ROYAL INSTITUTE
OF TECHNOLOGY

Architecting Autonomous Automotive Systems

With an emphasis on cooperative driving

SAGAR BEHERE

Licentiate Thesis
Stockholm, Sweden, 2013

TRITA MMK 2013:06
ISSN 1400-1179
ISRN KTH/MMK/R-13/06-SE
ISBN 978-91-7501-712-9

KTH School of Industrial
Engineering and Management
SE-100 44 Stockholm
Sweden

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie licentiatexamen i maskinkonstruktion torsdagen den 25 april 2013 klockan 15.00 i seminarierum B319, Institutionen för Maskinkonstruktion, Kungl Tekniska högskolan, Brinellvägen 83, Stockholm.

Tryck: Universitetsservice US AB

Abstract

The increasing usage of electronics and software in a modern automobile enables realization of many advanced features. One such feature is autonomous driving. Autonomous driving means that a human driver's intervention is not required to drive the automobile; rather, the automobile is capable of driving itself. Achieving automobile autonomy requires research in several areas, one of which is the area of automotive electrical/electronics (E/E) architectures. These architectures deal with the design of the computer hardware and software present inside various subsystems of the vehicle, with particular attention to their interaction and modularization. The aim of this thesis is to investigate how automotive E/E architectures should be designed so that 1) it is possible to realize autonomous features and 2) a smooth transition can be made from existing E/E architectures, which have no explicit support for autonomy, to future E/E architectures that are explicitly designed for autonomy.

The thesis begins its investigation by considering the specific problem of creating autonomous behavior under cooperative driving conditions. Cooperative driving conditions are those where continuous wireless communication exists between a vehicle and its surroundings, which consist of the local road infrastructure as well as the other vehicles in the vicinity. In this work, we define an original reference architecture for cooperative driving. The reference architecture demonstrates how a subsystem with specific autonomy features can be plugged into an existing E/E architecture, in order to realize autonomous driving capabilities. Two salient features of the reference architecture are that it is minimally invasive and that it does not dictate specific implementation technologies. The reference architecture has been instantiated on two separate occasions and is the main contribution of this thesis.

Another contribution of this thesis is a novel approach to the design of general, autonomous, embedded systems architectures. The approach introduces an artificial consciousness within the architecture, that understands the overall purpose of the system and also how the different existing subsystems should work together in order to meet that purpose. This approach can enable progressive autonomy in existing embedded systems architectures, over successive design iterations.

Sammanfattning

Den ökande användningen av elektronik och mjukvara i fordon kan förverkliga många avancerade funktioner. En sådan funktion är autonom körning. Autonom körning innebär att en förare inte längre behövs för att styra fordonet; fordonet kör sig själv. Att uppnå autonomi i fordon kräver forskning inom flera områden där ett område är arkitektur inom el/elektronik (E/E). Dessa arkitekturer behandlar utformningen av hårdvara och mjukvara och specifikt systemets modularisering samt delsystemens interaktion. Syftet med denna avhandling är att undersöka hur E/E arkitekturer inom fordon bör utformas så att 1) det är möjligt att realisera autonoma funktioner och 2) en mjuk övergång kan göras från befintliga E/E arkitekturer, som inte har något uttryckligt stöd för autonomi, till framtida E/E arkitekturer som är explicit avsedda för detta.

Denna avhandling inleds genom att betrakta det specifika problemet med att skapa autonomt beteende för kooperativa körförhållanden. Kooperativa körförhållanden är sådana där trådlös kommunikation är kontinuerligt etablerad mellan ett fordon och dess omgivning, som består av det lokala vägnätet samt andra fordon i närheten. I detta arbete definierar vi en ny referensarkitektur för kooperativ körning. Referensarkitekturen visar hur ett delsystem med specifik autonomifunktionalitet kan kopplas till en befintlig E/E arkitektur för att förverkliga autonom körning. Två framträdande drag i referensarkitekturen är att den är "minimalt störande" och att den inte begränsar vilken teknologi som kan användas vid implementeringen. Referensarkitekturen har realiserats vid två tillfällen och är det främsta bidraget i denna avhandling.

Ett annat bidrag i denna avhandling är en ny metod för utformning av generalla och/eller autonoma inbyggda systemarkitekturer. Denna metod inför ett konstgjort medvetande inom arkitekturen som förstår det övergripande syftet med systemet, och hur de olika befintliga delsystemen bör arbeta tillsammans för att möta detta. Detta tillvägagångssätt möjliggör successivt införande av autonomi i befintliga systemarkitekturer.

Terminology

ADL Architecture Description Language - A computer language used to create a description of a system architecture.

Architecture A system's blueprint as reflected in the key building blocks of the system, their composition, their interplay, the resulting extra-functional properties, and so on. More formally, it is defined within a systems engineering context by ISO42010:2011 as, "..fundamental concepts or properties of a system in its environment, embodied in its elements, relationships, and in the principles of its design and evolution."[61]

Reference architecture A predefined architectural pattern, or set of patterns, possibly partially or completely instantiated, designed, and proven for use in particular business and technical contexts, together with supporting artifacts to enable their use.[68]

Automotive E/E Architecture The architecture of an automobile's Electrical/Electronic systems.

Autonomous machines Machines which can perform their task(s) with minimal human intervention.

Consciousness The quality or state of being aware, especially of something within oneself.

Cooperative driving Driving in a situation where vehicles and road infrastructure in the vicinity of a vehicle continuously exchange wireless information with the vehicle, and where this information is then used to control the motion of the vehicle.

ECU Electronic Control Unit

Intelligence The ability of a system to act appropriately in an uncertain environment, where appropriate action is that which increases the probability of success, and success is the achievement of behavioral subgoals that support the system's ultimate goal.[23]

Artificial Intelligence The science and engineering of making intelligent machines, especially intelligent computer programs.[74]

Middleware A software layer that lies between the operating system and applications of a computer system.

System A regularly interacting or interdependent group of items forming a unified whole. In our context, a system is embodied as a programmable machine, together with its included program. When referring to a system, we are concerned with both the hardware and the software of the machine, together with all the interfaces in the system, including those between the hardware and the software, between the system and its user and between the system and its environment.

Embedded system A computer system that is part of a larger system and performs some of the requirements of that system. [12]

Complex system A system that can be analyzed into many components having relatively many relations among them, so that the behavior of some components may depend on the behavior of others, and the behavior of the system cannot simply be derived from the summation of individual components' behavior.

Acknowledgements

Whom should you blame for this thesis? Traditionally, names are changed to protect the innocent but nobody associated with this work can claim innocence anymore. Here then, are the names of some incredible people who encouraged and supported my work.

Martin Törngren has been my primary supervisor and go-to guy for all sorts of problems. I often enter his office feeling slightly worried and confused, but walk away relieved and clearheaded. DeJiu Chen always has a fresh perspective on things; I turn to him whenever I feel stagnated for words and ideas. Jad El-khoury is single-handedly responsible for bringing this thesis to closure. His careful scrutiny of the drafts and insightful feedback has significantly raised the quality of the text. Moreover, working with Jad is really nice, because it is the equivalent of being handed a proper map with a 'You are here' marker drawn on it.

Björn Liljeqvist and I had a brief but stimulating association, during which many autonomy related ideas came to the forefront. Working with him gave me the opportunity to experience a very real synergy of minds and the plentiful productivity that results from it.

The GCDC 2011 team, which included Jonas Mårtensson, Dennis Sundman and Henrik Pettersson, provided a fun and interesting working environment. We had a lot of fun driving around in our big, big trucks :)

Monsieur Frédéric Loiret and I exchange opinions on Life, the Universe and Everything. His unwavering enthusiasm and positive energy have seen me through some tough days. Other friends and colleagues at the department of Machine Design have provided a pleasant working atmosphere.

Most of this work has been possible due to direct or indirect support from the ITM school at KTH, Volvo Car Corporation and Scania CV AB. Their role is gratefully acknowledged.

Finally, I would like to acknowledge the role played by my parents, who do not always understand, but always accept and support.

Contents

Appended publications

- **Publication A**
  A reference architecture for cooperative driving
  Authors: Behere, Sagar ; Törngren, Martin ; Chen DeJiu
  Provisionally accepted for publication. Journal of Systems Architecture, Special edition on Embedded Software Design

  Sagar wrote the paper. Martin and Chen provided feedback.

