Automotive Architecture Framework: the experience of Volvo Cars[☆]

Patrizio Pelliccione[a,*], Eric Knauss[a], Rogardt Heldal[a], Magnus Ågren[a], Piergiuseppe Mallozzi[a], Anders Alminger[b], Daniel Borgentun[b]

[a] *Chalmers University of Technology | University of Gothenburg, Department of Computer Science and Engineering, Sweden*
[b] *Volvo Cars, Sweden*

Abstract

The automotive domain is living an extremely challenging historical moment shocked by many emerging business and technological needs. Electrification, autonomous driving, and connected cars are some of the driving needs in this changing world. Increasingly, vehicles are becoming software-intensive complex systems and most of the innovation within the automotive industry is based on electronics and software. Modern vehicles can have over 100 Electronic Control Units (ECUs), which are small computers, together executing gigabytes of software. ECUs are connected to each other through several networks within the car, and the car is increasingly connected with the outside world. These novelties ask for a change on how the software is engineered and produced and for a disruptive renovation of the electrical and software architecture of the car. In this paper we describe the current investigation of the Volvo Cars to create an architecture framework able to cope with the complexity and needs of present and future vehicles.

Keywords: Architecture Framework, Software architecture, Automotive domain, Systems of Systems, Continuous Integration and Deployment, Automotive ecosystem

1. Introduction

Eric ►*just checking if github/sharelatex work as expected before working on plane.*◄ Today's automotive industry is much more than the traditional view of assembled mechanical parts controlled by a human for transportation. During the past twenty years vehicles have become

[☆]This work was partially supported by the NGEA and NGEA step 2 Vinnova projects and by the Wallenberg Autonomous Systems Program (WASP)

[*]Patrizio is the corresponding author.

Email addresses: patrizio.pelliccione@gu.se (Patrizio Pelliccione), eric.knauss@gu.se (Eric Knauss), heldal@chalmers.se (Rogardt Heldal), magnus.agren@chalmers.se (Magnus Ågren), mallozzi@chalmers.se (Piergiuseppe Mallozzi), anders.alminger@volvocars.com (Anders Alminger), daniel.borgentun@volvocars.com (Daniel Borgentun)

URL: www.patriziopelliccione.com (Patrizio Pelliccione)

more and more robot like, interpreting and exploiting input from various sensors to make decisions and finally commit actions that were previously made by humans.

Accordingly, the role of software is continuously changing. Initially, it was introduced in cars to optimise the control of the engines. Since then the growth of software within the car has been exponential for each year and today not a single function is performed without the involvement of software. 80% to 90% of the innovation within the automotive industry is based on electronics[1], as mentioned for instance by Peter van Staa - Vice-President Engineering of Robert Bosch GmbH at the European Technology Congress in June 2014[2]. A big part of electronics is software.

Considerable parts of the software is safety-critical, with human life at stake if the system is not performing as expected. Thus, the focus is gradually switching over from human control of the mechanics to software and electronics supporting decision-making and even taking over the control. The development is similar to the more popularized history of air plane development and the invention of auto pilots and "fly by wire", with some striking differences. The higher complexity of the environment where the vehicle moves has slowed down the development of automated vehicle control. The most advanced cars have more/comparable software than fighter airplanes[3] Vehicles are also produced in higher volumes than airplanes, which put harder requirements on the technology cost.

This evolution of the automotive industry creates new challenges regarding software architecture development and maintenance. The architecture of a modern car has to consider a large amount of concerns, including safety, security, variability management, networking, costs, weight, etc. Also, the increasing amount of people involved in the software development projects imposes additional challenges to the architecture teams, as the development and design literally cannot be controlled, or even understood, in detail by a single group any more. The development is inevitably parallelized; this obviously also holds for the large amounts of externally developed software, which is integrated as black box functionality. The important integration work is done in an iterative manner by developers and test teams, focusing on vital systems first and then gradually establishing various functionalities. Architects might be not involved in integration, however, the architecture for sure influences the integration.

