Ideal Software Architecture for the Automotive Industry

Adnan Shaout, and Gamal Waza

The Electrical and Computer Engineering Department
The University of Michigan - Dearborn
Dearborn, Michigan, USA
shaout@umich.edu; gwaza@umich.edu

Abstract - An Ideal Software Architecture is the foundation to solving software engineering methods. In this paper, all of the automotive industry current and future challenges will be analyzed, and the paper will propose an Ideal Software Architecture for the automotive industry. After gathering all the requirements, software architecture styles will be evaluated against the unique environment found in the automotive industry. Research results show that the component based and layered architectural styles were the most suitable for the automotive industry. Some other architectural styles were suitable for particular layer and/or component. One of the most interesting outcomes of this research was the introduction of run-time adaptation to the automotive software architecture. This was handled by introducing the Dynamic Configuration manager Module in the middleware layer. Finally, an ideal architecture design that solves all current and future concerns was presented.

Keywords–Software for Automotive Industry, Software Architecture, Dynamic Configuration Manager, Layered Software

1. Introduction

In the past 30 years, the amount of software in cars has been growing. Cars in the future also expect to demand more and more software functionalities. Additionally, the automotive industry has specific constraints and requirements that mandate unique solutions [1]. These specific constraints also bring many challenges to software engineering in cars [2]. Therefore, there is an opportunity for an automotive software engineeringresearch to help solve many current and future challenges [3, 4].

An Ideal Software Architecture is the foundation to solving software engineering methods. In this paper, all of the automotive industry current and future challenges will be analyzed and the paper will present an Ideal Software Architecture for the automotive industry.

The paper is organized as follows; Section 3 of the paper will present the current and future challenges of software engineering by analyzing the needs and concerns of all the involved stakeholders. In section 4 software architecture requirements will be generated to address all current and future challenges. Section 5 will present the software architecture design were the requirements will be used as input to choose most suitable design decisions. Section 6 will present the layered Ideal Software Architecture for the automotive industry. Finally, section 7 presents conclusion remarks.

2. Automotive Industry Challenges Study

A challenging tree was constructed by starting with all automotive industry stakeholders. Then, needs, concerns and challenges of each stakeholder were listed. Finally, software architecture requirements were identified for each branch. [1][5][6][7][8][9]

Figure 1shows the basic overview for the challenges tree.

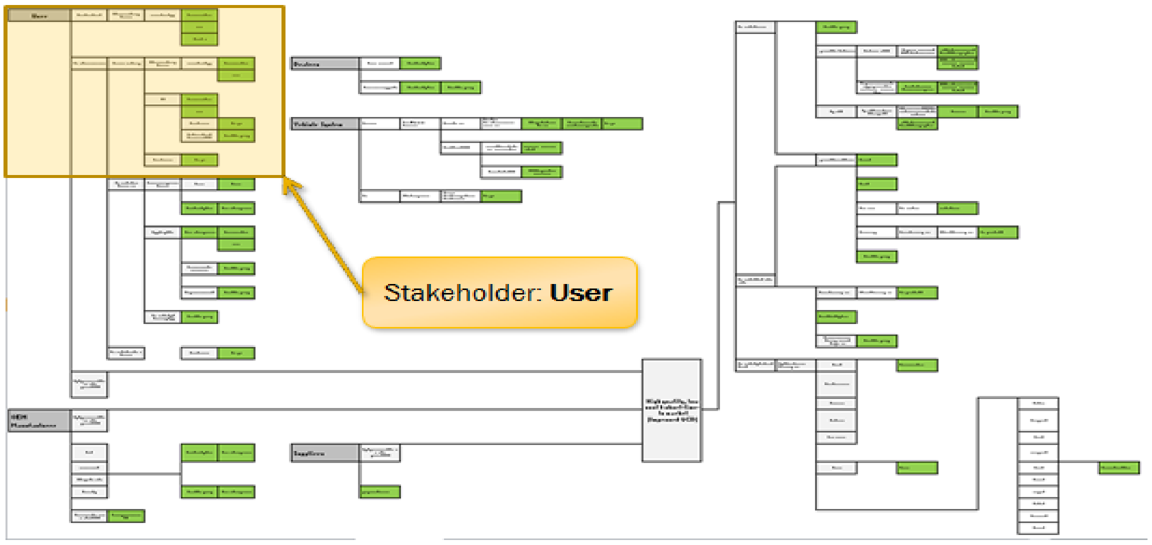


Figure 1: Automotive Industry Challenges Tree.

