An architecture pattern for embedded systems autonomy

Sagar Behere[1], Martin Törngren[1], Jad El-Khoury and DeJiu Chen[1]
[1]*Department of Machine Design, Kungliga Tekniska Högskolan, Brinellvägen 83, Stockholm, Sweden*
{*behere,martint,jad,chendj*}*@kth.se*

Keywords: Autonomy, Embedded Systems, Architecture, Multi-level Hierarchical Systems

Abstract: Embedded systems are being increasingly utilized to enable autonomous behavior in machines. Embedded system architectures must therefore keep pace with the autonomy requirements. In this position paper, we outline an architectural approach to complement the most commonly utilized principles for architecting embedded systems, like decentralization, separation of concerns and loose coupling between subsystems. The approach provides an explicit pattern for facilitating safe and predictable system level autonomy. It involves the addition of a single subsystem at the apex of the functional hierarchy in the system. The minimum objectives that need to be fulfilled by such a subsystem are also presented.

1 INTRODUCTION

Modern machines increasingly incorporate embedded systems as the backbone of their functionality. The embedded systems enable advanced functionality in the machine, more precise and accurate control as well as the development of features which would be extremely difficult to realize in any other way. One such feature is machine autonomy. Autonomy may be defined as: *The ability to operate without human intervention*. Autonomy is a discernible trend in the evolution of many types of machines. For example, in a modern car, an increasing number of functions like changing gears, maintaining speed or distance to the vehicle ahead, holding lanes et cetera now require no human intervention. Similarly in aviation, the evolving functionality of autopilots requires decreasing human intervention in the operational phases of a flight. Embedded systems enable autonomy because they make it easy to perform computations and take decisions based on those computations.

Research on autonomous systems is conducted in broadly two areas: algorithms and architecture. Algorithms typically deal with sensor data perception, planning, reasoning etc. Architecture typically deals with the hardware, software, communication and execution structures. This paper focuses exclusively on the architecture aspects of autonomy, with particular regard to the following two questions:

1. What should the architecture of the embedded systems incorporated a machine look like, if the machine should exhibit autonomy?
2. How should the existing embedded systems architecture of a machine be evolved towards autonomy?

Question 2 is of particular interest to designers and developers of existing commercial products due to reasons of legacy, cost and prior investment in technologies and solutions. To take an example from the automotive domain, the Swedish project "FUSE" (which partially funds this work) has an explict goal to *"..define a migration path from existing vehicle architectures towards ... reference architecture for autonomy."*. In another case, the authors have been approached by a major European vehicle manufacturer for conducting research on how its vehicle architectures should be evolved, such that it is possible to undertake safe and standardized integration of third-party (OEM) autonomy solutions into the vehicle platform.

A general observation is that the integration of competitively developed subsystems has been the way to reduce costs in domains such as automotive and aerospace. This practice is spreading to other domains as well. The integration is aided by architectural principles like system partitioning, good separation of concerns between subsystems and distributed and decentralized architectures. The exposition of these principles, their methodologies and best practices have received wide coverage in the literature (Törngren and Wikander, 1978; Kleinrock, 1985; Schoeffler, 1984; Fielding, 2000; Watkins and Walter, 2007; Eloranta and Hartikainen, ; Guth and D'Epinay, 1983). However, are these principles sufficient or

even adequate to meet the trend of increasing system level autonomy? In this paper, we argue (in section 2) that the typical decentralized, distributed embedded system architecture is insufficient for the development of predictable and safe system level autonomy. We then present a practical approach to complement the existing distributed system design such that the resulting architecture facilitates system autonomy. Specifically, the existing architecture of the machine is treated as a multi-level hierarchical system which is complemented by the introduction of a single, top level subsystem at the apex of the hierarchy. The complementing subsystem needs to fulfill some minimal objectives, which are also described (section 3). We then present a brief discussion of the approach along with preliminary evidence of its applicability (section 4) and our conclusion (section 5).

2 APPROACH

2.1 Research Context

In this research, we constrain ourselves to those machines that reach towards autonomy by means of the embedded system incorporated within them. This means that we look at the embedded system as the sole enabler of machine autonomy. For the purposes of this text, the term 'embedded system' refers to the sum total of all the electronic computing hardware, software and communications incorporated into the machine. The embedded system is assumed to be divided into communicating subsystems. The division may be physical, in the sense of multiple physical computing units or logical in the sense of sofware components. The different functions performed by the embedded system as a whole may be arranged in a hierarchy, or as a sequence, or functions may be composed out of multiple sub-functions. Related functions are usually grouped within the same subsystem. In practice, these constraints describe a very wide range of embedded system designs. For example, in the automotive domain, the embedded system of a vehicle consists of multiple Electronic Control Units (ECUs) all of which are attached to a common communication bus (see Figure 1), via which they may interact with each other. Vehicle features like 'Traction Control' and 'Cruise Control' may be logically arranged in a hierarchy as shown in Figure 2.

Such embedded systems are often completely 'distributed'. The system is partitioned into subsystems in a manner which minimizes communication between the subsystems. Also, a subsystem does not always need to be aware of the role and functioning

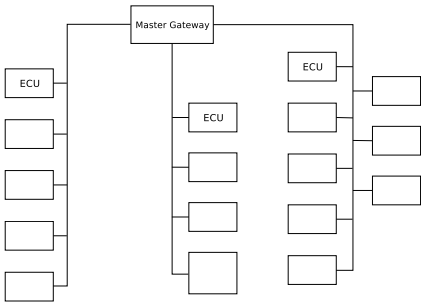

Figure 1: Typical layout of ECUs on a vehicle bus network

of other subsystems, even though it may be exchanging signals and information with them. Such a design paradigm is termed 'separation of concerns' in the literature and is one of the established methods of managing complexity(Kopetz, 2008). The subsystems in such a design may also be 'loosely coupled'. A loosely coupled subsystem can send data without knowing who the receivers are and it may receive data without knowing who the sender of that data is. System partitioning and loose coupling not only simplify the design and implementation of the system, they also reduce the cognitive effort needed to comprehend and reason about the system behavior(Kopetz, 2008) while making it easier to implement desirable characteristics like fault isolation and (where required,) subsystem redundancy. However, we argue (in the forthcoming section 2.2) that it can be rather difficult on even outright impossible to impose predictable and safe system level autonomy on an embedded system which is comprised solely of a distributed, loosely coupled aggregate of subsystems. Assuming for the moment that this argument is valid, how then can the proven advantages of a loosely coupled, distributed system be retained, while simultaneously enabling intelligent system level autonomy? We propose that system level autonomy may be conveniently achieved by the introduction of a single subsystem at the very top of the functional hierarchy.

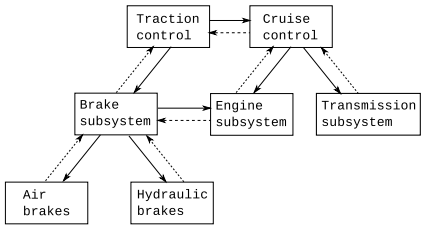

Figure 2: Hierarchy of traction and cruise control features in a vehicle

2.2 Autonomy by means of a supremal subsystem

We claim that to construct an embedded system which demonstrates deliberate(i.e. not accidental), predictable and intelligent system level autonomy, there needs to be an entity within the system that is characterized by at least the following three abilities

1. The ability to gather relevant information from within the system
2. The ability to reason about the gathered information in relation to the overall goal(s) of the system. This presupposes a knowledge of the overall purpose the system is constructed for
3. The ability to examine (or at least, query) the capabilites of individual subsystems and to influence their working

A principal characteristic of an embedded system that is comprised solely of a distributed, loosely coupled aggregate of subsystems is that each subsystem concerns itself with its own narrow area of influence. There is no subsystem that has an overview of the purpose and status of the entire system as a whole. Furthermore, subsystems are not encouraged to be aware of the functioning and capabilites of other subsystems (at least those subsystems which are at the same level in the hierarchy) and usually have a limited influence on their operation. Consequently, the widely practiced and useful technique of system partitioning *by itself* runs counter to the requirement of global, system level abilities mentioned above. This indicates that there needs to be some kind of system structure or principle *in addition to* partitioning that can permit the development of intelligent system level autonomy.

