. Cypress[32] is a domain independent framework for creating agents that can accept goals and synthesize and execute complex plans while staying reactive to changes in their world. It is implemented as a loosely coupled integration of some established AI tools, together with a new, common representation for sharing knowledge between them. CLARION[92] is a model for developing a bottom-up approach to skill learning, where procedural knowledge develops first, and declarative knowledge develops later. This is different from most existing models that employ a top down approach to learning of high level skills. The Global Workspace Architecture[89] is a cognitive architecture that incorporates an approximation of consicousness as well as emotion and imagination. The CoSy architecture schema[44] enables embodied robots to perform human-like tasks of object search, object manipulation, locomotion and spatial reasoning. It contains mechanisms for the focus of attention and for dynamically assigning task priorities. However, although several physical demonstrators have been built using instantiations of the schema, the instantiations have not included support for hard real-time control. ICARUS[60] is an architecture that has been strongly influenced by results from cognitive psychology and aims to reproduce

qualitative characteristics of human behavior. It consists of specific processes and memories that the processes interact with. The most basic activity of the architecture is the so-called 'conceptual inference' which is run on every execution cycle. Conceptual inference updates long term beliefs about the state of the world, based on lists of perceived objects and their relations. The architecture also includes modules for problem solving and associated learning processes.

3.3 Real-time control architectures

Among the architectures designed expressly for controlling physical robot systems, probably the most famous one that departs from the traditional sense-calculate-actuate model is the so called Subsumption architecture[26]. This architecture does not decompose the system into functional subsystems like perception, modeling, planning, motor control etc. Rather, it advocates the development of narrowly focused subsystems that fulfil specific *tasks* of the system, like 'explore surroundings', 'identify objects' etc. Each subsystem is then optimized for the particular task it performs. Any conflicts between the tasks are resolved by an arbitration mechanism. There is no architectural support for resource management, planning or abstractions.

3T[78] is a three tier architecture that coordinates planning activities with real-time behavior for dealing with dynamic robot environments. The tiers consist of a dynamically reprogrammable set of reactive skills coordinated by a skill manager, a sequencer that (de)activates sets of skills and a deliberative planner that reasons in depth about goals, resources and timing constraints. Distribution of taks aspects across the tiers depends on four possible task dimensions: time taken, bandwidth needs, task requirements and modifiability. For example, the skills tier has a cycle time in the order of milliseconds, while the planning tier operates at tens of seconds. So if something must run in a tight loop (e.g. obstacle avoidance) then it should be a skill. 3T has been implemented on several mobile and manipulator robots and its authors claim that it offers a unifying paradigm for control of intelligent systems. To quote the authors, *"The architecture allows a robot, for example, to plan a series of activities at various locations, move among the locations carrying out the activities, and simultaneously avoid danger, maintain nominal resource levels, and accept guidance from a human supervisor."* Superficially at least, these goals seem aligned to those of a future autonomous car.

The Task Control Architecture (TCA)[90] is a framework for combining deliberative and reactive behaviors to control autonomous robots. A robot built using TCA consists of task-specific modules and a central control module. The task modules perform all robot-dependent information processing, while the central module routes messages and maintains task control information. Each task module uses some TCA specific mechanisms for specifying information about the decomposition of tasks, how tasks should be monitored and how to react to exceptional situations. The TCA has been used in over half a dozen mobile robot systems, including six-legged robots and mobile manipulators.

ATLANTIS[40] is another architecture for control of autonomous robots in

dynamic and uncertain environments. ATLANTIS also combines a reactive control mechanism with a tarditional planning system. Its shows how a traditional symbolic planner can be smoothly integrated into an embedded system for pursuing multiple goals in real time.

3.4 Discussion

An autonomous automotive architecture needs to blend concepts of both cognitive as real-time control architectures. The cognitive concepts enable perception and reasoning of sensed data, as well as planning, prioritization and sequencing of tasks. The real-time control concepts are needed for tight control over actuators and processing within time-bound and safety critical subsystems.

The split between cognition and action in the Guardian architecture[46] could be applied to automotive architectures, where the existing automotive architecture could be considered as a distributed perception/action component and a cognitive component would then have to be introduced for overall system level reasoning and control. The blackboard concept is already utilized, in some form or the other, in existing automotive subsystems. This is because any global memory can be considerd to be a 'blackboard' and global memories are a fairly common practice in embedded systems programming. However, the use of the blackboard by different agents as a way to collaboratively solve problems is probably a novel idea for programmers of automotive subsystems.

The subsumption architecture[26] has some remarkable similarities to existing automotive architectures, in the sense that the automotive architectures consist of individual subsystems (ECUs), which are optimized for their specific task. As with the subsumption architecture, there is no global resource planning and management. Neither is the vehicle as a whole partitioned into subsystems like vehicle motion planning, environment perception, etc. Such capabilities, if present, are isolated into individual ECUs, where they operate within the limited context of that ECU. A problem with the subsumption architecture, which is also heavily manifested in automotive architectures, is that as the number of tasks/behaviors grow and there is increased interaction between them, it becomes increasingly difficult to find adequate arbitration schemes, or to even predict behavior in general.

One of the biggest differences between existing automotive architectures and and the robot architectures discussed in this section, is the existence of concurrent processing in architectural components. Such processing is inherent in automotive architectures by virtue of construction. It is a given that multiple subsystems can be active in parallel, handling sensor inputs and coordinating actions among different subsystems. However, as pointed out in [44], most of the architectures discussed above support only one thread of control at a time. This is true for SOAR, ACT-R, CLARION, the subsumption architecture, 3T and ICARUS. An exception is the Global Workspace Architecture which actually revolves around having concurrently active processes.