- **Publication B**
  The development of a cooperative heavy-duty vehicle for the GCDC 2011: Team Scoop
  Authors: Mårtensson, J.; Behere Sagar, et. al.
  Intelligent Transportation Systems, IEEE Transactions on , vol.13, no.3, pp.1033-1049, Sept. 2012 doi: 10.1109/TITS.2012.2204876

  Sagar wrote the section II on Architecture. Other authors wrote sections about their individual work related to control, communications etc.

- **Publication C**
  Scoop Technical Report: Year 2011
  Author: Sagar Behere
  Technical report, 2011, Dept, of Machine Design
  KTH TRITA - MMK2012:12, ISSN 1400-1179, ISRN/KTH/MMK/R-12/12-SE

  Sagar wrote the entire report.

- **Publication D**
  Towards Autonomous Architectures: An Automotive perspective
  Authors: Behere, Sagar (KTH) ; Liljeqvist, Björn (EIS by Semcon)
  Technical report, Oct. 2012, Dept. of Machine Design
  KTH TRITA – MMK 2012:10, ISSN 1400-1179, ISRN/KTH/MMK/R-12/10-SE

  Sagar wrote most of the text. Sagar and Björn together brainstormed to generate the ideas in the paper.

Chapter 1

Introduction

In the beginning the Universe was created. This has made a lot of people very angry and has been widely regarded as a bad move.

Douglas Adams, The Hitchhiker's Guide to the Galaxy

1.1 The big picture

We live in a world where practically all the machines that surround us contain tiny computers. The cars we drive, the airplanes we fly in, the phones we use, the ovens we cook our food in.. all contain computers. These computers control the functioning of the machine into which they are embedded. These computers are called *Embedded Systems* in engineering lexicon. More formally, an embedded (computer) system is defined by the IEEE as "*..a computer system that is part of a larger system and performs some of the requirements of that system.*"[12]

There is an increasing desire to make machines autonomous. Autonomous means that the machine is able to perform its task without direct human intervention. This is beneficial for a number of reasons. Autonomous machines have the potential to increase safety and efficiency of operations. They can operate in evironments which are hazardous to humans, or where humans simply cannot be physically present to operate the machines. Sometimes machine autonomy is desired because it enables humans to be more lazy! Regardless of the reasons that justify machine autonomy, the *means* to achieve it is through the use of computers. This is because computers make it convenient to perform calculations and take decisions based on those calculations. In the context of machines, computers are utilized in the form of embedded systems. Therefore, we can introduce the term *Autonomous Embedded Systems* and use it to refer to those embedded systems that enable machine autonomy.

An embedded system in a machine may consist of multiple subsystems and each subsystem can in turn consist of multiple sub-subsytems and so on, as illustrated in Figure 1.1. Similarly, multiple systems can come together to form a 'system of systems'. Thus, a system can be *decomposed* into its

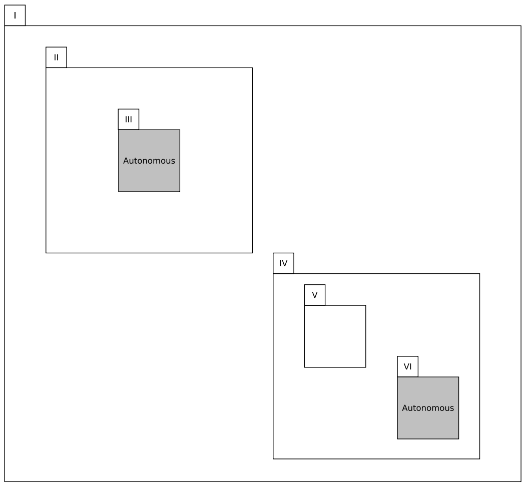


Figure 1.1: A system is composed of subsystems, some of which could be autonomous

constituent subsystems, or systems can be *composed* to form greater (or *composite*) systems. Both composition and decomposition are standard practices in the design of embedded systems. Now, if we wish to design an autonomous embedded system, it is reasonable to assume that we should examine the constituent subsystems, some of which might be autonomous. Next, we will have ways to compose the various autonomous and non-autonomous subsystems to create a system which is autonomous and without conflicts among the subsystems. This rather obvious process is sometimes overlooked in practice. Usually, it is overlooked when the system is not initially designed for autonomy, but attempts are subsequently made to introduce autonomy via incremental system modifications. In such a scenario, it is commonly observed that individual subsystems are made autonomous without much regard for the autonomy status of the overall system. Furthermore, each individual subsystem may be designed to differing *levels of autonomy*[1]. Then, when the different autonomous subsystems are composed to create a system, conflicts may arise between the individual subsystems, since they were not designed according to an overall autonomy plan. These conflicts are often resolved by ad-hoc methods, ugly hacks or local fixes, which in turn contribute to the rising complexity of the system. It becomes difficult to attain whole system autonomy and to assure essential charactersistics like system safety.

The automotive industry provides a good example of the above practice. Initially, the automobile was completely mechanical. Gradually, embedded systems were introduced into various subsystems (Figure 1.2). Examples are

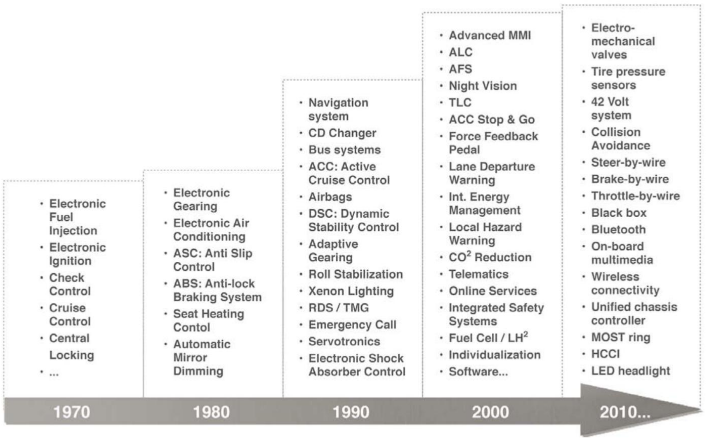


Figure 1.2: Growth of automotive embedded systems (source [35])

electronic Engine Management System, Anti-lock brakes, power windows etc. Then came the desire for autonomy. The first subsystem to go autonomous was the transmission. The so-called automatic transmission requires little human intervention during operation, and is therefore autonomous according to our definition. Next came the cruise control and traction control features,

[1]The subsystems may require differing extents of attention from the human operator.

which are enabled by embedded systems, and which incorporate a certain level of autonomy[2] within them. Notice that at this point, the automobile has three subsystems with varying levels of autonomy, but the automobile as a whole is still not fully autonomous. And already at this point, conflicts start appearing. What if the cruise control system wishes to accelerate the vehicle while simultaneously the traction control system wishes to apply the brakes? Such undesirable interaction between features is termed *feature interaction* in the engineering lexicon[63, 78]. As the complexity and autonomy of individual subsystems rises, the automotive industry is finding it increasingly difficult and costly to deal with such feature interactions and design safe and reliable vehicles.

There are technical reasons why individual subsystems can reach increasing levels of autonomy, before the system as a whole can be autonomous. These are the subsystems which have simple and well-defined roles (e.g.: the transmission) and are consequently easier to automate. In fact, autonomy of subsystems is a key enabler for overall embedded system autonomy. However, there is a clear need for new ways to construct and compose autonomous embedded systems. To begin with, there needs to be an overall "autonomy plan" that dictates their design. An overall autonomy plan is that which considers the autonomous behaviour required from the system as a whole. Such a plan needs to be present because it helps to specify and streamline the expected behavior of the individual subsystems. The individual subsystems then need to conform to the overall plan. But immediately at this point, practical problems present themselves. Specifically, many subsystems have legacies of established and proven designs. The challenge then becomes to move from existing designs towards those that are based on new theoretical principles of autonomy. Two outstanding issues thus present themselves

1. We need new designs based on new principles of autonomy

2. We have to deal with the inertia of existing designs, for reasons of compositionality and risk management

It is indeed required to create new designs based on new principles, but they should be created in such a way that they can subsume existing designs. Complementary efforts are also needed to lay out the path or means via which existing designs can then be evolved towards the new designs. The aim of this thesis is to investigate ways of creating new designs and to provide an

[2]the features are self-operating, although they may require occasional human intervention.

evolutionary path for existing designs. The aim will be fulfilled via embedded system architectures. Before moving on to a discussion of architecture, the next section provides an outline of the overall thesis.

1.2 Thesis outline

The organization of this thesis is depicted in Figure 1.3.

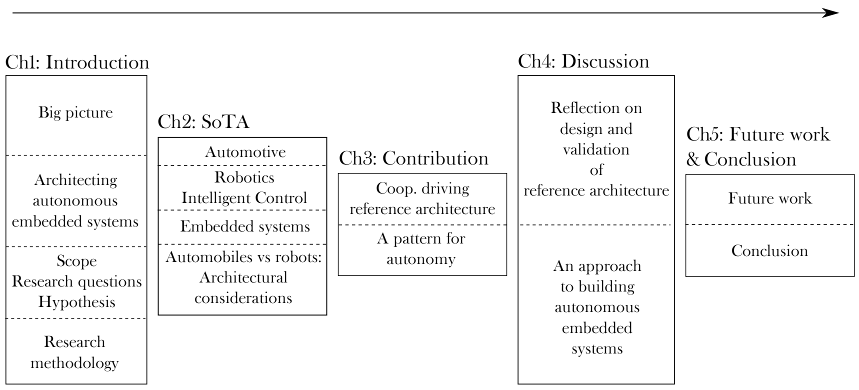

Ch1: Introduction
Big picture
Architecting autonomous embedded systems
Scope
Research questions
Hypothesis
Research methodology
Ch2: SoTA
Automotive
Robotics
Intelligent Control
Embedded systems
Automobiles vs robots: Architectural considerations
Ch3: Contribution
Coop. driving reference architecture
A pattern for autonomy
Ch4: Discussion
Reflection on design and validation of reference architecture
An approach to building autonomous embedded systems
Ch5: Future work & Conclusion
Future work
Conclusion

Figure 1.3: Thesis structure

- Chapter 1 (this chapter) begins with an introduction of the overall topic under discussion. The big picture that constitutes the motivation for this research is first presented. Section 1.3 then introduces the role of the architecture in tackling the overall problem. The discussion is narrowed down in section 1.4, by applying a series of constraints, which eventually results in the specific research question and a hypothesis. Finally, section 1.5 describes the research methodology followed during this thesis work.

- Chapter 2 presents related work, which is organized into three categories: Automotive, Robotics and Intelligent Control, and Embedded Systems. These three categories cover the bulk of the research that contributes to the theory of embedded systems autonomy as applied to automobile architecture. The chapter also presents a discussion of the architectural considerations that differentiate the automotive and robotics domains.

- Chapter 3 summarizes the principal contributions made by this thesis work. It does so by providing an overview of the contents of the appended publications.
- Chapter 4 presents a reflection of the work done, as well as an approach to building autonomous embedded systems.
- Finally, chapter 5 presents the direction that future work will take and concludes the thesis by re-visiting the research questions put forth in chapter 1 and the contributions made by the thesis.

1.3 Architecting autonomous embedded systems

While introducing the big picture in section 1.1, it was noted that there needs to be an overall autonomy plan that dictates the design of autonomous embedded systems. The autonomy plan can be expressed in the form of the system's *architecture*. The term *architecture* represents the system's blueprint and it is reflected in the key building blocks of the system, their composition, their interplay, the resulting extra-functional properties, and so on. More formally, it is defined within a systems engineering context by ISO42010:2011 as, "*..fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution.*"[61]. So we now pose the question, *"How should we architect autonomous embedded systems?"*

In order to decide how we should architect autonomous embedded systems, it is useful to understand the impact of autonomy on embedded systems design. The concept of autonomy introduces a *paradigm*[3] shift in the architecture of embedded systems. The shift occurs because autonomy affects practically every aspect of the architecture. It affects desired behavior, safety, error diagnosis and recovery, predictability as well as interaction with human users and other devices. Precisely due to their far reaching impact, autonomy-related considerations should be incorporated into the architecture

[3]The term paradigm is defined in the dictionary as, "*a pattern or model, an exemplar*". The science historian, Thomas Kuhn, sharpened the meaning of the term, when he defined a scientific paradigm as, "*universally recognized scientific achievements that, for a time, provide model problems and solutions for a community of researchers*"[69]. Kuhn saw the sciences as going through alternating periods of *normal science*, when an existing model of reality dominates a protracted period of puzzle-solving, and *revolution*, when the model of reality itself undergoes sudden drastic change. This is relevant because considerations of autonomy introduces a change in both, the "model problem" and "model solutions" for the community of researchers in embedded systems architecture.

from the start; not as an afterthought. This is especially true for large scale, distributed embedded systems. Consider, for example, the embedded system in a modern car. It is a distributed system, consisting of multiple *Electronic Control Units* (ECUs) that are connected to a set of common communication buses. As the system evolves, new features are added either as new software functions in the existing ECUs, or as new ECUs which are connected to the communication bus. Sooner or later, the bus capacity is exceeded whereupon more buses are introduced into the system. As the complexity grows, the system is partitioned. For example, the ECUs are gathered into groups such that the inter-group communication requirements are minimized. The underlying paradigm here is, "Create new functionality and stick it onto the bus." In this rather ad-hoc manner, automotive embedded systems have grown large and unwieldy. There is no single entity in the modern automobile that is aware of all the different subsystems and how they are supposed to work, in order to deliver the expected behavior. Rather, it is a case of multiple ECUs independently doing their own thing (sometimes with restricted communication with other ECUs). The lack of a single entity that is aware of the the entire system state makes it tricky to perform correct arbitration between the subsystems, respond to unanticipated conditions and generally assure coherent system behavior.

This approach of multiple, isolated, communicating units is good from a viewpoint of system growth and managing complexity via partitioning. Indeed, it may even be the natural way for such systems to have developed. However, it becomes very difficult to guarantee the satisfaction of overall autonomy requirements for a system that has been developed in such a way, especially when the development has taken place over a period of time. The isolation makes it difficult for a subsystem to have a wider context for its operation and to consider or predict the system level effects of local decisions. By making comparisons to human decision making mechanisms, it is possible to theorize that the presence of a system-aware "Ego" (i.e. a conceptual entity which is aware of the system's purpose and how the subsystems should interact to fulfill it) may help in the development of system autonomy. We will explore the concept of the system "Ego" in sections 4.2 and 5.1. For now, it is enough to observe that although separation of subsystems, partitioning and communication are *necessary* they may not be *sufficient* for architecting autonomous embedded systems. The gap could be filled by an additional architectural specification that dictates how system level autonomy should be achieved. Such an additional specification could demand fundamental design changes to existing subsystems in order to form an elegant, coherent and autonomous whole. However, fundamental design changes may not always be possible due to legacy and other concerns. Therefore, the task before the researcher in autonomous embedded systems is to create principled, top down autonomous architectures for the overall system, while keeping in mind that the implementations of these top down designed architectures need to be realized via incremental evolution of existing bottom up designs. Naturally, an evolutionary path also needs to be provided to go from the existing to the newly created autonomous architectures.

1.4 Research scope and question

The overall problems we can formulate based on the discussion so far are

1. What are the requirements, principles and patterns for architecting autonomous embedded systems which fulfill desired functional and extra-functional goals?

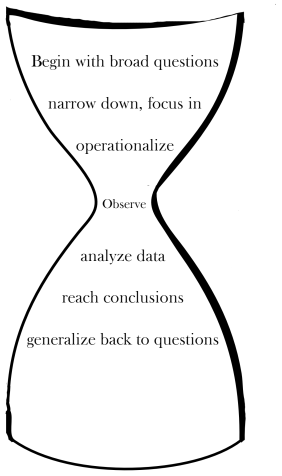
2. What are idealized, architectures for distributed, embedded systems that are expressly designed for autonomy?

3. What are the engineering and technical ways to evolve existing embedded systems architectures towards those that are expressly designed for autonomy, while minimizing impact on established, legacy designs?

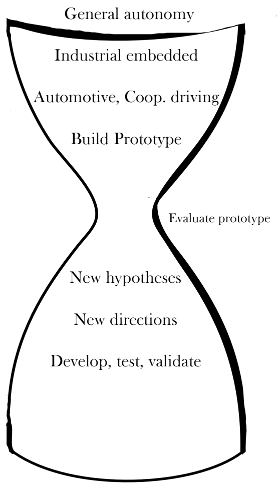
These questions are rather broad in scope. The scope needs to be more tightly focused, in order to yield manageable research questions for a licentiate thesis work that can yield tangible benefits. Such focusing, or narrowing of the scope is an established research practice and is referred to as the *hourglass* notion of research[105], as illustrated in Figure 1.4a. For this thesis work, the scope is successively focused by a sequence of decisions (the delimitations), which are illustrated in Figure 1.4b and summarized as follows

1. **We will consider industrial[4] embedded systems only.** These are embedded systems produced in a factory, with the goal of being, or being incorporated into, commercial products. This focusing is done to emphasize the difference from esoteric systems in the field of academic artificial intelligence or research robotics. Architectures for industrial embedded systems should ideally take into account matters related to

[4]Note our definition of 'industrial' in the forthcoming sentence! We do not intend the definition to connote 'heavy industry' (iron & steel, mines etc.).



(a) Hourglass notion of research



(b) Hourglass applied to this thesis

Figure 1.4: Research scope

product legacy, development methodologies (vendors, consultants, industrial practices,..), applicable standards, product lifecycle and legislation. There may also be requirements related to warranties, support and mass production.

2. **We will consider automotive embedded systems.** These are good representative examples of embedded systems that are distributed and safety critical, and whose development is affected by legislative requirements and standards. Automotive embedded systems are growing rapidly; there is a strong interest in their autonomy and the field provides rich opportunities for case studies and application scenarios.

3. **We will consider vehicle autonomy in cooperative driving scenarios.** The motion control subsystems of the vehicle[5] will be collec-

[5]We do not consider subsystems like infotainment, climate control etc. which are not

tively treated as an autonomous machine, designed to have an understanding of a variety of internal and external conditions and to drive itself without human intervention. Vehicle autonomy will only be present in cooperative driving scenarios. In such scenarios, there exists continuous wireless communication between all vehicles in a geographical area of interest. The road infrastructure also continuously communicates information about traffic lights, speed limits etc. The reason for focusing on autonomy in cooperative driving scenarios is that in these scenarios, a lot of organized information is available over the wireless. This makes the environment more structured, and consequently, autonomy is easier to achieve. In contrast, autonomy is more difficult to achieve in the type of environment that humans drive around in today, because organized information is not as readily available.

Filtering the overall problems (listed at the start of this section) through the delimitations above, we arrive at the primary research question investigated by this licentiate thesis

What is a good way to introduce autonomy in a vehicle for the purpose of cooperative driving?

In this context, a 'good way' is that which

1. requires minimum changes to the embedded systems in the existing vehicle
2. can be implemented using established industrial tools and development methodologies
3. is generic enough to serve as a reference solution for different types of vehicles

We make the hypothesis that a working answer to the above research question can be provided in the form of a *reference architecture* for autonomous, cooperative driving. Validation of the hypothesis can be performed by creating and instantiating such a reference architecture, which would also yield two further benefits. The first benefit, of course, would be the availability

directly responsible for motion of the vehicle.

of a tested and proven reference architecture for cooperative driving. The architecture itself would be a novel engineering result. The second benefit would be the emergence of promising theories and directions for architecting autonomous embedded systems. These theories may then be developed and validated during future research.

1.5 Research methodology

Engineering *design* is used as the methodology for this research. The process of engineering design is argued to be a research methodology in [43]. The argument is presented as follows[6]: "Research may be succintly defined as *an activity which develops new knowledge.* Engineering research is concerned with the development of a suitable solution to a need. The process of designing a novel prototype for addressing a need produces significant new knowledge of how that specific need can be addressed. Therefore, design is a method of creating new knowledge, which makes it a methodology for research." This thesis work attempted to design a novel engineering prototype, with the understanding that a correctly functioning prototype provides *proof by construction* that the solution developed is valid and effective. This methodology is sometimes referred to as case study based research. A more detailed discussion of engineering design as a research methodolgy, together with the key differences between scientific and engineering research can be found in[43, 42]. The overall methodology of this work is illustrated in Figure 1.5. The Figure shows that the availability of a theory for building autonomous systems leads to the design and development of one or more prototypes. The insights which are consequently obtained are then used to refine the original theory and this triggers a new cycle of prototype(s) creation. This thesis work resulted in the creation of two prototypes and the gain of consequent insights. Thus, it represents one cycle (the inner- or left- most) of Figure 1.5. It is expected that the insights obtained will lead to the creation of more prototypes as the research progresses.

Given the research question posed in section 1.4, a prototype system for autonomous, cooperative driving was designed [publications B and C]. The design process commenced with a review of state of the art and practice related to vehicle control, wireless communication as well as system architectures (hardware and software) for road vehicles and autonomous robotic systems. Subsequently and over several iterations, a system design was created which

[6]The quoted sentences that follow have been picked from various parts of the text in[43], where they do not appear in the same sequence as presented here.

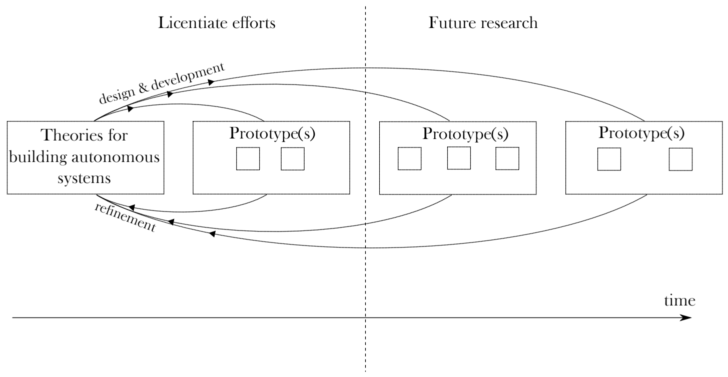


Figure 1.5: Research methodology

was deemed to provide a sufficient solution to the problem. A system prototype was developed, which was installed in an existing, commerical truck, the R730 model, from Scania CV AB. The system was then tested and validated during the Grand Cooperative Driving Challenge (GCDC) 2011 [7]. The experiences with the prototype resulted in a more generalized theoretical solution for autonomy and cooperative driving. This solution is in the form of a reference architecture [publication A], which is the primary contribution of this thesis. The reference architecture was instantiated and validated a second time on a different Scania truck model, during the CoAct-2012 Cooperative Driving Demonstration in Göteborg. The experience with the reference architecture and its instantiations also uncovered some promising hypotheses and a research direction [publication D], which are a secondary contribution of this thesis and will form the base for further research. The research direction is further discussed in section 4.2.

[7]The GCDC was an international event for demonstrating cooperative, autonomous driving on public roads, which took place in the Netherlands in May 2011.

Chapter 2

State of the Art

"When we try to pick out anything by itself, we find it hitched to everything else in the universe."

John Muir

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 2.1) lies at the intersection of these three domains and this chapter therefore provides an overview of related work in all three domains. Each domain is covered in a separate section of this chapter. 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 chapter 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

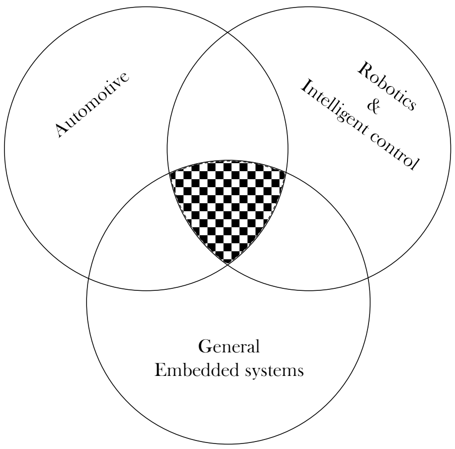


Figure 2.1: Automobile autonomy lies at the intersection of three research areas

autonomy. Finally, we show where this thesis work is positioned, in relation to the three domains.

2.1 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 [73], 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 [82], 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 [72]. A roadmap of software engineering for automotive systems is presented in [91], 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 [108]. 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 [99]. 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[54] 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 [30]. 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 [38], 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)*[109] architecture in the avionics domain, where a similar transition from federated architectures has been initiated[110].

The design and development of component based embedded systems for automotive applications is covered in [84]. 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 [94], which includes a discussion of communication, distributed systems, composability and compositionality, especially in the context of AUTOSAR.

AUTOSAR (AUTomotive Open System ARchitecture) [18, 46] 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)[75], specific to the automotive domain[36]. In particular, EAST-ADL2[33], 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 [40], 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[93, 26, 34, 89] 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.

2.1.1 Discussion

The issues highlighted above from [108] 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 [84], 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.

2.2 Intelligent control and robotics architectures

Architectures in the areas of intelligent autonomy and robotics can be broadly split into two categories[51, 90]. 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[28], are organized as hierarchical graphs in which nodes at the higher levels have broader scope, longer time constants and less detail[21]. The theory of hierarchical control is well-explained in [76]. 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, intelligence 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.

2.2.1 Intelligent control

A sketch of the theory behind intelligent control, together with some of its specific traits in outlined in [80]. A general examination of architectures for intelligent control systems is made in [79].

An excellent literature overview of intelligent autonomous control is presented in [27]. 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 [97]. 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 [95] 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 [96].

An outline of a general theory of intelligence is given in [23]. This is one of the seminal works in the field and evolved into the engineering of the mind[20] 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[24], called Real-time Control System (RCS)[28]. RCS evolved through at least four versions, and RCS-3 was adapted for the NASA/NBS Standard Reference Model Telerobot Contros System Architecture (NASREM)[22], which was developed for applications in space telerobotics.

2.2.2 Cognitive architectures

Surveys of artifical cognitive systems and cognitive architectures can be found in [106, 39].

The Guardian architecture[53] is a blackboard architecture[54] 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[70] 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) [25] 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[37] 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[104] 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[101] is a cognitive architecture that incorporates an approximation of consicousness as well as emotion and imagination. The CoSy architecture schema[51] 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[71] 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.

2.2.3 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[29]. 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[90] 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)[102] 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[47] 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.

2.2.4 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[53] 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[29] 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 [51], 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.

2.3 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*[64]. 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 [86]. A survey of autonomic computing, including motivation, concepts and seminal research in presented in [59], 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 [65, 111]. At a more practical level, there exists a guide to IBM's autonomic computing toolkit[62], that exemplifies how an application or system component can participate in an autonomic 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 mentioned in early in section 1.1 are well documented and explored in the GENESYS project[60, 19]. 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 [66]. 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.

2.3.1 Middleware and software development

A comparative evaluation of robotic software integration systems is made in [100]. Surveys of available middleware and development environments for robotics are made in [81, 41, 83, 67]. In particular, Player/Stage[48, 7] and OROCOS[31, 16] 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)[92, 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[15]. YARP (Yet Another Robot Platform)[77, 44, 9] 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[52] is a middleware for cognitive robotics that is closely related to the CoSy architectural schema[51] mentioned in section 2.2. CLARATy[107, 85] 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 constraints. An example of the composition of complex robot applications by using data flow integration is given in [103].

The Object Management Group's Data Distribution Service (OMG DDS)[6, 87] 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 [112]. The use of data centric publish/subscribe for building highly dependable, adaptive, real time system architectures is presented in [45]. Best practices for data centric programming and using DDS to integrate real world systems are described in [88]. A more general set of communication patterns for composability of components is described in [98]. ZeroMQ[17, 10] 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 trans-

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

ports. It has bindings for 30+ programming languages and supports a variety of UNIX and Windows operating systems.

The Internet Communications Engine (ICE)[11, 55] 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[56] 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 [57].

2.4 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 2.2, 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 pro-

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

totyping 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.

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[13, 14], 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[49, 50, 5]. (In particular, [50] 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 favored 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 2.2. 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 2.2, there exists no comprehensive 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 2.3 , which primarily shows the logical elements involved in vehicle motion. There are two main elements involved:

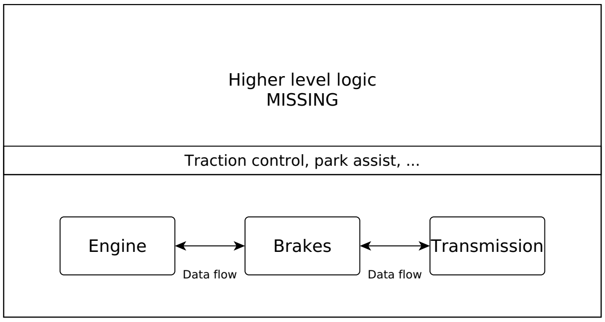


Figure 2.2: Logical hierarchy for an automobile

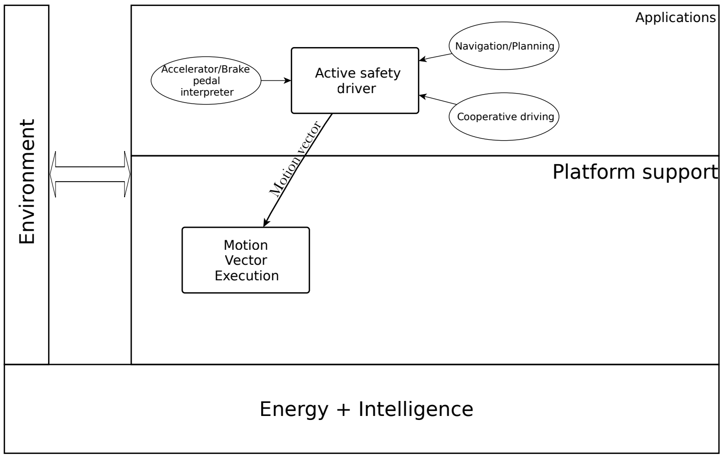


Figure 2.3: Logical architecture for a robotic car

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 2.3 to Figure 2.2, shows that the Engine, Brake and Transmission ECUs (i.e. the lowermost layer in the Figure 2.2) are abstracted in the Motion Vector Execution component of Figure 2.3, which precisely emphasizes the higher level logic that was deemed missing in Figure 2.2.

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'.

2.5 Positioning of this thesis work

Research results in cognition, artificial intelligence, machine learning, task planning and prioritaization, sensing and perception are needed when designing intelligent, autonomous automobiles. These research results have evolved to a greater extent in the research robotics domain than in the automotive domain. Therefore, it makes sense to study what the robotics domain has to offer and to adapt its results to the automotive domain.

This thesis sits in between the domains of automotive E/E architecture and cognitive, autonomous and intelligent robotics. It attempts to adapt the principles and architectures of intelligent autonomy, as developed in robotics, to the automotive domain. In particular, the research direction attempts to introduce a cognitive component in existing automotive architectures. Such a cognitive component would acknowledge the presence of other existing subsystems in the architecture and utilize them to generate the desired system behavior. Progressive increase in the capabilities of the cognitive component would be one way to achieve progressive vehicle autonomy. Our approach would also involve a refactoring of existing automotive architecture such that it better works with the cognitive component. In this thesis work, the foundation for such an approach has been laid.

Chapter 3

Contributions

"Talk is cheap. Show me the code."

Linus Torvalds

The primary contribution of this thesis (publications A, B and C) is a reference architecture for autonomous, cooperative driving. A secondary contribution (publication D) is a pattern for thinking about autonomous systems architecture, which also presents a fresh perspective on complexity related problems of existing automotive architectures. This pattern was inspired by the work done on the reference architecture.

3.1 A reference architecture for cooperative driving

Can a vehicle drive by itself in a scenario where vehicles and infrastructure in the vicinity are continuously broadcasting information about themselves? Can such a feature be added to an existing vehicle architecture? Can it be added in a way that requires no significant changes to the existing vehicle subsystems? Moreover, is there a general pattern or template for doing it, which can be applied to diverse practical cases? The answer to all these questions is, "Yes" and the primary contribution of this thesis is to describe exactly how all this can be achieved. In doing so, it answers the research question posed in section 1.4, *What is a good way to introduce autonomy in a vehicle for the purpose of cooperative driving?* This section describes the artifacts resulting from the work. A discussion of how these artifacts answer the research question and validate our hypothesis is presented later in section 4.1.

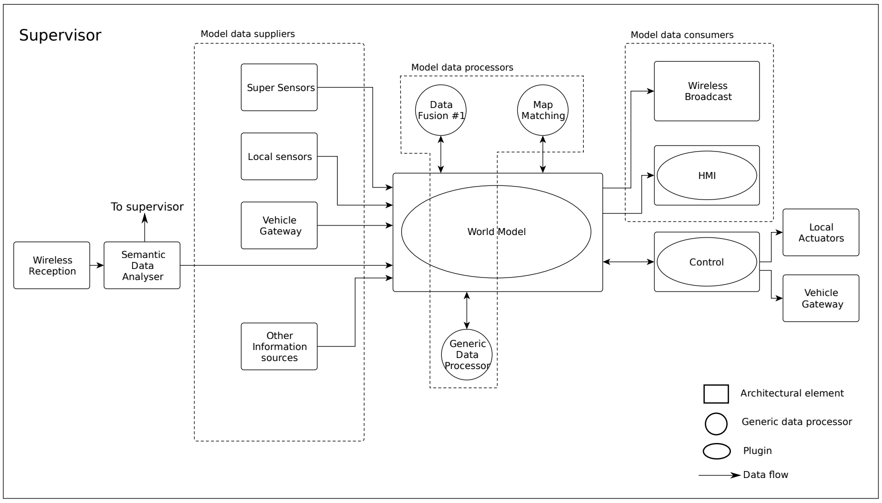


Figure 3.1: A reference architecture for cooperative driving

Publication A presents a reference architecture (Figure 3.1) for cooperative driving. The reference architecture describes a (sub)system which can be "plugged into" an existing vehicle architecture. When activated, the (sub)system takes over from the human driver and drives the vehicle. As shown in Figure 3.1, the architecture centers around a 'World Model', which is a database of all current (and possibly some past) information in the system. This information could be sensor data, fused sensor information, state information, control setpoints etc. There are 'Model data suppliers', which are entities that supply information to the World Model and 'Model data consumers' which consume tha data in the World model. Other architectural elements involve a Supervisor and a means to Control the vehicle motion.

In publication A, we first investigate the technical considerations involved in the design of such a system. Next, the services that are required within the vehicle to achieve cooperative driving functionality are identified. These services include positioning, world modeling, wireless communication, supervision and some others. Based on the technical considerations and services, we then define an architecture that can provide the required services. The key architectural elements are described, together with their inter-relationships. Guidelines are provided for the instantiation of the reference architecture. The paper also describes a validation of the proposed reference architecture via a specific instantiation and describe the experiences with the instantiation. This particular instantiation was validated during the Grand Cooperative Driving Challenge (GCDC) 2011 at Helmond, the Netherlands. The GCDC consisted of vehicle platooning on public roads. Platooning is a specific scenario in cooperative driving that involves vehicles driving autonomously, one behind another, as though in a road train. Finally, the reference architecture is compared to state of the art architectures for autonomous systems, and also to the AUTOSAR standard. It is concluded that the architecture agrees well with established principles of autonomous systems architecture and that it can be implemented with AUTOSAR concepts and infrastructure.

Publication B describes in greater detail the specific instantiation of the reference architecture that was used during the GCDC 2011. This publication covers some of the communication and control algorithm aspects that are outside the scope of the reference architecture, but which were needed for that specific instantiation. Finally, publication C describes all the technical implementation details of the instantiated architecture, which are missing from publication B.

3.2 An approach to embedded systems autonomy

How should one think about autonomy in the context of existing general embedded systems architectures? What are the main challenges being faced by these architectures in their migration towards autonomy? How can existing embedded systems be moved closer to autonomy, without disruptive changes to their existing designs? Is there a mental framework/pattern that can be used while thinking about the design of autonomous embedded systems?

These are rather big questions. Publication D is a first, exploratory step towards answering them. The ideas in this report have been partially triggered by the work with the reference architecture.

The paper postulates that all intelligent autonomous systems, regardless of their level of autonomy, can be described by a finitely recursive pattern consisting of four components: User, Environment, Control and the Self (Figure 3.2). The User is the entity that tells the autonomous system what it must do, and a User model enables the autonomous system to interpret and understand the wishes of the User. The Environment is a representation of the external and internal world that the autonomous system operates in. The Control is any output being controlled by the system. The Self is the accumulation of intelligence that is ultimately responsible for the behavior of the system. It is the entity that is aware of the different system components and how they

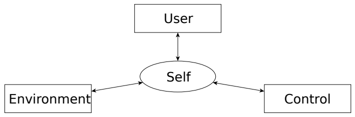


Figure 3.2: Components of an intelligent, autonomous system

must interact in order to form the desired system behavior. The Self is the "consciousness" within the system, if such a poetic analogy is allowed within the realm of engineering.

In most of the current system designs, this Self is implicit. It is present as a result of the overall system construction. This means that there is no specific component or subsystem that is responsible for the behavior of the entire system, and which directs and monitors the functioning of all the other subsystems. We claim that it is necessary to have a single[1], explicit Self in a system, which is responsible for the ultimate behavior of the whole system. We argue that as systems and subsystems grow towards autonomy, multiple implicit selves appear and the conflicts among them are and will continue to be among the foremost challenges to the integration of autonomous subsystems a.k.a the architecture of autonomous embedded systems. We hypothesize that awareness of the four distinct components that make up an intelligent autonomous system will be a valuable aid during the design of such systems. We also hypothesize that the introduction of explicit and coordinated Selves is the way to non-disruptively migrate existing embedded systems designs to autonomy.

Two other problems identified in publication D are state space explosion and the problem of partitioning the architecture into relatively independent sections to manage complexity. These problems are addressed by the automotive industry every few years i.e. solutions are found which fix current and anticipated needs; however, as the system complexity grows, the same problems crop up again. Therefore, we argue that these problems will need to be solved using fundamentally different principles that prevent them from recurring. It is postulated that systems design based on the principles of autonomy will go a substantial way towards a sustainable solution to these problems.

It is hoped that some of the concepts from publication D will be a foundation for further PhD research on the topic of embedded system autonomy. A discussion of using the Self for migration of existing systems towards au-

[1] *conceptually* single. It may be possible to *implement* the self in a distributed way.

tonomy is presented in section 4.2, while the work that needs to be done in order to concretely develop this approach further is presented in section 5.1.

Chapter 4

Discussion

"..all I wanted was compliance with my wishes, after reasonable discussion."

Winston Churchill

The discussion is split into two chronological parts: the past and the future. A backward glance at the work done is presented in section 4.1, while section 4.2 looks towards the future and discusses a specific approach to designing autonomous embedded systems.

4.1 Reflection on work done

In section 1.4, a hypothesis was posed that a working answer to the question, "What is a good way to introduce autonomy in a vehicle for the purpose of cooperative driving?" could be provided in the form of a reference architecture. So is the hypothesis valid and has the reference architecture presented in 3.1 provided an answer to the question? To answer this question, it must be considered that the reference architecture was instantiated and tested in two scenarios. These were the GCDC 2011 and the CoAct-2012 Cooperative Driving Demonstration held in Göteborg in November 2012. The two instantiations differ in a number of aspects. For starters, they share very little code with each other. Secondly, there are differences in the number of software components that implement the various reference architecture elements. The distribution of these components across computing nodes is also different. Finally, the CoAct-2012 instantiation has support for overtaking and lane changing maneuvers, which is distinct new functionality that was missing from the GCDC 2011 instantiation. Both the instantiations performed satisfactorily. By 'satisfactory performance' we mean that at no point did the architecture impose unreasonable constraints on the system behavior that was required in the scenario. The architecture remained in the background and allowed the developers to focus on algorithms. It was also possible to adapt the system behavior to last minute change requests, for example, the creation of a 'radio silence' mode during the GCDC 2011. We also believe that the reference architecture is valid outside the scenarios. This is because while creating the reference architecture we carefully considered the requirements, needed services and design patterns for a general purpose cooperative driving system i.e. one which is not scenario specific. Therefore, the reference architecture has a broader scope than the platooning scenarios for which it was instantiated. While it is possible that the two differing instantiations succeeded purely by chance, it is more probable that the careful consideration that went into the design of the reference architecture paid off. Nevertheless, it is possible that some edge cases and scenarios are not captured by the reference architecture. However, based on our experiences with the two different instantiations and and the careful thinking that went into the reference architecture design, we can say with reasonable certainty that the hypothesis has been validated. The reference architecture has explicitly provided some requirements, principles and patterns for architecting autonomous embedded systems. Therefore, it provides partial answers to the first two of the overall problems mentioned at the start of section 1.4.

How well did the chosen research methodology (based around designing engineering prototypes) work? The research intended to generate situated knowledge for addressing the particular need of autonomous, cooperative driving. The chosen research method resulted in the generation of such knowledge, as well as the validation of that knowledge. Therefore, if we look at research methodology purely as the means to an end, then the methodology chosen for this thesis worked very well indeed. But there are at least two more reasons for why the methodology worked well. Firstly, the methodology was conducive to simultaneous development and design activities. The development of a prototype in parallel with the design enabled rapid examination of the in-progress design. Therefore, it was possible to catch some design flaws early, rectify them, and validate the rectifications even as the design details were still being finalized. This lead to an incrementally modified and tested design which lead to a state of continuous confidence in the "design-so-far". Contrast this with a development approach where a design is theoretically worked out to the last detail and only then is the practical validation initiated. The second reason for why the chosen research methodology worked well is non-technical. It has to do with the motivation needed by the reseacher working on the problem. There are those who prefer to work at a somewhat abstract, theoretical level without caring very much about specific implementations; and there are others who prefer a more hands-on approach and absolutely need a physical system to tinker with, in order to stay motivated. The author of this thesis falls into the latter camp and therefore considers as ideal, a research methodology that encourages prototype building. Without a real truck to program and drive around in, this work might never have happened.

Would it have been better to make a number of smaller case studies, instead of the single, large one that was made? After all, it takes a formidable amount of time to mess with the tiny practical details that really have nothing at all to do with the theory being developed. It is also not always convenient to sit and program in a truck and drive it around at odd hours. We think it is necessary to go all the way to implementation with at least one case. This is because doing so often yields valuable insights that may not be obtained from partial implementations or simulations. Experience with implementation technologies is helpful for future design work and sometimes the actual implementation may be of importance to someone you are working with. For example, the GCDC 2011 implementation uncovered several issues with implementations of the 802.11p wireless protocol, which were fixed along the way. Also, in section 1.4, we stated that we would consider industrial embedded systems. Therefore, we needed to take into account matters related to product legacy and development methodologies (vendors, consultants, industrial practices etc.). By working on a large system, issues related to these aspects manifest themselves more easily. Given the success of one large case study, it may now be acceptable to perform a number of smaller studies for further analysis and improvements to the system.

4.2 An approach to incremental system autonomy

We now return to the last of the overall problems posed in section 1.4. *What are the engineering and technical ways to evolve existing embedded systems architectures towards those that are expressly designed for autonomy, while minimizing impact on established, legacy designs?* This section briefly describes one approach that appears promising and could be a topic for further research. Note also that we now widen the scope from automotive E/E architectures to general embedded system architectures, because the approach is not automotive specific and is applicable to any embedded system that is composed of subsystems. A rudimentary implementation of this approach has already been attempted in the reference architecture (see section 3.3.6 of publication A), although at the time of that implementation, an overall understanding of the underlying theory had not been developed.

The proposed approach is inspired by the field of *Artificial Consciousness (AC)*[32, 58]. AC is devoted to the creation of *consciousness* in engineering artifacts, such as a digital computers. Consciousness is defined as *the quality or state of being aware, especially of something within oneself*. Applying this definition to the architecture of embedded systems, it is possible to imagine a system component, or subsystem, that reifies[1] a consciousness within the system. In this thesis, we denote such a component, or subsystem, with the term *Self*[2]. The Self can contain knowledge about the other subsystems present in the system and how they should act and interact to fulfill the system's behavioral goals. It could also monitor and direct the subsystems and their interactions and thus maintain a system state with operational information regarding the system goals being achieved. Such knowledge is a simplified form of *awareness* within the system and therefore, a means for system consciousness.

The Self could be non-disruptively introduced into an existing embedded system and its capabilities can be gradually enhanced. Initially, there would be no requirement for heavy modifications to the other subsystems and the Self itself would have extremely limited capabilities, perhaps simple monitoring and reporting. But gradually, more capabilities could be added to the Self, so that it becomes active in decision making and directing the functioning of the other subsystems. As the architecture evolves, the design of the other subsystems could be modified in such a way that they can better interact with the Self. The ultimate, if rather abstract, goal would be to take the system level knowledge embedded in the head of the system designer(s) and system users and build it right into the Self, so that the system itself can make decisions similar to those that the designers or users would make. The designer's knowledge would be used to understand and direct the detailed subsystem behavior, while the user's knowledge would be used for high level operation of the system. As the capabilities of the Self rise, the machine will be able to perform its tasks with less and less human intervention and this, by definition, is progressive autonomy.

[1]Reify (verb) /ˈrēəˌfī/: Make (something abstract) more concrete or real.

[2]The discussion of the Self in this section is directly triggered by, and builds upon, the concepts introduced in appended paper D and briefly described in section 3.2.

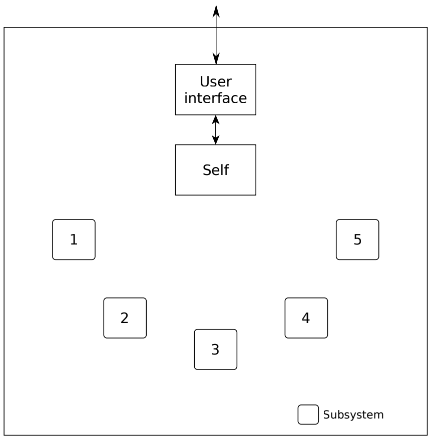


Figure 4.1: Two elements of autonomy

Consider, for the sake of argument, a system as shown in Figure 4.1. It shows a system made up of five subsystems, with two additional entities: A User Interface and the Self. The User Interface is, from the system's perspective, the *single* channel via which the demands of the system's user are obtained. Every interaction that the user has with the machine and its operational controls is captured by the User Interface and this is then passed on to the Self. None of the subsystems directly see the user demands. The Self interprets the user demands and directs the subsystems' operations in order to meet the demands. The Self may also choose to ignore a User demand entirely, if it deems such an action appropriate. The Self may also monitor the subsystem actions and provide appropriate feedback to the user, via the User Interface.

What advantages would such a scheme provide over existing architectures? To begin with, each subsystem would not necessarily have to know about the overall system state before deciding whether to perform its commanded action. Having the Self interpret and filter the user's demands prevents a scenario where a subsystem directly responds to the user demands and causes a system level problem, whilst fulfilling its own local goals. A subsystem need not have to constantly monitor a global system state before performing its actions, because it may assume that if the action request has come from the Self, then it is safe to perform that action. This could lead to simplification of subsystem local logic and decision making. The Self becomes the natural and appropriate place for proactive control and resolution of conflicts among the subsystems with regards to resources, intended actions and unintended consequences. A quick example illustrates this point. There is an anecdotal case involving a truck parked uphill, with its engine and air conditioner running. As the Sun shines in through the window, the cabin internal temperature rises, causing the air conditioning (AC) ECU to do more cooling. In order to fulfill its task, the AC ECU requests more power from the engine. The engine ECU revs up the engine in order to generate more power. The parking brake ECU notices the engine revving up, decides that the driver is attempting to drive away and releases the parking brake. Such unintended interaction among features is referred to as *feature interaction.* In a system with a User-Self, like the one in Figure 4.1, the Self would ideally know that its current prime directive is *Do Not Move* and would not allow the parking brake to be released, and even if the parking brake ECU 'went rogue' the Self would involve the braking system to satisfy its prime directive. Of course, a bad design/implementation of the Self could prevent this from happening, but the point here is that in the current architecture, with no Self, there is no higher level logic in the system that 'knows' that the system is currently required to be motionless. *Introducing a Self merely provides a convenient mechanism to aggregate and direct system level behavior.* For long term autonomy, the Self is the place for connecting the planning and prioritizing mechanisms needed for generation of appropriate system behavior. In summary, this is a feasible engineering mechanism to give the machine a mind of its own.

It may be argued that the Self becomes a single point of failure for the entire system. This is true, to a certain extent. Analogously, even biological organisms (humans) have their brains as a single point of failure, but this design works well enough that it has not been discarded during natural selection yet. In the case of machines, the problem can be alleviated via specific implementation methods. For example, having replicated Selves, with one active and the others in 'hot standby' mode can lower the risk associated with the catastrophic failure of one Self, for safety critical systems. Alternatively, the implementation of the Self could be distributed, so that partial functionality can still be provided even if some components fail.

Note that the concept of the Self is not just the old concept of 'centralized' (versus distributed) control stated in a different way. While we argue that the Self should be a conceptually unified entity, we do not make any statements regarding the physical implementation (centralized vs distributed) of the Self. It should be possible to implement a conceptually single Self in a physically distributed manner. Also, even if the Self is implemented as a single physical entity, it is the system level intelligence that is being centralized, leaving open the possibility of distributing specific control loops.

A potential problem for autonomous systems incorporating a Self (and even for complex systems in general) is a *lack of determinism.* This means that due to the sheer complexity of a system, or by virtue of its autonomy, it may be impossible to correctly predict the system behavior under all possible circumstances. The user may not even be in absolute control of the machine, as his/her demands may simply be disregarded by the Self. Such scenarios are unacceptable to the designers of current embedded systems. However, increased autonomy may force the acceptance criteria to shift from deterministically correct behavior to *probabilistically correct* behavior. Probabilistic correctness in this context would imply that the probability of incorrect behavior approaches zero when the design practices, architectures and runtime checks and guards are followed.

Chapter 5

Future work and Conclusion

A conclusion is the place where
you got tired of thinking.

Martin H. Fischer

5.1 Future work

The approach and discussion from section 4.2 will serve as the basis of our future work in embedded systems autonomy. The future work shall then consist of a more formal and detailed development of the artificial consciousness approach. This would be followed by an implementation, which would then be validated. Ideally, the approach should also be tested in domains other than automotive.

Section 4.2 merely presented an outline of the approach. Digging deeper, at least the following questions need to be answered before the approach can be considered as fully developed:

1. What is a good way to encode knowledge about a system's construction and operation into the Self? We have stated that the Self needs to know about the presence and capabilities of the different subsystems and how the subsystems should interact to fulfil the system's current behavioral goal. Therefore, the Self needs to constantly represent, refresh and reason on its own knowledge of the system and this question is related to the ways and means of doing so.

2. How should the user's demands be internally represented and communicated with the Self? The user's intentions need to be translated into formal semantics in order to communicate them to the Self. The user interface of a vehicle will probably retain the familiar controls which are present in existing vehicles, but the interpretations of the control inputs obtained from the user may have to be changed or represented in specific ways. For example, depression of the brake pedal could be understood and represented as an intention of 'slow down', rather than being directly interpreted as a proportion of braking torque to be applied at the brake disc of each wheel.
3. How can subsystems be designed so that they can better interact with the Self? What forms should such interactions take?
4. What are the patterns of monitoring and control that the Self can exert on the subsystems?
5. What reference architectures can be generated around these concepts?
6. How can such architectures be verified?

The future work will commence by finding answers to the above questions.

5.2 Conclusion

This licentiate thesis effort started with broad questions about architecting autonomous embedded systems. The scope was narrowed down to architectures for cooperative autonomous driving and the research question was formulated (section 1.4) as "**What is a good way to introduce autonomy in a vehicle for the purpose of cooperative driving?**" We also put forth the hypothesis that a working answer to this question could be provided in the form of a reference architecture for cooperative driving.

During the course of this thesis work, a reference architecture for cooperative driving was created. The reference architecture was instantiated on two separate occasions, and on both the occasions, the respective instantiations enabled achievement of vehicle autonomy in a satisfactory manner. Based on these instantiations, and a rational discourse on the design of the reference architecture, we feel that our hypothesis has been validated.

The specific contributions of this work are

1. Two functioning prototype vehicles capable of autonomous motion in cooperative driving scenarios

2. A validated reference architecture for cooperative driving
3. A proposed approach for introducing progressive autonomy in embedded systems architectures.

The approach mentioned in point 3 above requires a more thorough evaluation. It was spawned by the work done on creating and instantiating the reference architecture for cooperative driving. The approach involves introducing an artificial consciousness within the system, whose capabilities can be increased over successive design iterations. Future work will consist of more detailed investigations and development of this artificial consciousness approach.

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] YARP: Yet Another Robot platform. URL http://eris.liralab.it/yarp/.

[10] ØMQ - Multithreading Magic. URL http://www.zeromq.org/whitepapers:multithreading-magic.

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

[12] IEEE Standard Glossary of Software Engineering Terminology, 1990. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=159342.

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

[14] 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.

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

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

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

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

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

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

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

[29] 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.

[30] 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.

[31] 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.

[32] Giorgio Buttazzo. Artificial consciousness: Utopia or real possibility? *Computer*, 34(7):24–30, July 2001. ISSN 00189162. doi: 10.1109/2.933500. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=933500.

[33] 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.

[34] 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.

[35] Andrew Chong. *Driving Asia - As Automotive Electronics Transforms a Region*. Infineon Technologies Asia Pacific Pte Ltd, 2010. URL http://www.infineon.com/cms/cn/DrivingAsia.html.

[36] 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.

[37] 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.

[38] 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.

[39] 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+%3F&ots=n15Trrs_KI&sig=rcL8hcj_ZN5k-84oBZiCvQXP_wE.

[40] 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.

[41] 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/.

[42] TLJ Ferris. On the methods of research for systems engineering. *Annual Conference on Systems Engineering Research*, 2009(April), 2009. URL http://cser.lboro.ac.uk/papers/S10-62.pdf.

[43] TLJ Ferris. Engineering Design as Research. In Manuel Mora, Ovsei Gelman, Annette L. Steenkamp, and Mahesh Raisinghani, editors, *Research Methodologies, Innovations and Philosophies in Software Systems Engineering and Information Systems*. IGI Global, February 2012. ISBN 9781466601796. doi: 10.4018/978-1-4666-0179-6. URL http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-4666-0179-6.

[44] 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.

[45] 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.

[46] 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.

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

[48] 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.

[49] 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.

[50] 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.

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

[52] 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.

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

[54] 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.

[55] 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.

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

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

[58] Douglas R. Hofstadter and Daniel C. Dennett. *The Mind's I*. Bantam Books, 1982. ISBN 0-553-34584-2.

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

[60] 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.

[61] International Organization For Standardization. ISO/IEC 42010:2007 Systems and software engineering - Recommended practice for architectural description of software-intensive systems, 2007. URL http://www.iso-architecture.org/.

[62] 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.

[63] Alma L. Juarez Dominguez. Feature Interaction Detection in the Automotive Domain. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pages 521–524. IEEE, September 2008. ISBN 978-1-4244-2187-9. doi: 10.1109/ASE.2008.97. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4639390.

[64] 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.

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

[66] 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.

[67] 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.

[68] Philippe Kruchten. *The Rational Unified Process*. Rational Software White Paper. Addison-Wesley, 2003. ISBN 0321197704.

[69] Thomas S Kuhn. *The Structure of Scientific Revolutions*, volume II of *SO Source: University of Chicago Press: Chicago. (1962)*. University of Chicago Press, 1970. ISBN 0226458032. doi: 10.1119/1.1969660. URL http://www.jstor.org/stable/10.2307/2183664.

[70] 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.

[71] 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.

[72] 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.

[73] 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.

[74] John McCarthy. What is Artificial Intelligence, 2007. URL http://www-formal.stanford.edu/jmc/whatisai/whatisai.html.

[75] 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.

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

[77] 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.

[78] Andreas Metzger. Feature interactions in embedded control systems. *Computer Networks*, 45(5):625–644, August 2004. ISSN 13891286. doi: 10.1016/j.comnet.2004.03.002. URL http://linkinghub.elsevier.com/retrieve/pii/S138912860400043X.

[79] 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.

[80] 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.

[81] 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.

[82] 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.

[83] 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.

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

[85] 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.

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

[87] 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.

[88] 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.

[89] 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.

[90] 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.

[91] 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.

[92] 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.

[93] 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.

[94] 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.

[95] 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.

[96] 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.

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

[98] 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.

[99] 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.

[100] 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.

[101] 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.

[102] 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.

[103] 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.

[104] 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.

[105] William M. Trochim. The Research Methods Knowledge Base, 2nd Edition, 2006. URL http://www.socialresearchmethods.net/kb/.

[106] 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.

[107] 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.

[108] 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.

[109] 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.

[110] 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.

[111] 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.

[112] 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.