3. Software Architecture Design

3.1. Requirements Analysis

Requirements was analyzed and categorized to better understand system effect. Some requirements impacted the overall architecture, but, other requirements were very straight forward. For example, addressing quickly changing platforms has a big impact to the architecture style. On the other hand, the requirements for a functional safe Real-Time OS was straight forward. So, from the high level analysis, we generated core requirements to assist in the design decision.

Table 1 show the software architecture requirements analysis summary. As seen in table 2, the core architecture requirements are specified as follows:

- Features-Specific Software
- Hardware-Specific Software
- Communication Abstraction
- Functional Safe Real-Time OS
- Dynamic Configuration Manager Module

Table 1: Software Architecture Requirements Analysis.

SW Architecture Requirements Description	System Effect	More Architectural Specific Requirements
Need to address the demand for more software	Architectural	Need to address features specific software scalability
Need to address the concern with Quickly Changing Platform	Architectural	Separate Hardware-specific Software
Need to address concerns with real time requirements for many features	Functional	RTOS
Need to allow OEMs to own innovative features	Architectural	Need for Features specific Modules with standard Interface
Need to allow suppliers to own particular modules	Architectural	Modular Design with standard Interface
Need to address concern with changing communication methods	Architectural, Module	Communication Abstraction
Need to address concerns with ecu dependent functionalities	Architectural	Separate Hardware specific functions
Need to allow for software to be Composable and ease of Functional Integration	Architectural	Modular Design with standard Interface
Design for high reusability	Architectural, Module	Reusability
Design for high portability	Architectural, Module	Portability
Ensure standard interfaces between components and layers	Architectural, Module	Modular Design with standard Interface
Design with conformance to functional Safety standards	Functional	Functional Safe OS
Active safe Control Apps to be protected from malicious attacks	Functional	secure Dynamic Configuration Module
Dynamic Configuration to be secure	Functional	secure Dynamic Configuration Module
Due to connectivity, privacy concerns needs to be addressed by enhanced security	Functional	secure Dynamic Configuration Module
Need to support Cloud computing	Module	Dynamic Configuration Module
Need to support Over the Air Update	Module	Dynamic Configuration Module
Need to support dynamic configuration	Module	Dynamic Configuration Module
Need to design software with high quality	Module	Software Modules to be designed with high Quality

3.2. Architecture Design

Software architecture design approach considered the following Goals

- Increase Design qualities
  - Design Quality Attributes
- Faster Development
- Lower Cost
- Meet all Software Architecture Requirements

3.3. Software Architecture Styles

The software architecture styles that were considered are as follows [3, 10]:

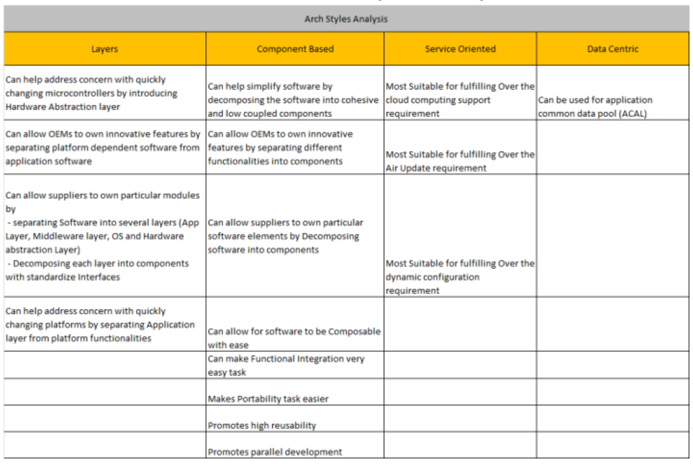
- Layered Architectural Style
- Component-Based Architectural Style
- Service-Oriented Architectural Style
- Data Centric Architectural Style

3.4. Chosen Software Architectural Styles

Software architecture styles were evaluated against the requirements generated from the unique environment found in the automotive industry. Results show that the component based and layered architectural styles were the most suitable for the automotive industry as shown instable 2.