Many of the robot architectures necessarily involve concepts which, if applied to the automotive industry, are frowned upon if not forbidden outright, by

existing automotive safety and certification considerations. This is especially true for those concepts that introduce elements of uncertainty in the behavior of the system. For example, the dynamic selection of task priorities based on the current operational context, desired behavioral goals and sensed data implies that the runtime behavior may not be predictable in advance. This would make automotive architects uncomfortable because in the automotive world, determinism has high value and the architects would rather prefer a strict, static scheduling where all execution characteristics are rigorously determined and investigated in advance. As automobiles become more autonomous, designers will have to make more robust designs that are tolerant to a certain extent of non-determinisic behavior.

4 General embedded systems and software development

Architectures for automotive E/E subsystems and robotics are both specializations of the broader category of embedded systems. Therefore, it makes sense to look at some relevant research and results in general embedded systems. This section focuses on a small number of specific results relevant to autonomy, composition and complexity management of embedded subsystems. Additionally, a subsection presents some middleware, software development frameworks and libraries that can aid the developer of embedded autonomous systems.

The autonomic nervous system of the human body has inspired an initiative for self-management of distributed computing resources. This initiative, started by IBM in 2001, is termed *Autonomic Computing*[55]. It aims to tackle the problem of increasing complexity, specifically the complexity of managing, distributed computer systems. In an autonomic system, the human operator's role is to define the policies, rules and guidelines for the self-management process. The autonomic computing concept is relevant because the E/E architecture of an autonomous vehicle is essentially a complex, distributed computer system and the role of the architect is to define the policies, rules and guidelines for self-management of the architecture. Therefore, the principles and results of autonomic computing could find application in autonomous automotive architectures. An autonomic system consists of blocks for *sensing*, knowing the *purpose* of the system and the required *know how* for operating itself. The actual operation is performed by the *Logic*, which can realize the system's purpose. An overview of autonomic computing is given in [74]. A survey of autonomic computing, including motivation, concepts and seminal research in presented in [52], where the authors conclude that all distributed system architectures will soon contain reflective and adaptive elements of autonomic computing. The key features of autonomic systems, their relation to general AI and a generic architecture for autonomic computing are discussed in [56, 98]. At a more practical level, there exists a guide to IBM's autonomic computing toolkit[54], that exemplifies how an application or system component can participate in an au-

tonomic computing environment. IBM has also defined a deployment model[3] that identifies five levels of deployment for autonomic systems, where level 1 is the existing way where management is basically manual and level 5 represents the ultimate goal of autonomic systems.

The concepts of composition and decomposition are well documented and explored in the GENESYS project[53, 16]. The project aimed to develop a cross-domain reference architecture for embedded systems that meets the requirements and constraints related to composability, networking and security, robustness, diagnosis and maintenance, integrated resource management, evolvability and self-organization. It produced an analysis of architectural requirements and a description of a cross-domain architectural style that offer good insights into the nature of the problem and characteristics of relevant solutions.

Complexity challenges in embedded systems design are described in [57]. The author argues that the complexity challenge needs to be addressed by making the system models simple and understandable, by introducing appropriate levels of abstraction. Also presented is a set of design patterns for supporting component based design of embedded systems.

4.1 Middleware and software development

A comparative evaluation of robotic software integration systems is made in [88]. Surveys of available middleware and development environments for robotics are made in [69, 36, 71, 58]. In particular, Player/Stage[41, 7] and OROCOS[28, 13] have enjoyed wide adoption by academic robotic researchers[1]. Player provides a software server for network transparent robot control, while Stage is a lightweight robot simulator. OROCOS is more oriented towards hard real-time control and software component based architectures. In OROCOS, software components can be defined by starting off from a template, and the entire system configuration can be specified via an XML file that describes the instances of each component that should be created, as well as the execution semantics of the components and the inter-component data flows. The Robot Operating System (ROS)[80, 8] is a relatively recent, open source, 'meta-operating system' for robots that is gaining popularity in the robotics community. ROS is not a real-time framework, although it can be integrated with hard real-time frameworks like OROCOS[12]. YARP (Yet Another Robot Platform)[65, 37, 66] is a communication middleware, or "plumbing", for robotic systems. It supports many forms of communication (tcp, udp, multicast, local, MPI, mjpg-over-http, XML/RPC, tcpros, ...) and in the words of its creators, *"If data is the bloodstream of your robot, then YARP is the circulatory system."* BALT & CAST[45] is a middleware for cognitive robotics that is closely related to the CoSy architectural schema[44] mentioned in section 3. CLARATy[94, 73] is a two layered architecture and software framework for robot autonomy. It consists of a Functional Layer that provides abstractions for various subsystems and a Decision Layer that can do high level reasoning about global resources and mission con-

[1] as evidenced from citations, referrals and mailing list conversations.

straints. An example of the composition of complex robot applications by using data flow integration is given in [91].

The Object Management Group's Data Distribution Service (OMG DDS)[6, 75] is a publish/subscribe communication specification for Quality of Service (QoS) based, real time data exchange between publishers and subscribers. Some architectures for distributed, real time embedded systems that use DDS, and evaluations of the implementation of the architectures are presented in [99]. The use of data centric publish/subscribe for building highly dependable, adaptive, real time system architectures is presented in [38]. Best practices for data centric programming and using DDS to integrate real world systems are described in [76]. A more general set of communication patterns for composability of components is described in [86]. ZeroMQ[14, 51] is a broker-less, intelligent transport layer that supports a very wide variety of communication patterns including publish/subscribe, N-to-N via fanout, pipelining and request/response over in-process, TCP and multicast transports. It has bindings for 30+ programming languages and supports a variety of UNIX and Windows operating systems.

The Internet Communications Engine (ICE)[9, 48] is an object-oriented middleware for building distributed systems. It offers remote procedure calls, grid computing and publish/subscribe mechanisms for a wide variety of programming languages and operating systems. It comes with a variant for resource constrained, embedded systems, called Ice-E[4]. CORBA[2] has been the traditional middleware for implemented distributed software services and has its share of detractors[49] and supporters[1]. A comparison of three middleware platforms and a discussion of when performance and scalability matters (and when it does not) is presented in [50].