At this point, it is important to address an apparently reasonable objection that by enabling a systematic, 'high-bandwidth' communication matrix between the subsystems, they could be made to communicate and cooperate in a way that generates system level autonomy as an *emergent* phenomenon. Indeed, there have been studies in the fields of organization theory, biological social systems and applications to robotics and artificial intelligence agents [see for example (Forrest, 1990; Crutchfield and Mitchell, 1995)] which suggest that the emergence of system autonomy is a plausible phenomenon and can even be directed using a careful selection of rules and heurestics. However, whichever way such designs are considered, they suffer from drawbacks which negatively affect their selection as an architecture of choice for safe and predictable autonomous systems

1. System architectures where the communication matrix is carefully determined and controlled and within which voluminous data exchange takes place among the subsystems, are complex to understand and analyze. This inevitably leads to subtle errors errors which are difficult (and costly) to anticipate, trace and eliminate. The massive state space of such systems leads to challenges in behavior verification and achieving sufficient test coverage. Furthermore, this approach does not scale easily when more and more subsystems are added to the system.
2. Systems where the communication between the subsystems is not as well regulated, and where the subsystems tend to make their own decisions based on heuristics and the current state, tend to be sub-optimal and unpredictable in some states of their operation. This unpredictability constitutes an unacceptable hazard for industrial, commerical systems. Few people would get into an airplane wherein the embedded computers interact in ways that the designers may have not foreseen, even though the chances of everything eventually working out are high because the system has been designed that way! Architects of embedded systems generally prefer to have behaviors which are highly deterministic and demonstrably hazard free. Indeed, for safety critical and certifiable systems this is an absolute requirement.

What is needed then, is a practical architecture for intelligent, safe and predictable autonomy that combines well-designed, relatively isolated, distributed subsystems with an entity that is capable of system level reasoning and action.

So we return to the aforementioned point in this subsection: What kind of system structure or principle *in addition to* partitioning is needed in order to permit the development of intelligent system level autonomy? Consider again the hierarchy shown in Figure 2. The hierarchy contains a distinguishing characteristic which makes it a representative of the very large class of distributed embedded systems with which we are concerned in this paper. The distinguishing characteristic of this hierarchy is the existence of a family of subsystems, each of which has its own individual goal. The individual goals are not necessarily conflicting; indeed during normal operation the subsystems may collaborate to generate some predefined system behavior. However, *in case of a conflict, none of the subsystems has the capabilities to resolve the conflict*. The subsystems utterly lack the three abilities introduced at the start of this subsection. Our approach is to introduce a single 'supremal subsystem' as shown in Figure 3 which incorporates an entity that reifies those three abilities. The term 'supremal' is borrowed from (Mesarovic et al.,

1970), which refers to, *"..higher level subsystems as supremal units while the subsystems on the lower levels are termed infimal units."* The addition of a *single*[1] supremal unit as the apex of a hierarchy effectively makes the system into a "multi-echelon type system" as described by (Mesarovic et al., 1970), who argue that, *"The existence of a supremal unit is the principal charactersitic of multi-echelon systems."*. The formal, mathematical analysis of multi-echelon systems can then be applied to this architecture. In particular, we believe that the following three items are of interest to the architecture of autonomous systems

1. The assignment of tasks and roles to the various hierarchical levels and their individual subsystems.
2. The definition of interfaces within the hierarchy.
3. The rules according to which the supremal units can intervene and coordinate the functioning of the infimal units.

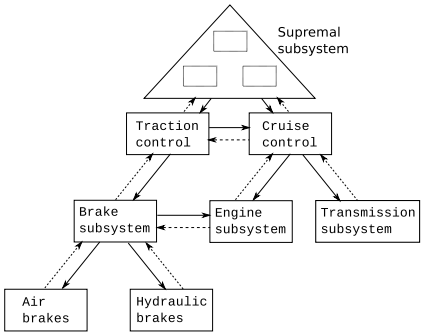

Figure 3: Adding a single supremal unit to the hierarchy

While specific answers to the above will always be application and implementation specific, we believe that it is possible to develop a generalized set of principles at a level of abstraction that can be applied to the creation of *reference architectures* for particular autonomous systems.

To recapitulate the ideas in this section

- Distributed embedded systems constructed on the prevalent principles of separation of concerns and system partitioning have insufficient structure to develop predictable and safe autonomy
- There needs to be a *single* entity within the system boundary which is capable of gathering, reasoning and acting on system level concerns.
- The approach of adding of a single supremal subsystem to top off an existing hierarchy retains the advantages of an existing distributed architetcture while providing a formally tractable engineering solution to the problem of evolving that architcture towards autonomy.

However, merely stating, "Stick a subsystem on top" provides little value. In the next section, we provide some details regarding the minimum objectives of the supremal subsystem vis-à-vis the desire for system level autonomy.

3 MINIMUM OBJECTIVES OF THE SUPREMAL SUBSYSTEM

The motivation for inclusion of a supremal subsystem in the architecture includes the idea that the subsystem knows about the purpose of the system as a whole and the possible desires of the user with respect to the system's behavior. The subsystem would also be aware of the other subsystems present in the system, their roles and capabilities and it would have the knowledge to orchestrate the functioning of those subsystems with the ultimate intention of generating the desired system level behavior. With this in mind, we propose that the supremal subsystem must contain structures to satisfy at least the following objectives:

Interaction with the user The user's commands need to be translated into system behavior. Most of the commands need to be filtered through the supremal subsystem, which can translate them into inputs suitable for infimal subsystems. In existing distributed systems, the user commands are often communicated directly to a specific subsystem, whereupon the subsystem may react without considering the state of other subsystems or how they might be affected (Consider a car: The user has pressed the accelerator pedal. The pedal position is mapped to fuel quantity injected into the engine. More the pedal is pressed, more the fuel sent to the engine. But what if the parking brake was on? Should the engine check on the brake? Let's say it does.. but what if the user is going uphill with a heavy load and has engaged the parking brake to prevent a backward slide while stopped?). In order to correctly interact with the user, the supremal subsystem needs a model of the user interactions and the semantics of possible inputs vis-à-vis system behavior.

Understanding subsystem capabilities The supremal subsystem must be able to query the infimal subsystems with an intent to understand their capabilities and the functionality with which they

[1] If, instead of a single unit, there are multiple supremal units at the top of the hierarchy, the question of conflict resolution between them reappears and the entire argument is reduced once again to the old one.

contribute to the overall system behavior. Perhaps there are multiple subsystems with the same roles but which perform them with differing Quality of Service(QoS). Perhaps the functionality of multiple subsystems can be efficiently combined to yield a greater QoS. Such knowledge is required when orchestrating the infimal subsystems to achieve a user specified system behavior.

Translating user commands to subsystem inputs Depending on the user interaction model, the user inputs may enter the system in a variety of forms. This could range from voice inputs and facial recognition techniques to the states of levers, pedals, buttons or switches being manipulated by the user. The supremal subsystem should have the capabilities of interpreting the various inputs with regard to the system behavior. For example, if the user presses on the accelerator pedal of the car, the supremal subsystem may interpret that input as the desire to increase the velocity of longitudinal motion. This interpretation is the basis of subsequent interventions and coordinations of the infimal subsystems, which may involve substantial and recurring planning and sequencing operations.

Active monitoring and control of infimal subsystems This is related to regular operation of the system. It deals with the lifecycle management of the infimal subsystems, diagnosing error states and assessing their impact on system capabilities handling asynchronous requests from the infimal subsystems, generating short term goals for and coordinating the execution of infimal subsystems and resolving conflicts.

Minimize own workload Finally, one of the most important functions the supremal subsystem needs to perform is to minimize its own workload. Given the overriding influence the supremal subsystem has, it is tempting and easy to push all sorts of functionality into it. However, the goal is to design the architecture such that the infimal subsystems actually perform most of the work with as less interference from the supremal subsystem as possible. Therefore, the supremal subsystem should be able to set goals and then hand off the execution of those goals, reserving intervention to exceptional circumstances. This requires, among other things, well-defined system initialization, hand over procedures and trading of responsibilities between the supremal and infimal subsystems.

4 DISCUSSION AND PRELIMINARY EVIDENCE

The approach of introducing a single supremal subsystem in order to facilitate autonomy can be applied not just to the system level, but also to the individual subsystems, where each subsystem is considered as a system in itself. Thus, an autonomous system may be composed out of multiple autonomous and non-autonomous subsystems. Equally likely is the scenario that a non-autonomous system includes (partially) autonomous subsystems. This is the case in a modern car, for example, where subsystems like Active Safety are architected as a hierarchy with a single 'Active Safety Coordinator' function at the top.

The intervention and coordination influence of the supremal subsystem is not restricted to the infimal subsystems in the hierarchical layer immediately below it. Subsystems on any level may be able to traverse through the hierarchy either through the interim levels or by directly reaching out to the lower levels. This is true of all multi-level, hierarchical systems in general.

It must be emphasized that the introducion of the single supremal subsystem is not a sufficient condition to reach autonomy, even from an 'architecture only' perspective. Complex, safety critical industrial embedded systems are affected by an extremely wide range of concerns like scalability, performance, development methodology, certification requirements, maintainability, robustness etc. The value of the single supremal subsystem lies in the fact that in its absence, it is difficult to reach autonomy at all in a distributed, well-partitioned embedded system. Thus, it should be seen as a first and necessary step towards the solution, rather than the solution in itself.

Our approach has some similarities and differences to the concept of 'centralized control' in the literature. Even though the single supremal subsystem is responsible for the overall system behavior, the subsystems that actually realize most of the system functionality are still distributed. In fact, the minimization of tasks which the supremal subsystem should perform is an important goal for the architect. As much functionality as possible should be pushed out to the infimal units.

Even in the area of decentralized autonomy of large systems, it has been argued (Kopetz, 2003) that the constituent units of abstraction need to be autonomous in and by themselves. Thus, our suggested approach does not run counter to decentralization, but can be applied to the design of autonomous systems that collectively make up the decentralized systems.

A significant prior work conducted by the authors

was the creation of a reference architecture for cooperative driving (Behere et al., 2013). The work resulted in the creation of a vehicle subsystem that enable autonomous vehicle operation under specific circumstances. Reinterpreting that work from the perspective of the proposed approach shows that the reference architecture was essentially the single supremal unit at the top of a hierarchy of vehicle propulsion subsystems. The internal structure of the reference architecture closely followed the structures described in section 3. Specifically, the reference architecture interacted with the vehicle operator, interpreting the commands and controlling the underlying brake and cruise control subsystems, while minimizing its own workload.

Informal discussions with practicing engineers have, on a number of occasions, revealed similarities between their work and our approach when their projects demanded elements of autonomy. Specific architectures for certain spacecraft and automation systems as well as some architectures for control of hybrid and intelligent systems(cf. (Albus and Proctor, 1996)) also show a similar approach. We see this as evidence of the wide utility of the suggested approach and our ongoing efforts are directed towards making the pattern explicit so that it is available as a ready reference, rather than knowledge which needs to be gleaned from case studies and experience.

5 CONCLUSION

Distributed embedded system architectures with good separation of concerns are highly recommended in the state of practice. In this paper, we have argued that such architectures are insufficient when it comes to generating intelligent system level autonomy while retaining characteristics like safety, predictability, efficiency and certifiability. We have proposed an approach to complement the distributed design in order to facilitate system autonomy. This approach involves the introduction of a single subsystem to top off the system hierarchy. The minimal objectives to be fulfilled by this subsystem have also been presented. We believe that this architectural pattern is applied in practice by engineers in many situations, without fully realising its rationale and potential. Our work aims to make the pattern more explicit in the context of autonomous architectures, explains the driving forces behind it and is the basis of ongoing work related to formalized reference architectures for autonomous embedded systems.

ACKNOWLEDGEMENTS

The work presented in this paper has been partially funded by the EIT ICT Labs project CPSE (Cyber-Physical Systems Engineering) and the Swedish VINNOVA/FFI program supporting the FUSE project. The authors would like to gratefully acknowledge the support.

REFERENCES

Albus, J. and Proctor, F. (1996). A reference model architecture for intelligent hybrid control systems. In *Proceedings of the 1996 Triennial World Congress, International Federation of Automatic Control (IFAC)*.

Behere, S., Törngren, M., and Chen, D. (2013). A reference architecture for cooperative driving. *Journal of Systems Architecture*.

Crutchfield, J. P. and Mitchell, M. (1995). The evolution of emergent computation. *Proceedings of the National Academy of Sciences of the United States of America*, 92(23):10742–6.

Eloranta, V. and Hartikainen, V. Patterns for distributed embedded control system software architecture.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis.

Forrest, S. (1990). Emergent computation: self-organizing, collective, and cooperative phenomena in natural and artificial computing networks: introduction to the. *Physica D: Nonlinear Phenomena*, 42:1–11.

Guth, R. and D'Epinay, L. (1983). The Distributed Data Flow Aspect of Industrial Computer Systems. In *5th IFAC Workshop on Distributed Computer Control Systems*, Sabi-Sabi.

Kleinrock, L. (1985). Distributed Systems. *Computer*, 18(11):90–103.

Kopetz, H. (2003). The future of autonomous decentralized systems. *The Sixth International Symposium on Autonomous Decentralized Systems, 2003. ISADS 2003.*, page 329.

Kopetz, H. (2008). 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.

Mesarovic, M., Macko, D., and Takahara, Y. (1970). *Theory of Hierarchical, Multilevel Systems*. Academic Press.

Schoeffler, J. D. (1984). Distributed Computer Systems for Industrial Process Control. *Computer*, 17(2):11–18.

Törngren, M. and Wikander, J. (1978). A decentralization methodology for real-time control applications. *Communication*.

Watkins, C. B. and Walter, R. (2007). 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.