Some other architectural styles were suitable for particular layer and/or component. For example, data centric architectural style was used to address concerns with the complex interaction of the many application components. The Application Communication Abstraction Layer (ACAL) was chosen to have data centric approach for all application components. Finally, an ideal architecture design was proposed and determined that solves all current and future concerns [3].

Table 2: Architecture Styles Analysis



Arch Styles Analysis			
Layers	Component Based	Service Oriented	Data Centric
Can help address concern with quickly changing microcontrollers by introducing Hardware Abstraction layer	Can help simplify software by decomposing the software into cohesive and low coupled components	Most Suitable for fulfilling Over the cloud computing support requirement	Can be used for application common data pool (ACAL)
Can allow OEMs to own innovative features by separating platform dependent software from application software	Can allow OEMs to own innovative features by separating different functionalities into components	Most Suitable for fulfilling Over the Air Update requirement	
Can allow suppliers to own particular modules by - separating Software into several layers (App Layer, Middleware layer, OS and Hardware abstraction Layer) - Decomposing each layer into components with standardize Interfaces	Can allow suppliers to own particular software elements by Decomposing software into components	Most Suitable for fulfilling Over the dynamic configuration requirement	
Can help address concern with quickly changing platforms by separating Application layer from platform functionalities	Can allow for software to be Composable with ease		
	Can make Functional Integration very easy task		
	Makes Portability task easier		
	Promotes high reusability		
	Promotes parallel development		

4. Layered Arch vs. Component-Based Arch Styles Analysis

After determining that Layered and component-based architecture styles are the most suitable for the ideal auto industry software architecture, further analysis was necessary to determine which of the two styles is more suitable to be the high level architecture. Table 3 shows the performance comparison between layered and component based architectural styles. As seen in table 3, Layered architecture is the most suitable high level architecture for the automotive industry. Results make sense given the need for logical separation of hardware specific and vehicle features specific functionalities. Finally, given the high score of component based architecture style, it was decided to design each layer with component based architecture style [3, 10].

Table 3: Layered Arch vs. Component-Based Arch Styles Analysis

Rating Scale: 1-5 (0: Not Suitable & 5: Most Suitable)				Performance score		
				4.73	4.43	
SW Architecture Requirements Description	**More** Architectural Specific Requirements	Quality Attributes	Importance	Layers	Component Based	Solutions comments
Need to address the demand for more software	Need to address features specific software scalability	Scalability, Simplicity, Modularity	25	5	4.5	**Layered Arch:** Abstracted Application Layer to address simplicity and scalability & Modularity. **Component Based Arch:** Many separate components to address scalability and modularity. But complexity could be introduced.
Need to address the concern with Quickly Changing Platform	Separate Hardware-specific Software	Modularity	25	5	3.5	**Layered Arch:** Abstracted Hardware Layer. **Component Based Arch:** Separate Hardware specific components
Need to allow OEMs to own innovative features	Need for Features specific Modules with standard Interface	Modularity	15	5	4.5	**Layered Arch:** Abstracted Application Layer to isolate features specific software **Component Based Arch:** Separate software components to allow ease of software ownership.
Need to allow suppliers to own particular modules	Modular Design with standard Interface	Modularity	10	4.5	5	**Layered Arch:** Abstracted Application Layer to isolate features specific software. In addition, abstracted hardware specific software to separate features specific software from hardware-specific. **Component Based Arch:** Separate software components to allow ease of software ownership.
Need to allow for software to be Composable and ease of Functional Integration	Modular Design with standard Interface	Modularity	10	3.5	5	**Layered Arch:** Separate layers to ease integration of subsystems. **Component Based Arch:** Separate software components to allow ease of integration for very large software system.
Design for high reusability	Reusability	Reusability	7.5	4.5	5	**Layered Arch:** Separate layers to make reusability of subsystems easier **Component Based Arch:** Separate software components to allow ease of achieving reusability for all software elements
Design for high portability	Portability	Portability	7.5	4.5	5	**Layered Arch:** Separate layers to make Portability of subsystems easier **Component Based Arch:** Separate software components to allow ease of achieving Portability for all software elements

5. Software Architectural High-Level Design

The most suitable high-Level Architecture Design was chosen to be Layered architecture style. As seen in figure 2, the layered architecture separated the application specific from hardware specific functionalities [3]. The layered architecture has the following layers:

- Application Abstraction Layer
- Middleware layer
- OS
- Hardware Abstraction Layer (HAL)

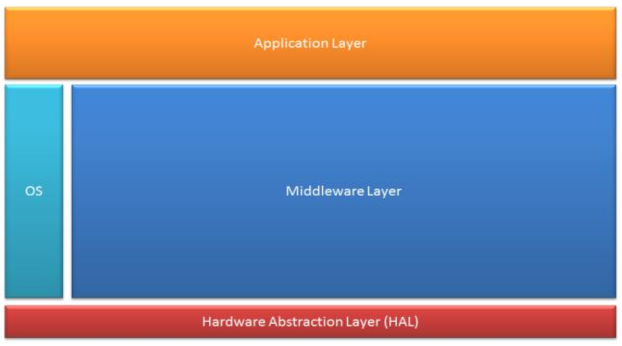


Figure 2: Software Architecture Design – Layered View.

Component-based architecture was used throughout the design to address many of the concerns in the automotive industry as shown in figure 3. For example, the application layer was chosen to be component-based style due to large amount of features and the need for many different suppliers to work on these features [10].

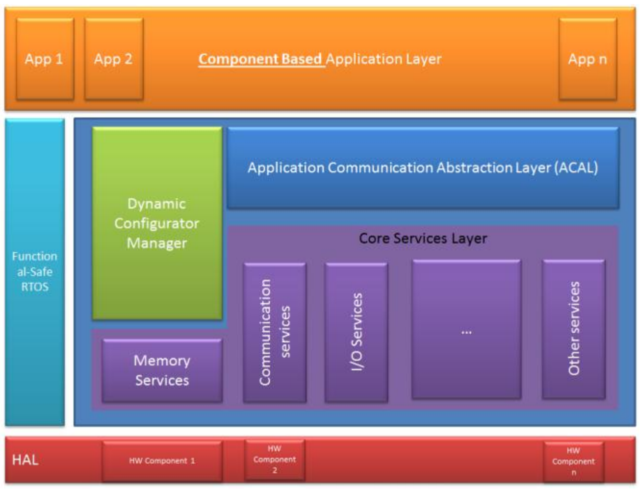


Figure 3: Software Architecture Design – Components View.

5.1 Software Architecture Design Details

5.1.2 Application Abstraction Layer

This layer is abstracted from any platform or hardware specific functionalities. It's designed for addressing features specific functionalities regardless of hardware or platform selected. To address the scalability and modularity concerns in this layer, it was designed as a component-based style.

To maximize software quality attributes, software components should be designed with the aim to achieve product line development. Therefore, application components are designed with separated common from configurable software elements. In the common part, core assets to be established to achieve high reusability and portability. In addition, configurable part of the component to be designed with all product variation configurations. (See Figure 4)

5.1.3 Middleware layer

The middleware layer is designed to provide core services to the application layer. This layer will also abstract communication challenges from the application layer. Middleware layer core services will be abstracted from hardware specific software via standardized interfaces. The middleware layer is made of the following layer components (as seen in figure 4):

- Application Communication Abstraction layer (ACAL)
- Core Services Layer (CSL)
- Dynamic Configuration Manager (DCM)

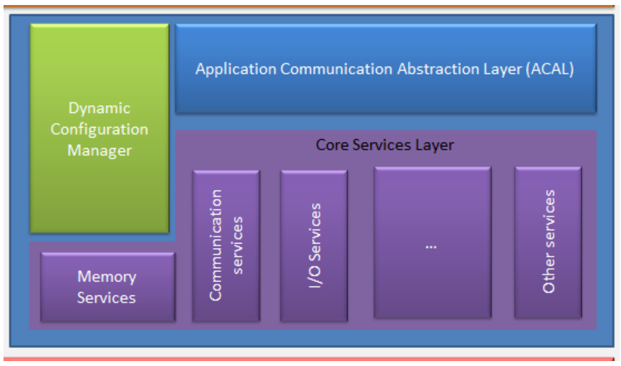


Figure 4: Middleware Layer Detailed View.

5.1.4 OS

The OS is s separate component and is treated as one of the core elements abstracted from application, middleware and hardware layers. Given the direct requirements, OS component must be real-time and support functional safety features.

5.1.5 Hardware Abstraction Layer (HAL)

The Hardware Abstraction Layer is designed to abstract all hardware related functionalities as shown in figure 6. Hardware specific functionalities can be related to the selected microcontroller or to the electronic control unit (ECU) hardware design. So, we separate the Micro-specific from the ECU-specific to address the concern with reusing micro specific software across ECUs. To maximize reusability and to ease components integrations, all interactions with HAL are done via standardized Interfaces.

5.1.6 Application Communication Abstraction layer (ACAL)

ACAL is designed to abstract application components interactions. This is done by allowing application components to read and write signals to and from a shared memory that is transparent to application layer. ACAL reads and writes signals from and to Core service layer if data needs to be communicated from and another ECU as shown in figures 7 and 8.

5.1.7 Dynamic Configuration Manager (DCM)

Due to the need for run-time adaptation and configuration, dynamic configuration manger (DCM) module is designed to handle the dynamic configuration of ECUs. Its functions are the following:

a) Install new Application components (New Apps)
b) Update existing Apps
c) Remove existing Apps

The benefits of the DCM are as follows:

1. Apps are updated over the air instantly
2. Maximum benefits of cloud computing

5.1.7.1 The Cloud to the electronic control unit (ECU) Configuration Process trails the following steps:

1. Cloud sends am encrypted update request to targeted vehicle
2. Vehicle receives Update request
3. Master ECU buffers the complete request
4. Master ECU sends corresponding update requests to targeted ECUs
5. Targeted ECUs validate request
6. Targeted ECUs decrypt and store requests and make them ready for launch
7. Master ECU waits for all dependent components to be ready for launch
8. Master ECU sends launch request
9. New application components are running

The DCM Configuration Process on an ECU level trails the following steps:

1. App Update request received by an ECU
2. Request is read by DCM

3. Request is written to memory (see figure 5)

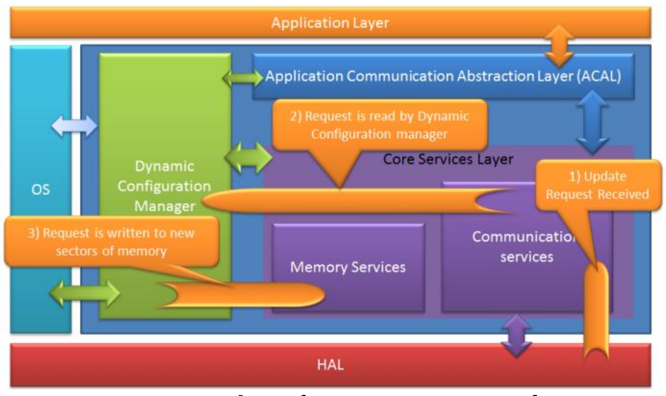

Application Layer
Application Communication Abstraction Layer (ACAL)
2) Request is read by Dynamic Configuration manager
Core Services Layer
1) Update Request Received
OS
Dynamic Configuration Manager
3) Request is written to new sectors of memory
Memory Services
Communication services
HAL

Figure 5: ECU internal Configuration process 1 of 3.

4. DCM acknowledges completion of app installation
5. ECU receives Launch request by master ECU
6. DCM Configures the following tables for Application components reconfiguration:
   - OS Configuration Table for tasks reconfiguration
   - Core Services Table for signals mapping and reconfiguration
   - ACAL Configuration Table for signals reconfiguration
7. New/Updated App is running (see figure 6)

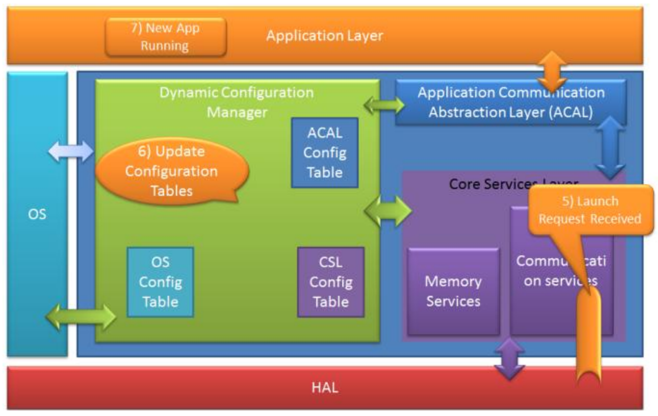


Figure 6: ECU internal Configuration process 3 of 3.

6. Conclusion

The automotive industry has unique challenges that require careful study of its current software engineering challenges. In this paper, we thoroughly analyzed the requirements of the automotive industry as whole by analyzing the needs and challenges of all involved stakeholders. Then, many software architecture requirements were presented.

Software architecture requirements were summarized into a manageable list. The list was further analyzed to map requirement into architecture design considerations. After that, software architecture solutions and techniques were considered to address the proposed requirements. Software Architecture styles was one of the main architectural techniques that was studied thoroughly in this paper. Results show that the component based and layered architectural styles were the most suitable for the automotive industry. This is due to the increasing complexity and the increasing demand for more software. Some other architectural styles were suitable for particular layer and/or component. For example, data centric architectural style was used to address concerns with the complex interaction of the many application components. The Application Communication Abstraction Layer (ACAL) was chosen to have data centric approach for all application components.

The paper also presented the core requirements proposed for the high level architecture requirements. The following summarizes all the main features that the new ideal software architecture for the automotive industry includes:

1. Features-Specific Software which was addressed by the application abstraction layer
2. Hardware-Specific Software which was addressed by the hardware abstraction layer
3. Communication Abstraction which was addressed by the middleware layer
4. Functional Safe Real-Time OS
   a. OS is separated that is Real-Time and functionally-safe
5. Dynamic Configuration Manager Module (DCM)
   a. DCM for automotive was introduced in the middleware layer

One of the most interesting outcomes out of this research paper was the introduction of run-time adaptation to the automotive software architecture. This was handled by introducing the Dynamic Configuration manager Module in the middleware layer. Finally, an ideal architecture design was presented that can solve all current and future concerns.

References

[1] Bray, Manfred, "Challenges in Automotive Software Engineering ", The Proceeding ICSE '06 Proceedings of

the 28th international conference on Software engineering, Pages 33-42, ACM New York, NY, USA ©2006.

[2] Adman Shout and Jamal Waza (2015), "Solutions to Automotive Software Engineering Challenges", the International Journal of Computer & Organization Trends (IJCOT), Volume X Issue Y – January 2015.

[3] El-Haik and Adnan Shaout, Software design for six-sigma – A roadmap for excellence. John Wiley, ISBN 978-0-470-40546-8, 2010.

[4] Adnan Shaout and Cassandra Dusute, (2013)," ResPCT – A new Software Engineering Method", International Journal of Application or Innovation in Engineering & Management (IJAIEM) 12/2013; Volume 2(Issue 12): Page 436 – 442, Impact Factor: 2.379.

[5] Alexander Pretschner, Manfred Broy, Ingolf H. Kruger, Thomas Stauner, "Software Engineering for Automotive Systems: A Roadmap" the proceedings of FOSE '07: 2007 Future of Software Engineering, May 2007.

[6] Dan Gunnarsson, Stefan Kuntz, Glenn Farrall, Akihito Iwai, Rolf Ernst, "Trends in automotive embedded systems", the Proceeding ofEMSOFT '12 Proceedings of the tenth ACM international conference on Embedded software, Pages 9-10, ACM New York, NY, USA ©2012 October 2012.

[7]http://www.autosar.org/fileadmin/files/presentations/AUTOSAR_OA_Summit_04NOV2014.pdf

[8] ArkadebGhosal, Paolo Giusto, Alberto Sangiovanni-Vincentelli, Joseph D'Ambrosio, Ed Nuckolls, Harald Wilhelm, Jim Tung, Markus Kuhl, Peter van Staa, "Education panel- designing the always connected car of the future", the Proceeding of DAC '10 Proceedings of the 47th Design Automation Conference, Pages 617-618, ACM New York, NY, USA ©2010.

[9] Simon Fürst, "Challenges in the Design of Automotive Software", the '10: Proceedings of the Conference on Design, Automation and Test in Europe, March 2010.

[10] Software Architecture 2 by MouradChabanneOussalah (Editor)ISBN: 978-1-84821-688-4 256 pages September 2014, Wiley-ISTE