In this paper we report a current initiative of the Volvo Cars to renovate the electrical architecture. The work is part of a Vinnova FFI Swedish project, called Next Generation Electrical Architecture (NGEA)[4], which mainly focuses on the following areas: (i) continuous integration and deployment; (ii) cars as constituents of a system of systems; (iii) reducing the number of ECUs through an architecture that allows the identification of key functions to be implemented in domain nodes; and (iv) strategies to improve the automotive ecosystem so

[1] http://www.etcwroclaw.eu/files/presentations/peter_van_staa.pdf

[2] http://www.etcwroclaw.eu/files/presentations/peter_van_staa.pdf

[3] As said by Alfred Katzenbach, the director of information technology management at Daimler: http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code

[4] http://www.vinnova.se/sv/Resultat/Projekt/Effekta/2009-02186/Next-generation-electrical-architecture/

to enable rapid communication with suppliers and flexible development. Preliminary results have been presented in the short paper [**?**].

Within the project we are investigating the possibility to create a Volvo Cars Architecture Framework [**?**]. We believe that an architecture framework, together with its multiple viewpoints, is the instrument to manage the increasing complexity of modern vehicles. It aims at ensuring that descriptions of vehicle architectures can be compared and related across different vehicle programs, development units and organizations, thus increasing flexibility and innovation, while reducing development time and risks.

We build on existing architecture frameworks in the automotive domain, i.e. [**? ?**] and we base our work on the conceptual foundations provided by the ISO/IEC/IEEE 42010:2011 standard [**?**]. The standard addresses architecture description, i.e. the practices of recording software, system and enterprise architectures so that architectures can be understood, documented, analysed and realized.

The paper is organized as follows. Section 2 gives an overview of the state of the art on architecture framework in the automotive domain. Section 4 analyses the state of practice in the context of Volvo Cars and reports some of the lessons learned. Section 5 describes the first steps towards the definition of an architecture framework for Volvo Cars. Finally the paper concludes in Section 9 with final remarks and a discussion about future work.

2. Architecture Frameworks

An architecture framework is a coordinated set of viewpoints, conventions, principles and practices for architecture description within a specific domain of application or community of stakeholders [**?**]. More specifically, it is determined by: (i) a set of architecture-related concerns, (i) a set of stakeholders holding those concerns, (ii) a set of architecture viewpoints which frame (i.e., cover) those concerns, and (iv) a set of model correspondence rules to impose constraints between types of models and then demonstrate that constraints are satisfied by the architecture. Then, an architecture framework establishes a common practice for creating, interpreting, analyzing and using architecture descriptions within a particular domain of application or stakeholder community, developing architecture modelling tools and architecting methods, and establishing processes to facilitate communication, commitments and interoperation across multiple projects and/or organizations [**?**].

Uses of architecture frameworks include, but are not limited to [**?**]:

- creating architecture descriptions;
- developing architecture modelling tools and architecting methods;
- establishing processes to facilitate communication, commitments and interoperation across multiple projects and/or organizations.

An architecture framework is a prefabricated knowledge structure, identified by *architecture viewpoints*, that architects use to organize an architecture description into *architecture views*. The terms architecture view and architecture viewpoint are central to the standard [**?**

]: "*A viewpoint is a way of looking at systems; a view is the result of applying a viewpoint to a particular system-of-interest*". An *architecture viewpoint* encapsulates notations, conventions, methods and techniques to be used according to specific model kinds framing particular concerns and for a particular audience of system stakeholders. The concerns determine what the model kinds must be able to express: e.g., functionality, security, reliability, cost, deployment, etc. A model kind determines the notations, conventions, methods and techniques to be used. Viewpoints, defining the contents of each architecture view, are built up from one or more model kinds and correspondence rules linking them together to maintain consistency. Viewpoints, like patterns and styles, are a form of reusable architectural knowledge for solving certain kinds of architecture description problems derived from best practices. Viewpoints originated in the 1970s (in Ross' Structured Analysis) and refined in [**?**]. Architecting methods often define one or more viewpoints, e.g. [**?** **?** **?** **?**].

Many existing practices express architectures through collections of models, and models are further organized into cohesive groups, called *views*. A view can be defined as a "*work product expressing the architecture of a system from the perspective of specific system concerns*" [**?**]. As noted in the standard, the cohesion of a group of models is determined by specific concerns, which are addressed by that group of models. Viewpoints refer to the conventions for expressing an architecture with respect to a set of concerns.

For further discussion of the content model and architecture frameworks mechanism, see [**?**]. Recent frameworks include the ISO Reference Model for Open Distributed Processing, GERAM (Generalized Enterprise Reference Architecture and Methodology) [**?**], DODAF (US Department of Defense Architecture Framework) [**?**], TOGAF [**?**], and MODAF [**?**]. For an extensive survey of frameworks, see [**?**].

In this paper we use the template called *Architecture Viewpoint Template* for specifying architecture viewpoints in accordance with ISO/IEC/IEEE 42010:2011, *Systems and software engineering—Architecture description*[5]

In the following we report the template sections and an excerpt of the guidelines; please refer to the links above for a complete description of the template:

- *Viewpoint Name*: name of the viewpoint. If there are any synonyms or other common names by which this viewpoint is known or used, record them here.
- *Overview*: Provide an abstract or brief overview of the viewpoint. Describe the viewpoint's key features.
- *Concerns and stakeholders*: Architects looking for an architecture viewpoint suitable for their purposes often use the identified concerns and typical stakeholders to guide them in their search. Therefore it is important (and required by the Standard) to document the concerns and stakeholders for which a viewpoint is intended.

[5]The template is called *Architecture Viewpoint Template*, which is released under copyright © 2012–2014 by Rich Hilliard. The template is licensed under a Creative Commons Attribution 3.0 Unported License. The terms of use are here: `http://creativecommons.org/licenses/by/3.0/`.

- **Concerns**: Describe each concern. Concerns name "areas of interest" in a system. Concerns may be very general (e.g., *Reliability*) or quite specific (*e.g., How does the system handle network latency?*). Concerns identified in this section are critical information for an architect because they help her decide when this viewpoint will be useful. When used in an architecture description, the viewpoint becomes a "contract" between the architect and stakeholders that these concerns will be addressed in the view resulting from this viewpoint. It can be helpful to express concerns *in the form of questions* that views resulting from that viewpoint will be able to answer. E.g., *How does the system manage faults?* or *What services does the system provide?*
- **Typical stakeholders**: Provide a listing of the typical stakeholders of a system who are in the potential audience for views of this kind. Typical stakeholders would include those likely to read such views and/or those who need to use the results of this view for another task.
- **Anti-concerns** [**Optional**] It may be helpful to architects and stakeholders to document the kinds of issues for which this viewpoint is *not appropriate or not particularly useful*. Identifying the "anti-concerns" of a given notation or approach may be a good antidote for certain overly used models and notations.

- *Model Kind name*[6]: Identify the model kind
  - *Conventions*: Describe the conventions for models of this kind. Conventions include languages, notations, modeling techniques, analytical methods and other operations. These are key modeling resources that the model kind makes available to architects and determine the vocabularies for constructing models of the kind and therefore, how those models are interpreted and used. The Standard does not prescribe *how* modeling conventions are to be documented. The conventions could be defined:

    I) by reference to an existing notation or language (such as SADT, UML or an architecture description language such as ArchiMate or SysML) or to an existing technique (such as $M/M/4$ queues);

    II) by presenting a metamodel defining its core constructs;

    III) via a template for users to fill in;

    IV) by some combination of these methods or in some other manner.
  - *Operations*: Specify operations defined on models of this kind.
  - *Correspondence rules*: Document any correspondence rules associated with the model kind.

[6]In the Standard, each architecture view consists of multiple architecture models. Each model is governed by a *model kind* which establishes the notations, conventions and rules for models of that type. Repeat the next section for each model kind listed here the viewpoint being specified.

- *Operations on views*: Operations define the methods to be applied to views and their models. Types of operations include:

  construction methods are the means by which views are constructed under this viewpoint. These operations could be in the form of process guidance (how to start, what to do next); or work product guidance (templates for views of this type). Construction techniques may also be heuristic: identifying styles, patterns, or other idioms to apply in the synthesis of the view.

  interpretation methods which guide readers to understanding and interpreting architecture views and their models.

  analysis methods are used to check, reason about, transform, predict, and evaluate architectural results from this view, including operations which refer to model correspondence rules.

  implementation methods are the means by which to design and build systems using this view.

- *Correspondence rules*: Document any correspondence rules defined by this viewpoint or its model kinds. Usually, these rules will be across models or across views since, constraints within a model kind will have been specified as part of the conventions of that model kind.

- *Examples [Optional]*: Provide helpful examples of use of the viewpoint for the reader (architects and other stakeholders).

- *Notes [Optional]* Provide any additional information that users of the viewpoint may need or find helpful.

- *Sources*: Identify sources for this architecture viewpoint, if any, including author, history, bibliographic references, prior art, etc.

3. State of the art in Automotive Architecture Frameworks

Architecture frameworks have not been standardized in the automotive domain and automotive industry. However, some attempts exist and different types of architecture viewpoints and views have been introduced recently as part of automotive architecture frameworks. A first attempt towards a standardized architectural foundation and automotive-specific architecture framework is the Automotive Architecture Framework (AAF) proposed in [**?**]. The purpose of AAF is to describe the entire vehicle system across all functional and engineering domains and drive the thought process within the automotive industry. The AAF conforms to the ISO 42010 international standard [**?**], and therefore it is defined in terms of a set of viewpoints and views.

The AAF distinguishes between two sets of architecture viewpoints and views: (i) mandatory and general viewpoints and (ii) optional viewpoints. The following viewpoints are presented according to the viewpoint catalog in [**?**]. The mandatory viewpoints and their

respective views include: (i) Functional viewpoint - functional decomposition, functional architecture; (ii) Logical viewpoint - logical decomposition, logical architecture; this viewpoint is only mentioned in [**?**] but not detailed in [**?**]; (iii) Technical viewpoint - physical decomposition, technical architecture; (iv) Information viewpoint - perspective of information or data objects used to define and manage a vehicle; (v) Driver/vehicle operations viewpoint - vehicle environment; (vi) Value net viewpoint - OEM stakeholders and those of its suppliers and engineering partners. The optional viewpoints suggested by the AAF are: (i) Safety, (ii) Security, (iii) Quality and RAS - Reliability, Availability, Serviceability, (iv) Energy, possibly including performance, (v) Cost, (vi) NVH - noise, vibration, harshness, and (vii) Weight.

3.1. Architectural Design Framework

The Architectural Design Framework (ADF) [**?**] is developed by Renault and includes operational, functional, constructional, and requirements viewpoints. Although the AAF and ADF are related they have some differences.

- *Operational Viewpoint* - this is the more abstract viewpoint. The system is observed from a black box and user perspective [**?**].
- *Functional Viewpoint* - system functions identified in the views associated to the operational viewpoint are allocated to SysML Blocks.
- *Constructional Viewpoint* - this viewpoint describes a further allocation (or grouping) of system functions into physical components.
- *Requirements Viewpoint* - This viewpoint is orthogonal to other viewpoints. Each requirement view has a relationship with views of other viewpoints.

3.2. Architecture Framework For Automotive Systems

The Architecture Framework for Automotive Systems (AFAS) is proposed in [**?**] through an analysis of AAF, ADF and of Architecture Description Languages [**?** **?**] tailored to automotive domain, like EAST-ADL [**?**] and AML [**?**]. It contains an elaboration and unification of the viewpoints proposed in AAF and ADF and then proposes additional viewpoints, e.g.:

- *Feature viewpoint* - to be used to support the product line engineering.
- *Implementation viewpoint* - it describes the software architecture of the Electrical/Electronic (E/E) system in the vehicle [**?**].

4. Software development challenges within Volvo Cars

In a previous paper we made a study within Volvo Cars to identify, from the architecture point of view, the challenges that OEMs are facing in the last years [**?**]. To better understand some of the organizational issues with having different parts of the organization responsible for different parts or layers in the architecture, we decided to conduct nine in depth interviews with focus on the roles of architects and how organizational factors affect them. These interviews have been carried out at Volvo Cars and Volvo Group Truck Technology (VGTT), extending the knowledge about these specific companies, hence providing a detailed understanding of two independent but nevertheless similar automotive companies. In [**?**] we followed a lightweight grounded theory-type approach since we started from a general research question and we involved few people in order to collect more information. Based on the analysis of the first data and on the first emerging ideas from the data, we then refined the research questions, carefully planned the study and accordingly selected the people to be interviewed. In addition to initial workshops in which we collected initial ideas, we used interviews as data generation methods. Specifically, we used semi-structured interviews: we defined a list of themes to be covered and questions we aimed at asking. However, in some interviews we changed the order of questions according to received answers and to the flow of the "conversation". