Embedded-Software Architects

It's Not Only about the Software

Pablo Oliveira Antonino, Andreas Morgenstern, and Thomas Kuhn,
Fraunhofer Institute for Experimental Software Engineering

// Computer scientists' knowledge of embedded-systems concepts such as controllers and actuators is usually limited. So, the role of embedded-software architect is often played by engineers from other fields who have a limited education in software architecture. //

IN THE MID-1990S, software architecture had its boom, triggered mainly by the Software Engineering Institute's work, Rational's 4+1 views, and Siemens' four views.[1] From that point on, software architecture gained great visibility in the computer science community. Consequently, many of the mature architecture-centered methodologies and tools have been developed mainly by the computer science community.[2]

Computer scientists have been—and the great majority still are—educated with a mind-set concentrating on the structure of information systems. These systems traditionally rely on

- standard infrastructures to support a software platform, which provide commonly used base services and abstractions from the hardware, and
- virtualization to separate independent software systems from each other and improve robustness and availability through live migration.[3]

However, with the advent of embedded systems, which has been followed by the emergence of cyber-physical systems,[4] a tremendous demand has arisen for professionals who understand both software and hardware specifications. Because of computer scientists' traditional education, they have difficulty reasoning on aspects such as communication bus capacity, how delays and jitter affect control loop behavior, and functions realized by solenoids and other electromechanical devices.[5]

Because computer scientists generally lack knowledge of embedded systems' nonsoftware properties, electrical and mechanical engineers are assuming roles that computer scientists exclusively used to play, such as the role of architect. This wouldn't be a problem if these engineers had an architecture-related formal education or had developed this competence throughout their career. But in companies for which we provide architecture consultancy, we've observed that the architects have limited architecture knowledge. This has been the case in the automotive and transportation, automation and plant-engineering, and medical-device domains. (The average teams for which we provide consulting have more than 150 engineers, are globally distributed, and deal with systems of approximately 10 MLOC.)

Here, we look in detail at the

 0740-7459/16/$33.00 © 2016 IEEE

problems we've observed with embedded-software architects, the problems' causes, and possible ways to deal with them.

Recurring Problems

In our projects, we've identified the following two main problems.

Incompleteness and Inconsistencies Due to Missing Traceability

The development of software-based systems, regardless of whether or not they're embedded, involves different stakeholders, such as the project manager, developers, and users. They all have different concerns to be understood, prioritized, and realized.

One main responsibility of the architect is to identify core aspects that will drive the architecture design, which centers on identifying concerns that are risky and expensive to change—the *architecture drivers* (see Figure 1). In industry projects, we've observed that embedded-software architects neglect the fundamental traceability that should exist between the architecture drivers and architecture design. Instead, they tend to focus on isolated parts of the architecture design. Other scientists have also observed this situation in cases involving medical devices submitted for US Food and Drug Administration approval.[6]

For example, we've often encountered architects—usually non-computer scientists—who focus on only the hardware and network specifications and assume that the software engineers will deliver the software in an optimal state. On the other hand, we've observed experienced architects—mainly computer scientists—who focus on only the software and assume that the electrical and mechanical engineers are aware of all the assumptions and constraints necessary to properly deploy a complex piece of software in the hardware. So, the whole architecture specification is often incomplete and inconsistent, which results in an intense effort to properly integrate the embedded system's various aspects.

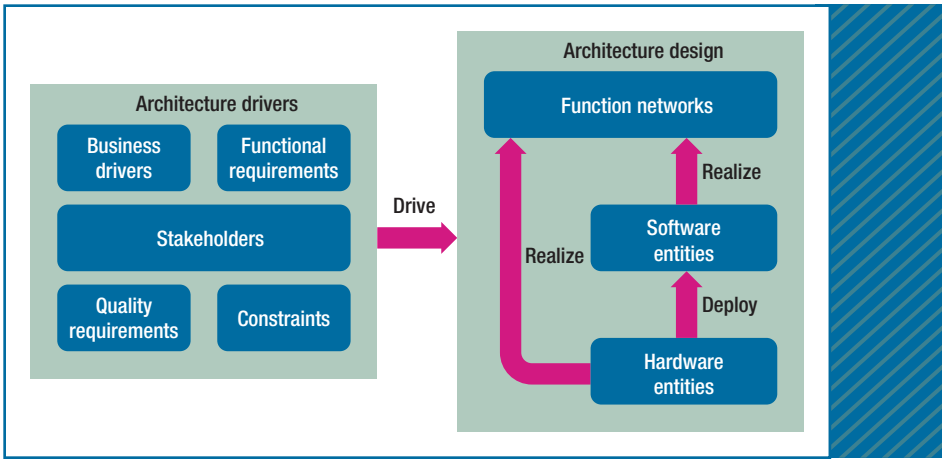

FIGURE 1. Architecture drivers and architecture perspectives as the main building blocks of architecture specifications. Architecture drivers are critical aspects that are risky and expensive to change.

Architectural Smells

Martin Fowler introduced the term "bad smells" in the software context.[7] He related them to characteristics in source code snippets that negatively affect quality aspects such as testability and reusability.

Similarly, an *architectural smell* is a "commonly used architectural decision that negatively impacts system quality."[8] Here are two examples:

- An *Extraneous Connector* occurs when two connectors of different types connect a pair of components.
- A *Scattered Functionality* occurs when multiple components realize the same high-level concern and some of them are responsible for orthogonal concerns.

Our intention isn't to present a catalog of architectural smells but to discuss how the profiles of embedded-systems architects have contributed to these smells' occurrence.

For example, in Figure 2a, the vehicle sensor processor computes sensor measures and makes them available on the CAN (Controller Area Network) bus. Despite this architecture practice being common, an Extraneous Connector often occurs because of the additional direct connection between two components deployed on the same ECU (Electronic Control Unit), as indicated in Figure 2a by the obstacle distance request sent from the cruise control to the vehicle sensor processor. This additional dependency might result in challenges for evolving that processor. This architectural smell also implies a deployment constraint because the vehicle speed control and cruise control must be deployed on the same ECU.

In this context, one possible solution to the Extraneous Connector involves the AUTOSAR (*Aut*omotive Open *S*ystem *A*rchitecture; www

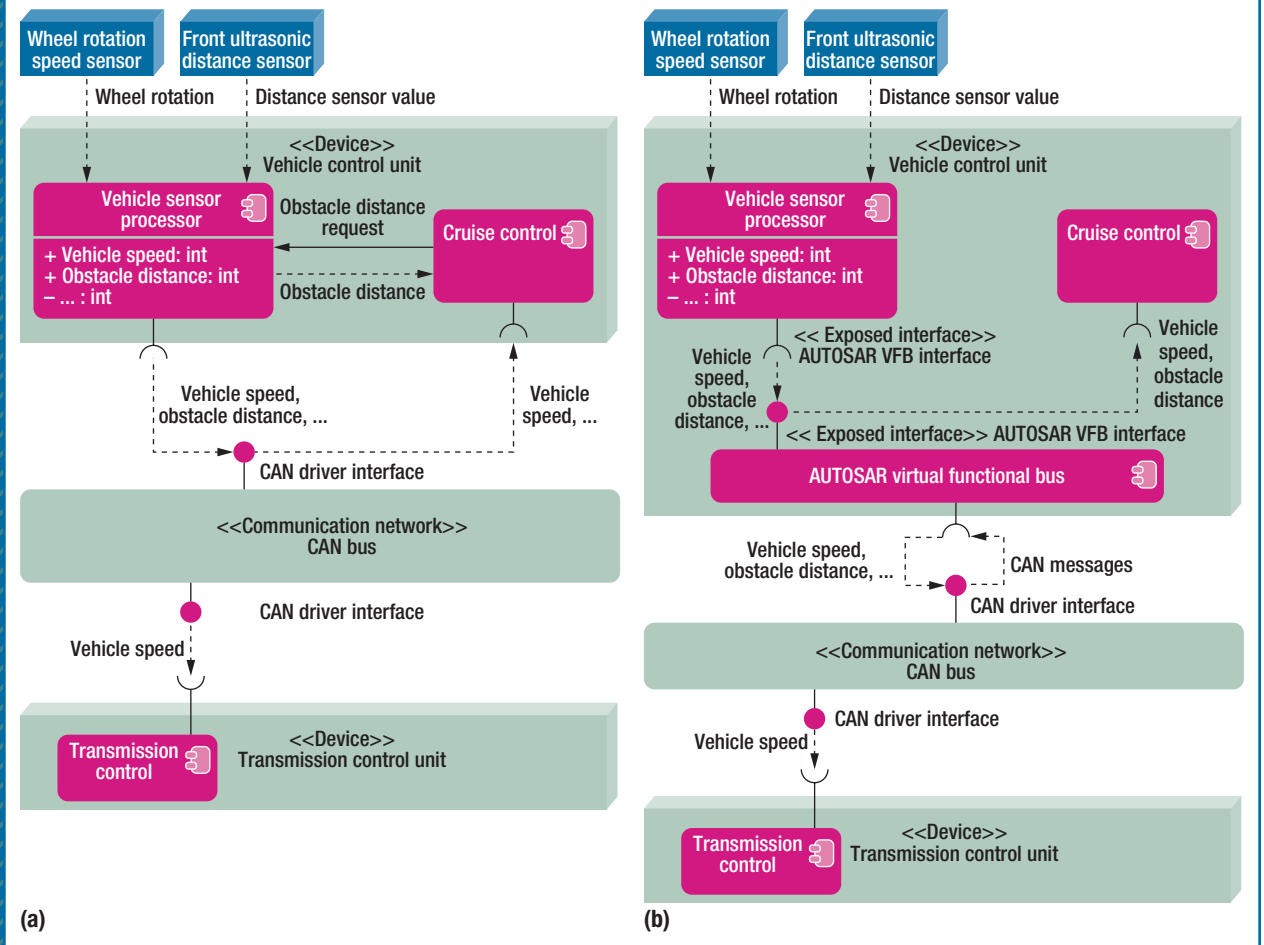

FIGURE 2. Dealing with an architectural smell. (a) An Extraneous Connector smell, which is often found in embedded software. (b) A mitigation strategy using the AUTOSAR (*Aut*omotive *O*pen *S*ystem *A*rchitecture) architecture. Architectural smells are architectural decisions that negatively impact system quality. CAN stands for Controller Area Network.

.autosar.org) reference architecture. Figure 2b shows this solution, in which every information exchange between software components is through an AUTOSAR Virtual Function Bus.

Had the embedded-software architects taken computer science courses, they would have learned such strategies for overcoming architectural smells. For example, some courses teach architecture tactics such as multilayers and microkernels. Other courses teach the "Gang of Four" design patterns[9] not only as a way to improve low-level design but also to mold the students' mind-set to consider these patterns at the architecture level.

The consequence of having architects who aren't familiar with these best practices is the enormous frequency of architectural smells, such as those we've often found in the embedded architectures we've reviewed. We've received several requests to assess an architecture's quality because of the difficulties companies faced in evolving their systems when system complexity increased beyond a certain limit. In many cases, a key reason for these difficulties was well-known architectural smells. Also, in most of these cases, the architects had no computer science education and were barely familiar with architecture best practices. So, they designed the architecture to address only a

particular set of features and not to cope with evolutionary processes.

Information systems architectures aren't completely free of architectural smells, either. However, our discussions with colleagues who work exclusively on information systems led us to conclude that embedded-software architectures suffer from architectural smells more than information systems architectures do.

The Problems' Sources

The recurring problems we just described have the following two sources.

Misunderstood Responsibilities

A recurring question is, what's the architect's role in the development team? Peter Eeles and Peter Cripps understood that architects not only are software project technical leads but also are responsible for the success or failure of the project as a whole.[10] Eeles and Cripps argued that, to achieve these goals, architects must

- lead software design teams, monitoring code quality and test coverage and ensuring that the system works as expected;
- understand the development process and ensure that the teams involved in development will follow it;
- understand the business domain—its concepts and terminologies;
- be good communicators; and
- be decision makers.

Our experience has shown that embedded-software architects must also

- lead hardware development teams and monitor the hardware's architecture-relevant properties and
- understand, as much as is necessary, the development process for all relevant portions of the system's subsystems (software, hardware, electrical, and their integration).

In the companies for which we provide consulting, we never found an architect with these seven characteristics. Rather, the architects focused only on the system's technical aspects that were closest to their strengths. For example, architects with a computer science background tended to focus on only the software artifacts, making unrealistic hardware assumptions. Electrical and mechanical engineers tended to go in the opposite direction, focusing mostly on sensors, actuators, communication buses, field-programmable gate arrays, and other hardware artifacts and neglecting the importance of sound software structures.

This technical experience is important because architects are supposed to make critical, long-term technical decisions. In addition, successful architects must have leadership and social skills. We've observed that some of our customers assigned the architect role to people who didn't fit the architect's profile. In spontaneous interviews with embedded-software architects, we discovered that most of them got their role because they were great engineers and had been working for the company for years. But we also found that many of them fit the profile of a senior engineer, not an architect.

Obviously, architects need communication skills, as Eeles and Cripps also highlighted. An embedded-systems architect might not have the technical capabilities to approach software and hardware artifacts equally. So, besides a general notion of both groups of artifacts and how they relate to each other, an architect should know who can dig into detail in each group of artifacts, assess the risks, and make the appropriate architecture decisions.

Aversion to Models or Inadequate Abstraction Skills

A considerable number of computer scientists are still skeptical about using models to document software-based systems. Often, engineers with a computer science education have told us that the architects were "those who make more money than we do and whose main duty is to draw boxes and lines that represent a system that will soon become outdated." They claimed that the software being implemented was so subject to changes that investing time in coding was more worthwhile than investing in documenting or modeling the architecture. Some of them even claimed that if the software is written well, the source code is the only documentation needed.

On the other hand, we've observed that electrical and mechanical engineers who were either involved in pure implementation tasks or had the role of architect saw the value of detailed model-based architecture documentation. However, we've also observed their limited ability to think on multiple abstraction levels. One reason for this is that their education didn't prepare them properly. In contrast, many computer scientists have this abstracting capability. Nevertheless, as we mentioned before, many of them didn't see the value in documenting or modeling the architecture because the source code was all that mattered. At this point, a dissonant

situation existed. Some professionals had the skills to perform the task but didn't see its value; the other group saw the value but wasn't educated to do the task properly.

When comparing these two groups, we understand that because mechanical and electrical engineers are used to electrical, hydraulic, and other models necessary to their domain, they see value in architecture models. They have this mind-set because the systems they usually build have something to do with tangible artifacts such as valves, pumps, and printed circuits, which must be well understood and analyzed before realization.

Regarding pure software, which is the main (and usually the only) artifact that traditional computer scientists manipulate, they have, in a certain sense, too much freedom to realize their products. For example, if an information system performs an erroneous operation, in many cases a rollback followed by a code fix, recompilation, and software redeployment is enough to recover from a critical software failure. However, when the software controls safety-critical embedded systems such as those in airplanes, cars, and medical devices, more is required than the usual architecture practice as defended by some of the computer scientists we mentioned.

We've actually observed widespread adoption of models in the embedded-systems domain. For instance, our customers in the transportation domain have made these comments:

> *Ninety percent of the software embedded in our systems is generated from models.*
>
> *For the critical parts of our system, we completely generated the code.*
>
> *I do not trust code written by humans.*
>
> *Engineers who do not welcome model-based engineering are not welcome in our company.*

Two important reasons to adopt architecture specification of embedded systems are to

- communicate the software properties to the engineers responsible for the nonsoftware artifacts and
- make the embedded system architecture specification as a whole (software plus hardware) consistent.

In this case, the architecture documentation fulfills one of its purposes: to facilitate communication among the different stakeholders in development.

We don't claim that the whole architecture model should be created before implementation starts. Rather, the architecture documentation created before implementation should contain just enough to document and communicate the system aspects that are risky and expensive to change.

Using Adequate Tools and Methodologies

Several approaches offer guidance for performing architecture activities. One such approach is the elicitation and documentation of architecture drivers using scenarios. Another is architecture evaluation using Fraunhofer's RATE (Rapid Architecture Evaluation) and the Software Engineering Institute's ATAM (Architecture Tradeoff Analysis Method). Both approaches offer great support and have been used around the world for both embedded systems and information systems.

A strong tendency exists to associate specification of the architecture design with UML (www.uml.org) or SysML (Systems Modeling Language; www.sysml.org), which are strongly influenced by the computer science community. Unfortunately, these approaches by themselves aren't specific enough for properly architecting embedded systems. But approaches exist that have been successfully used for this, which we describe next.

AADL (Architecture Analysis and Design Language; www.aadl.info), which the Society for Automotive Engineers defined in 2012, offers tailored support for dependability aspects such as safety and security in architecture specifications.[11] It also provides the means to capture architecture concepts from different abstraction levels. AADL has been widely used in the development of safety-critical systems—for example, by the SAVI (System Architecture Virtual Integration; http://savi.avsi.aero) initiative (which includes companies such as Airbus, Boeing, and Embraer), many other industries in Europe and Asia, and the US Army.

SCADE is part of the SCADE Suite (www.esterel-technologies.com/products/scade-suite). It supports the specification of control flows and state machines for control logic, which are important architecture principles of safety-critical systems. SCADE has been successfully used in the aerospace, defense, and railways industries, such as in the design of safety-critical systems of the Airbus A380 and Boeing 787.

EAST-ADL (Electronics Architecture and Software Technology—Architecture Description Language) is an architecture description language for automotive embedded systems.[12] It was designed in the context of several European research projects. EAST-ADL defines architecture views, which support specifying the structure and behavior of vehicle features and their realization through software and hardware. It does this with precise traceability between the elements.

The PREEvision (www.vector.com/preevision) tool also supports architecting automotive embedded systems, using the SPES (Software Platform Embedded Systems) 2020 methodology.[5] Many of our customers from the transportation industry in Europe and the US have adopted PREEvision. We've seen how it helps architects with different educational backgrounds design complex embedded systems.

The Fraunhofer Embedded Modeling Profile also centers on SPES 2020.[13] It continues to evolve through its use in architecture consultancy services in various embedded domains such as the automotive domain, agriculture, avionics, and astrophysics. In the astrophysics domain, the architects are physicists who are using the profile to architect gamma-ray telescope controller systems' software and hardware.[14] The Fraunhofer Embedded Modeling Profile is available for the Enterprise Architect (www.sparxsystems.com) and MagicDraw (www.nomagic.com) tools.

One front that still requires thorough investigation is the modeling of architectures of embedded systems that are tightly integrated into information systems—the cyber-physical systems we discussed earlier. Owing to the level of integration needed to ensure that a cyber-physical system works properly, it's necessary to represent concepts from both types of systems with the appropriate degree of abstraction. Methodologies exist that support the architecture activities of information and embedded systems, but they're most likely disjoint and often conflict with each other. So, they must be evolved into a more cohesive methodology.

INDUSTRY PRACTITIONERS BECOMING ARCHITECTS—LOOKING FORWARD

Before an industry practitioner is assigned the role of software architect, it's important to identify whether that person has an architect's required characteristics, such as those we discuss in the main article. For those who fit the profile, it's crucial to tailor their capabilities with complementary education that addresses traditional architecture activities such as architecture construction and assessment.

Regarding those who provide such coaching, it's important to consider that many companies that offer architecture consultancy services have a hard time supporting embedded-systems companies. This is because these consultancy companies normally consist of computer scientists with little or almost no knowledge of architecting embedded systems. The need exists for professionals with knowledge of not only architecture but also embedded systems.

We've identified the need to address architecture in the formal education of not only computer science students but also students in the other domains most likely involved in embedded-system development, such as electrical and mechanical engineering. We're not proposing how to update the university curriculum to integrate architecture-related courses; we simply wish to indicate measures that should be considered to educate embedded-software architects. (For more on educating practitioners, see the sidebar.)

First, universities should consider teaching software architecture in non-computer science courses covering such topics as electrical engineering, mechanical engineering, and mechatronics and in other courses that deal mainly with artifacts that have a close relationship with software.

Second, computer science courses should emphasize the architecture discipline, not just molding computer scientists to think beyond the source code but also teaching them how to better architect software-based systems. We're not talking about teaching design patterns, which is also a fundamental topic. Rather, we're focusing on the architect's role and on educating students in high-level architecture design and assessment.

Finally, computer science courses should also teach embedded-system basics such as controllers, sensors, actuators, and buses. Someone might

ABOUT THE AUTHORS

PABLO OLIVEIRA ANTONINO is a senior engineer and project manager at the Fraunhofer Institute for Experimental Software Engineering. He constructs and analyzes architectures of dependable embedded systems for domains such as the automotive domain, agriculture, medical devices, and avionics. Antonino received a PhD in computer science from the University of Kaiserslautern. Contact him at pablo.antonino@iese.fraunhofer.de.

ANDREAS MORGENSTERN is an engineer at the Fraunhofer Institute for Experimental Software Engineering. His research interests include behavior modeling in software architecture models and integrating formal methods such as static analysis into software engineering. Morgenstern received a PhD in computer science from the University of Kaiserslautern. Contact him at andreas.morgenstern@iese.fraunhofer.de.

THOMAS KUHN heads the Fraunhofer Institute for Experimental Software Engineering's Embedded Software Engineering department. His research interests include the virtual development of embedded systems and the substantiating of architecture decisions on the basis of measureable facts obtained by architecture prototypes. Kuhn received a PhD in computer science from the University of Kaiserslautern. Contact him at thomas.kuhn@iese.fraunhofer.de.

claim that computer-engineering courses, not computer science courses, are the ones intended to prepare such professionals. If this is the case, computer-engineering courses should address software architecture principles, following the same recommendations we've given for electrical and mechanical engineering. Nevertheless, with the advent of cyber-physical systems, computer scientists who have at least a basic knowledge of embedded systems will have an advantage in industry over those lacking this knowledge.

References

1. M. Shaw and P. Clements, "The Golden Age of Software Architecture," *IEEE Software*, vol. 23, no. 2, 2006, pp. 31–39.
2. P. Kruchten, H. Obbink, and J. Stafford, "The Past, Present, and Future of Software Architecture," *IEEE Software*, vol. 23, no. 2, 2006, pp. 22–30.
3. P. Liggesmeyer and M. Trapp, "Trends in Embedded Software Engineering," *IEEE Software*, vol. 26, no. 3, 2009, pp. 19–25.
4. "Cyber-Physical Systems," Program Solicitation NSF 16-549, US Nat'l Science Foundation, 2016; www.nsf.gov/pubs/2016/nsf16549/nsf16549.htm.
5. K. Pohl et al., *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*, Springer, 2012.
6. P. Maeder et al., "Strategic Traceability for Safety-Critical Projects," *IEEE Software*, vol. 30, no. 3, 2013, pp. 58–68.
7. M. Fowler, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Longman, 1999.
8. J. Garcia, D. Popescu, and G. Edwards, "Toward a Catalogue of Architectural Bad Smells," *Architectures for Adaptive Software Systems*, LNCS 5581, Springer, 2009, pp. 146–162.
9. E. Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Longman, 1995.
10. P. Eeles and P. Cripps, *The Process of Software Architecting*, Addison-Wesley Professional, 2010.
11. *Architecture Analysis & Design Language (AADL)—SAE AS 5506*, SAE Int'l, 2012.
12. P. Cuenot et al., "Managing Complexity of Automotive Electronics Using the EAST-ADL," *Proc. 12th IEEE Int'l Conf. Eng. Complex Computer Systems* (ICECCS 07), 2007, pp. 353–358.
13. T. Kuhn and P.O. Antonino, "Model-Driven Development of Embedded Systems," *Proc. 2014 Embedded Software Eng. Congress*, 2014, pp. 47–53.
14. I. Oya et al., "The Software Architecture to Control the Cherenkov Telescope Array," *Proc. SPIE*, vol. 9913, 2016; http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=2540562.