5 Automobiles vs robots: architectural considerations

Autonomous automobiles could be considered as mobile robots moving around in an unstructured environment. Therefore, it is natural to expect some knowledge transfer from the robotics to the automotive domain. However, even though many algorithms (related to control, sensing, data fusion, perception, information processing etc.) migrate between robotics and automobile design, a similar migration of architectures is yet to be seen. To understand why this is the case, it is helpful to compare commerical automobiles with robot prototypes. *Prima facie*, such a comparison appears to be unfair or even incommensurable i.e. apples-vs-oranges. This is actually not the case and the comparison is necessary because the theory and methods needed to create future commercial autonomous automobiles (like cognition, artificial intelligence(AI), behavior generation etc.) have traditionally been developed by robotics researchers and implemented in robot prototypes; *prototypes*, not commercial robots. Very few of the works related to AI and cognition, referenced in the robotics state of the

art in section 3, have been commercially implemented[2] and moreover any *new* research that will impact the design of autonomous automobiles is likely to be tested out first on robot prototypes. Therefore, it is towards robot prototypes that we need to look for inspiration, even if our goal is to ultimately produce commercial automobiles. However, since our goal is commercial automobiles, we must also be aware of the differences between commercial automobiles and robot prototypes, especially those differences that make it difficult to adapt architectures from one domain to the other. Some of those differences are simply the differences between prototyping and commercialization, and they have nothing to do with either automobiles or robots. That is fine and so be it. The important thing is to know that a comparison needs to be made, the reason it needs to be made and the differences to be aware of. The root cause of the differences is secondary. In this section, we first look at the differences, followed by how the differences affect the architecture. Later in the section, we'll see an example of how an automobile architecture could look like, if designed from a mobile robotics perspective.

We can briefly summarize some important differences between commercial automobiles and robot prototypes as follows

1. **Users:** An automobile is expected to be operated everyday by users with little technical understanding of the principles underlying its construction. Simplicity of the operational interface is important and it helps if the interface follows familiar and established idioms. On the other hand, robot prototypes are typically operated by people who know far more about the robot's construction, than the average person knows about the car he or she is driving. Given the ubiquity of automobiles and their potential safety hazards as well as the fact that all automobiles have essentially the same user interface, construction and purpose, it is necessary that automobiles are uniformly simple to operate. Contrariwise, robots have widely varying construction and human interfaces and it is okay if they are complex to operate. The architecture of a machine is significantly affected by requirements of hiding operational complexity from the user and providing simple and convenient means to operate the machine. In particular, automotive architectures have been influenced by the pertinent standards, development processes and legislation evolved by the automotive industry.

2. **Legacy:** A modern automobile is an evolution of a prior product version and similarly, the automobile of the future will be based on today's product. In a scenario like this, sweeping architectural changes based on novel (and unproven) concepts are difficult to introduce. In contrast, the burden of legacy is significantly lighter (often non-existent) when designing a novel robot prototype. It is more acceptable to ignore legacy when developing a robot protype than when developing the next generation automobile platform.

[2]Most commercial robots are mindless automatons in the sense that they repetitively perform pre-programmed tasks.

3. **Development processes:** Subsystems of commercial automobiles are often designed and developed by different vendors. These subsystems are then integrated into the product via traditional and standardized communication protocols and patterns. Thus, the development model is highly distributed. Introducing major architectural changes involves propagating the changes throughout the distributed development processes. This can be commercially unfeasible. Robot prototypes are not influenced by the inertia of the development and supply chain. Introducing an architectural change in a robot prototype does not require the same financial and contractual considerations that an automobile manufacturer would have to make.

4. **Safety, standards and legislation:** The ubiquity of automobiles together with the complexity of their design make them potential safety hazards, both for the general public as well as for the occupants. Therefore, stringent legislation is in place, and many development standards exist that affect the design and implementation of the automobile. The novelty of bleeding edge tools and technologies may make it difficult to get the required safety certifications and this factor must be considered during the architecting process.

A consequence of the above points is that the architectures and technical implementations of automobile embedded systems are markedly conservative, at least in comparison with robot system prototypes. For example, the automotive industry has created a software development standard for the C programming language, MISRA C[10, 11], that should be used for the programming of safety critical subsystems. The standard has many valid points, but it also results in a reduced language subset that trades off (prohibits) some of the more advanced language features for a purported decrease in programming related errors. For example, MISRA C prohibits any form of dynamic memory allocation (*malloc(), free()* etc.), usage of *errno*, *setjmp(), longjmp()* and also requires that no use shall be made of any signal handling facilities provided by <signal.h> or functions from <stdio.h> and <time.h>. The benefits of such "safer language subsets" and especially some of the restrictions dictated by MISRA-C are at times questionable[42, 43, 5]. (In particular, [43] compares the two most recent versions of MISRA C and provides a devastating critique that concludes with, *"MISRA C 2004 ... has not solved the most fundamental problem of MISRA C 1998, viz. that its unadulterated use as a compliance document is likely to lead to* ***more*** *faults and not less ... In its present form, there is a danger that the only people to benefit from the MISRA C 2004 update will be tool vendors."*) Conservativeness is also found in automotive implementation frameworks like AUTOSAR, which does not provide native support for communication patterns like publish-subscribe, the formation/deletion of or changes to data flow connections at run time, or the transfer of opaque, weakly typed data objects between software components. Regardless of opinions on conservativeness, the fact is that certain architectures and architectural patterns from the robotics domain can be rendered un-implementable due to the technologies and restrictions fa-

vored by the automotive industry in practice. For example, it would be difficult to adopt an architecture where dataflow ports between components are created, re-routed or destroyed dynamically during runtime, or where the types and sizes of data structures exchanged between components cannot be statically specified in advance. Such facilities are quite common in architectures for autonomy, artificial intelligence, cognition and robotics and are sometimes central to their designs.

Another important distinction between automotive and robotics architectures is that automotive architectures lack system level, central software processes that orchestrate the functioning of the vehicle as a whole. Rather, the emphasis is on subsystems; automotive embedded systems mostly focus on physical subsystems in the vehicle e.g. engine, brakes, transmission etc. The architecture comprises of embedded subsystems that cater to or are responsible for these physical subsystems. Thus, there exists the Engine Management ECU, the Anti-lock Brake System ECU, the Automatic Transmission ECU and so on. These ECUs and their place in the logical hierarchy of existing automotive architectures are shown in Figure 2. The Figure illustrates that the ECUs are present at the lowermost layer of the logical hierarchy. It also shows that there are some functions, like traction control, park assist etc. that may utilize more than one ECU and these are logically placed above the layer that contains the individual ECUs. Functions like these are a 'thin layer' on top of the ECUs; their place in the logical hierarchy is a result of necessity (they have to be where they are, because they can not be any lower down i.e. isolated within one of the existing ECUs). As shown in Figure 2, there exists no comprehensive

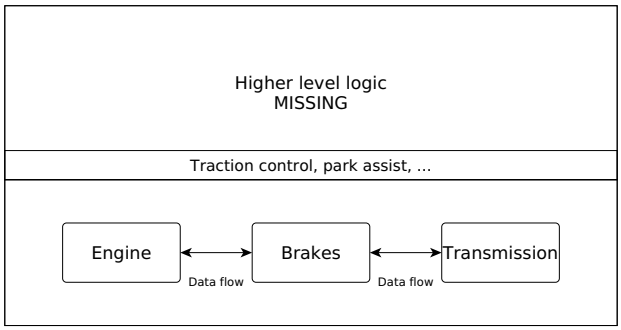

Figure 2: Logical hierarchy for an automobile

logic on top of this layer that drives the ECUs in accordance with some system level goals. Rather, each ECU 'does its own thing', perhaps with some limited interaction with other ECUs.

The bottom up approach to automotive architecture that is evident so far must be complemented with top down thinking of systems, which is missing. In contrast, architectures for (mobile) robots often have a strong top-down aspect and are designed around system level notions of functionality, like motion,

navigation, task planning etc.

If we think of a car as a mobile robot, how would its architecture look like? One example is shown in Figure 3 , which primarily shows the logical elements

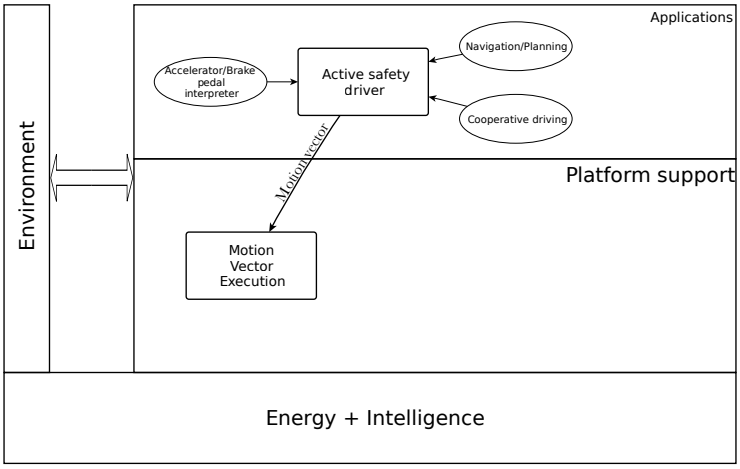

Figure 3: Logical architecture for a robotic car

involved in vehicle motion. There are two main elements involved: one for generating a motion vector and another for making the vehicle move along the generated motion vector. The motion vector generator, referred to as the 'Active safety driver' is the element that at all times provides the motion vector to be realized. This element can, in the simplest case, use inputs from the accelerator, brake and steering wheel to generate the basic motion vector setpoint. This basic setpoint can then be modified based on the current driving situation, subsystem states, operational constraints, physical laws etc. It is referred to as 'Active Safety Driver' because it can override or saturate inputs which are determined to be unsafe for the current operating situation. For advanced functionality, arbitration can be performed between the basic pedal and steering inputs, and inputs from a navigation/planning subsystem and/or a cooperative driving subsystem. The motion vector which is eventually generated after considering all necessary inputs and factors is then handed over to the subsystem that can move the vehicle along that vector. This subsystem internally utilizes the physical subsystems like engine, brakes and the transmission. At all time, data is constantly exchanged with an element representing the internal and external environment. Further, it is even possible to think of motion vector execution as a 'platform service' and the motion vector generation as an application running on top of this platform. Comparison of Figure 3 to Figure 2, shows that the Engine, Brake and Transmission ECUs (i.e. the lowermost layer in the Figure 2) are abstracted in the Motion Vector Execution component of Figure 3, which precisely emphasizes the higher level logic that was deemed missing in Figure 2.

As automobiles become more autonomous, it is likely that more and more system level logic needs to be incorporated and the functionality of existing ECUs will be shuffled around to fit a top down driven, system oriented architecture. However, considerations of legacy make it difficult to begin with clean implementations of such a top down architecture. Therefore, there need to be ways to migrate the implementations of existing, bottom up architectures towards a top down architecture that is conceptually 'designed-from-scratch'.

Bibliography

[1] Response to 'The Rise and Fall of CORBA' by Michi Henning, . URL http://www.dre.vanderbilt.edu/~schmidt/corba-response.html.

[2] CORBA, . URL http://www.corba.org/.

[3] IBM Unveils New Autonomic Computing Deployment Model. URL http://www-03.ibm.com/press/us/en/pressrelease/464.wss.

[4] Ice-E. URL http://zeroc.com/icee/index.html.

[5] Comments on the MISRA C coding guidelines. URL http://www.knosof.co.uk/misracom.html.

[6] The OMG Data Distribution Portal. URL http://portals.omg.org/dds/.

[7] The Player Project. URL http://playerstage.sourceforge.net/.

[8] ROS: The Robot Operating System. URL http://www.ros.org.

[9] The Internet Communications Engine (ICE). URL http://www.zeroc.com/ice.html.

[10] *MISRA-C:2004 Guidelines for the use of the C language in critical systems*. 2004. ISBN 0 9524156 2 3.

[11] Guidelines for the use of the programming language C in vehicle based systems, 2004. URL http://www.misra-c.com/MISRAChome/tabid/181/Default.aspx.

[12] Orocos RTT and ROS integrated, 2009. URL http://www.willowgarage.com/blog/2009/06/10/orocos-rtt-and-ros-integrated.

[13] Open Robot Control Software. http://www.orocos.org, 2011. URL http://www.orocos.org.

[14] ZeroMQ: The Intelligent Transport Layer http://www.zeromq.org/, 2012. URL http://www.zeromq.org/.

[15] AUTOSAR Consortium, 2013. URL http://www.autosar.org.

[16] GENESYS - GENeric Embedded SYStem Platform, 2013. URL http://www.genesys-platform.eu/.

[17] J. Albus. Outline for a theory of intelligence. *Systems, Man and Cybernetics, IEEE Transactions ...*, 21(3):473–509, 1991. ISSN 00189472. doi: 10.1109/21.97471. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=97471http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=97471.

[18] J Albus. The engineering of mind. *Information Sciences*, 117(1-2):1–18, July 1999. ISSN 00200255. doi: 10.1016/S0020-0255(98)10102-0. URL http://linkinghub.elsevier.com/retrieve/pii/S0020025598101020.

[19] J. Albus and F.G. Proctor. A reference model architecture for intelligent hybrid control systems. In *Proceedings of the 1996 Triennial World Congress, International Federation of Automatic Control (IFAC)*, 1996. URL http://www.isd.mel.nist.gov/documents/albus/ifac13.pdf.

[20] James S Albus, Ronald Lumia, J Fiala, A J Wavering, and Harry G McCain. NASREM - The NASA/NBS Standard Reference Model for Telerobot Control System Architecture. In *proceedings of the 20th International Symposium on Industrial Robots*, number NIST 1235. NIST, 1989.

[21] J.S. Albus. A reference model architecture for intelligent systems design. *An introduction to intelligent and autonomous control*, pages 27–56, 1993. URL http://www.isd.mel.nist.gov/documents/albus/Ref_Model_Arch345.pdf.

[22] John R Anderson, Daniel Bothell, Michael D Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. An integrated theory of the mind. *Psychological review*, 111(4):1036–60, October 2004. ISSN 0033-295X. doi: 10.1037/0033-295X.111.4.1036. URL http://www.ncbi.nlm.nih.gov/pubmed/15482072.

[23] Richard Anthony, Dejiu Chen, Martin Törngren, Detlef Scholle, and Martin Sanfridson. Autonomic Middleware for Automotive Embedded Systems. In Athanasios V. Vasilakos, Manish Parashar, Stamatis Karnouskos, and Witold Pedrycz, editors, *Autonomic Communication*. Springer US, Boston, MA, 2009. ISBN 978-0-387-09752-7. doi: 10.1007/978-0-387-09753-4. URL http://www.springerlink.com/index/10.1007/978-0-387-09753-4.

[24] PJ Antsaklis, KM Passino, and SJ Wang. Towards intelligent autonomous control systems: Architecture and fundamental issues. *Journal of Intelligent & Robotic Systems*, pages 315–342, 1989. URL http://www.springerlink.com/index/P86Q5832418GT7W7.pdf.

[25] Anthony J Barbera, James S Albus, and Leonard S Haynes. RCS : The NBS Real -Time Control System. 1984.

[26] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, 2(1):14–23, 1986. ISSN 0882-4967. doi: 10.1109/JRA.1986.1087032. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1087032.

[27] Manfred Broy, Ingolf H. Kruger, Alexander Pretschner, and Christian Salzmann. Engineering Automotive Software. *Proceedings of the IEEE*, 95 (2):356–373, February 2007. ISSN 0018-9219. doi: 10.1109/JPROC.2006.888386. URL http://papers.sae.org/r-361http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4142919.

[28] H. Bruyninckx. Open robot control software: the OROCOS project. *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, 3:2523–2528, 2001. doi: 10.1109/ROBOT.2001.933002. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=933002.

[29] D Chen, R Johansson, H Lönn, H Blom, M Walker, Y Papadopoulos, S Torchiaro, F Tagliabo, and A Sandberg. Integrated safety and architecture modeling for automotive embedded systems*. *e & i Elektrotechnik und Informationstechnik*, 128(6):196–202, June 2011. ISSN 0932-383X. doi: 10.1007/s00502-011-0007-7. URL http://www.springerlink.com/index/10.1007/s00502-011-0007-7.

[30] DJ Chen and R Anthony. An architectural approach to autonomics and self-management of automotive embedded electronic systems. In *4th European Congres ERTS (Embedded Real Time Software)*, pages 1–8, 2008. URL http://kth.diva-portal.org/smash/record.jsf?pid=diva2:497311.

[31] Philippe Automotive Cuenot, Patrick Frey, Rolf Johansson, Henrik Lönn, Martin Törngren, and Carl-Johan Sjöstedt. Engineering support for automotive embedded systems - Beyond AUTOSAR. (May):2008–2008, 2008.

[32] EW DAVID and LM KAREN. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental & Theoretical Artificial Intelligence*, 1995. URL http://www.tandfonline.com/doi/abs/10.1080/09528139508953802.

[33] M. Di Natale and A.L. Sangiovanni-Vincentelli. Moving From Federated to Integrated Architectures in Automotive: The Role of Standards, Methods and Tools. *Proceedings of the IEEE*, 98(4):603–620, April 2010. ISSN 0018-9219. doi: 10.1109/JPROC.2009.2039550. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5440059http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5440059.

[34] W. Duch, R.J. Oentaryo, and M. Pasquier. Cognitive Architectures: Where do we go from here? In *Artificial general intelligence*, 2008. URL http://books.google.com/books?hl=en&lr=&id=a_ZR81Z25z0C&

oi=fnd&pg=PA122&dq=Cognitive+Architectures+:+Where+do+we+go+from+here+?&ots=n15Trrs_KI&sig=rcL8hcj_ZN5k-84oBZiCvQXP_wE.

[35] Ulrik Eklund, Örjan Askerdal, Johan Granholm, Anders Alminger, and Jakob Axelsson. Experience of introducing reference architectures in the development of automotive electronic systems. In *Proceedings of the second international workshop on Software engineering for automotive systems - SEAS '05*, pages 1–6, New York, New York, USA, 2005. ACM Press. ISBN 1595931287. doi: 10.1145/1083190.1083195. URL http://portal.acm.org/citation.cfm?doid=1083190.1083195.

[36] Ayssam Elkady and Tarek Sobh. Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography. *Journal of Robotics*, 2012:1–15, 2012. ISSN 1687-9600. doi: 10.1155/2012/959013. URL http://www.hindawi.com/journals/jr/2012/959013/.

[37] Paul Fitzpatrick, Giorgio Metta, and Lorenzo Natale. Towards long-lived robot genes. *Robotics and Autonomous Systems*, 56(1):29–45, January 2008. ISSN 09218890. doi: 10.1016/j.robot.2007.09.014. URL http://linkinghub.elsevier.com/retrieve/pii/S0921889007001364.

[38] Brian Ford, Peter Bull, and Alan Grigg. Adaptive architectures for future highly dependable, real-time systems. In *7th Annual Conference on Systems Engineering Research*, volume 2009, 2009. URL http://research.rti.com/sites/default/files/2007_brian_ford_adaptive_arch_for_future_highly_dependant_RT_systems_S08-45.pdf.

[39] Simon Fürst, B M W Group, Jürgen Mössinger, Stefan Bunzel, Thomas Weber, Frank Kirschke-biller, Ford Motor Company, Klaus Lange, and Volkswagen Ag. AUTOSAR - A Worldwide Standard is on the Road. *VDI Congress*, pages 1–16, 2009. URL http://www.win.tue.nl/~mvdbrand/courses/sse/0910/AUTOSAR.pdf.

[40] Erann Gat. Integrating planning and reacting in a heterogenous asynchronous architecture for controlling real-world mobile robots. *aaai*, pages 809–815, 1992.

[41] B Gerkey, RT Vaughan, and Andrew Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the International Conference on Advanced Robotics (ICAR)*, number Icar, pages 317–323, 2003. URL http://robotics.usc.edu/~gerkey/research/final_papers/icar03-player.pdf.

[42] Les Hatton. Safer language subsets: an overview and a case history, MISRA C. *Information and Software Technology*, 46(7):465–472, June 2004. ISSN 09505849. doi: 10.1016/j.infsof.2003.09.016. URL http://linkinghub.elsevier.com/retrieve/pii/S0950584903002076.

[43] Les Hatton. Language subsetting in an industrial context: A comparison of MISRA C 1998 and MISRA C 2004. *Information and Software Technology*, 49(5):475–482, May 2007. ISSN 09505849. doi: 10.1016/j.infsof.2006.07.004. URL http://linkinghub.elsevier.com/retrieve/pii/S0950584906000991.

[44] N. Hawes, J.L. Wyatt, and A. Sloman. *An Architecture Schema for Embodied Cognitive Systems*. School of Computer Science, University of Birmingham, 2006.

[45] Nick Hawes, Michael Zillich, and Jeremy Wyatt. BALT & CAST: Middleware for Cognitive Robotics. In *RO-MAN 2007 - The 16th IEEE International Symposium on Robot and Human Interactive Communication*, pages 998–1003. IEEE, 2007. ISBN 978-1-4244-1634-9. doi: 10.1109/ROMAN.2007.4415228. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4415228http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4415228.

[46] B Hayes-Roth. An architecture for adaptive intelligent systems. *Artificial Intelligence*, pages 1–49, 1995. URL http://www.sciencedirect.com/science/article/pii/000437029400004K.

[47] Barbara Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26(3):251–321, July 1985. ISSN 00043702. doi: 10.1016/0004-3702(85)90063-3. URL http://linkinghub.elsevier.com/retrieve/pii/0004370285900633.

[48] M. Henning. A new approach to object-oriented middleware. *IEEE Internet Computing*, 8(1):66–75, January 2004. ISSN 1089-7801. doi: 10.1109/MIC.2004.1260706. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1260706.

[49] Michi Henning. The rise and fall of CORBA. Technical Report June, 2006. URL http://queue.acm.org/detail.cfm?id=1142044.

[50] Michi Henning. Choosing middleware: Why performance and scalability do (and do not) matter, 2009. URL www.zeroc.com/articles/IcePerformanceWhitePaper.pdf.

[51] Peter Hintjens and Martin Sustrik. ØMQ - Multithreading Magic. URL http://www.zeromq.org/whitepapers:multithreading-magic.

[52] MC Huebscher and JA McCann. A survey of autonomic computing-degrees, models, and applications. *ACM Computing Survey*, V:1–31, 2008. URL https://dspace.ist.utl.pt/bitstream/2295/584880/1/Autonomic.

[53] Sylvia Ilieva and Mario Zagar. GENESIS - A Framework for Global Engineering of Embedded Systems. *Genesis*, pages 87–93, 2008. doi: 10.1145/1370868.1370884. URL http://www.mrtc.mdh.se/index.php?choice=publications&id=1424.

[54] B Jacob, R Lanyon-Hogg, DK Nadgir, and AF Yassin. A practical guide to the IBM autonomic computing toolkit. 2004. URL http://www.redbooks.ibm.com/redbooks/pdfs/sg246635.pdf.

[55] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, January 2003. ISSN 0018-9162. doi: 10.1109/MC.2003.1160055. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1160055.

[56] Jana Koehler and C Giblin. On autonomic computing architectures. Technical report, 2003. URL https://www.zurich.ibm.com/pdf/csc/rz3487.pdf.

[57] Hermann Kopetz. The Complexity Challenge in Embedded System Design. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 3–12. IEEE, May 2008. ISBN 978-0-7695-3132-8. doi: 10.1109/ISORC.2008.14. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4519555http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4519555.

[58] James Kramer and Matthias Scheutz. Development environments for autonomous mobile robots: A survey. *Autonomous Robots*, pages 1–36, 2007. URL http://www.springerlink.com/index/V57531724H624440.pdf.

[59] J Laird. SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, September 1987. ISSN 00043702. doi: 10.1016/0004-3702(87)90050-6. URL http://linkinghub.elsevier.com/retrieve/pii/0004370287900506.

[60] Pat Langley and Dongkyu Choi. A unified cognitive architecture for physical agents. *Proceedings of the National Conference on Artificial . . .*, 2006. URL http://www.aaai.org/Papers/AAAI/2006/AAAI06-231.pdf.

[61] Ola Larses. *Architecting and Modeling Automotive Embedded Systems*. PhD thesis, Royal Institute of Technology, Stockholm, Sweden, 2005. URL http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Architecting+and+Modeling+Automotive+Embedded+Systems#0.

[62] G. Leen and D. Heffernan. Expanding automotive electronic systems. *Computer*, 35(1):88–93, 2002. ISSN 00189162. doi: 10.1109/2.976923. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=976923.

[63] N. Medvidovic and R.N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000. ISSN 00985589. doi: 10.1109/32.825767. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=825767.

[64] M.D. Mesarovic, D. Macko, and Y. Takahara. *Theory of Hierarchical, Multilevel Systems*. Academic Press, 1970.

[65] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. Yarp: Yet another robot platform. *Journal on Advanced Robotics*, 3(1):43–48, 2006. URL http://www.intechopen.com/source/pdfs/4161/InTech-Yarp_yet_another_robot_platform.pdf.

[66] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. YARP: Yet Another Robot platform, 2006. URL http://eris.liralab.it/yarp/http://www.intechopen.com/source/pdfs/4161/InTech-Yarp_yet_another_robot_platform.pdf.

[67] A Meystel. Architectures for intelligent control systems: The science of autonomous intelligence. In *Proceedings of 8th IEEE International Symposium on Intelligent Control*, pages 42–48. IEEE. ISBN 0-7803-1206-6. doi: 10.1109/ISIC.1993.397726. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=397726.

[68] A Meystel. Intelligent control: A sketch of the theory. *Journal of Intelligent & Robotic Systems*, (September):97–107, 1989. URL http://www.springerlink.com/index/q4786106075j8710.pdf.

[69] Nader Mohamed, Jameela Al-Jaroodi, and Imad Jawhar. Middleware for Robotics: A Survey. In *2008 IEEE Conference on Robotics, Automation and Mechatronics*, pages 736–742. IEEE, September 2008. ISBN 978-1-4244-1675-2. doi: 10.1109/RAMECH.2008.4681485. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4681485.

[70] Jürgen Mössinger. Software in Automotive Systems. *IEEE Software*, 27(2):92–94, 2010. ISSN 07407459. doi: 10.1109/MS.2010.55. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5420803.

[71] Molaletsa Namoshe, N S Tlale, C M Kumile, and G. Bright. Open middleware for robotics. *2008 15th International Conference on Mechatronics and Machine Vision in Practice*, pages 189–194, December 2008. doi: 10.1109/MMVIP.2008.4749531. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4749531.

[72] Marco Di Natale. Design and Development of Component-Based Embedded Systems for Automotive Applications. *Time*, pages 15–29, 2008.

[73] I.A.D. Nesnas, Anne Wright, Max Bajracharya, Reid Simmons, and Tara Estlin. CLARAty and challenges of developing interoperable robotic software. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 3, pages 2428–2435. IEEE, 2003. ISBN 0-7803-7860-1. doi: 10.

1109/IROS.2003.1249234. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1249234.

[74] Manish Parashar and Salim Hariri. Autonomic computing: An overview. *Unconventional Programming Paradigms*, pages 247–259, 2005. URL http://www.springerlink.com/index/8JWVM292E2N5NPMG.pdf.

[75] G. Pardo-Castellote. OMG data-distribution service: architectural overview. In *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, pages 200–206. IEEE, 2003. ISBN 0-7695-1921-0. doi: 10.1109/ICDCSW.2003.1203555. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1203555.

[76] Gerardo Pardo-castellote. Data-Centric Programming Best Practices : Using DDS to Integrate Real-World Systems. Technical Report November, 2010. URL http://community.rti.com/sites/default/files/DDS_Best_Practices_WP.pdf.

[77] Magnus Persson. *Adaptive Middleware for Self-Configurable Embedded Real-Time Systems*. Licentiate thesis, KTH Stockholm, 2009. URL http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-11608.

[78] R. Peter Bonasso, R. James Firby, Erann Gat, David Kortenkamp, David P. Miller, and Mark G. Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2-3):237–256, April 1997. ISSN 0952-813X. doi: 10.1080/095281397147103. URL http://www.tandfonline.com/doi/abs/10.1080/095281397147103.

[79] A Pretschner, M Broy, I H Kruger, and T Stauner. Software Engineering for Automotive Systems: A Roadmap. *Future of Software Engineering, 2007. FOSE '07*, pages 55–71, May 2007. doi: 10.1109/FOSE.2007.22. URL http://dx.doi.org/10.1109/FOSE.2007.22.

[80] Morgan Quigley and Brian Gerkey. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, number Figure 1, 2009. URL http://pub1.willowgarage.com/~konolige/cs225B/docs/quigley-icra2009-ros.pdf.

[81] TN Qureshi, Magnus Persson, DJ Chen, M Törngren, and L Feng. Model-Based Development of Middleware forSelf-Configurable Embedded Real Time Systems: Experiences from the DySCAS Project. In *Model-Driven Development for Distributed Real-Time Embedded Systems Summer School (MDD4DRES)*, 2009. URL http://kth.diva-portal.org/smash/record.jsf?pid=diva2:495712.

[82] Alberto Sangiovanni-Vincentelli and Marco Di Natale. Embedded System Design for Automotive Applications. *Computer*, 40(10):42–51, October 2007. ISSN 0018-9162. doi: 10.1109/MC.2007.344. URL http:

//ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4343688http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4343688.

[83] G. Saridis. Control performance as an entropy: An integrated theory for intelligent machines. In *Proceedings. 1984 IEEE International Conference on Robotics and Automation*, volume 1, pages 594–599. Institute of Electrical and Electronics Engineers. doi: 10.1109/ROBOT.1984.1087168. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1087168.

[84] G. Saridis. Intelligent robotic control. *IEEE Transactions on Automatic Control*, 28(5):547–557, May 1983. ISSN 0018-9286. doi: 10.1109/TAC.1983.1103278. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1103278.

[85] G.N. Saridis. Knowledge implementation - structures of intelligent control systems. *J. Robot. Syst.*, 5:255–268, 1988.

[86] Christian Schlegel. Communication Patterns as Key Towards Component-Based Robotics. *International Journal of Advanced Robotic Systems*, 3(1):1, 2006. ISSN 1729-8806. doi: 10.5772/5759. URL http://www.intechopen.com/journals/international_journal_of_advanced_robotic_systems/communication_patterns_as_key_towards_component-based_robotics.

[87] V. Schulte-Coerne, Andreas Thums, and Jochen Quante. Automotive Software: Characteristics and Reengineering Challenges, 2009. URL http://pi.informatik.uni-siegen.de/stt/29_2/01_Fachgruppenberichte/SRE/07-quante.pdf.

[88] Azamat Shakhimardanov and Erwin Prassler. Comparative evaluation of robotic software integration systems: A case study. *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3031–3037, October 2007. doi: 10.1109/IROS.2007.4399375. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4399375.

[89] Murray Shanahan. Consciousness, Emotion, and Imagination: A Brain-Inspired Architecture for Cognitive Robotics. In *AISB Workshop: Next Generation Approaches to Machine Consciousness*, pages 26–35, 2005.

[90] R.G. Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1):34–43, 1994. ISSN 1042296X. doi: 10.1109/70.285583. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=285583.

[91] Ruben Smits and Herman Bruyninckx. Composition of complex robot applications via data flow integration. *2011 IEEE International Conference on Robotics and Automation*, pages 5576–5580, May 2011. doi: 10.

1109/ICRA.2011.5979958. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5979958.

[92] R SUN, E MERRILL, and T PETERSON. From implicit skills to explicit knowledge: a bottom-up model of skill learning. *Cognitive Science*, 25(2): 203–244, April 2001. ISSN 03640213. doi: 10.1016/S0364-0213(01)00035-0. URL http://doi.wiley.com/10.1016/S0364-0213(01)00035-0.

[93] David Vernon, Giorgio Metta, and Giulio Sandini. A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents. *IEEE Transactions on Evolutionary Computation*, 11(2):151–180, 2007. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4141064.

[94] Richard Volpe, I. Nesnas, Tara Estlin, D. Mutz, Richard Petras, and H. Das. The CLARAty architecture for robotic autonomy. In *2001 IEEE Aerospace Conference Proceedings (Cat. No.01TH8542)*, volume 1, pages 1/121–1/132. IEEE, 2001. ISBN 0-7803-6599-2. doi: 10.1109/AERO.2001.931701. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=931701.

[95] Peter Wallin and Jakob Axelsson. A Case Study of Issues Related to Automotive E/E System Architecture Development. In *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2008)*, pages 87–95. IEEE, March 2008. ISBN 978-0-7695-3141-0. doi: 10.1109/ECBS.2008.46. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4492390http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4492390.

[96] Christopher Watkins. Integrated Modular Avionics: Managing the Allocation of Shared Intersystem Resources. In *2006 ieee/aiaa 25TH Digital Avionics Systems Conference*, pages 1–12. IEEE, October 2006. ISBN 1-4244-0378-2. doi: 10.1109/DASC.2006.313743. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4106349.

[97] Christopher B Watkins and Randy Walter. Transitioning from federated avionics architectures to Integrated Modular Avionics. In *2007 IEEE/AIAA 26th Digital Avionics Systems Conference*, pages 2.A.1–1–2.A.1–10. IEEE, October 2007. ISBN 978-1-4244-1107-8. doi: 10.1109/DASC.2007.4391842. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4391842.

[98] SR White and JE Hanson. An architectural approach to autonomic computing. In *International Conference on Autonomic Computing*, 2004. ISBN 0769521142. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1301340.

[99] Ming Xiong, Jeff Parsons, and James Edmondson. Evaluating the performance of publish/subscribe platforms for information management in distributed real-time and embedded systems. 2011. URL http://portals.omg.org/dds/sites/default/files/Evaluating_Performance_Publish_Subscribe_Platforms.pdf.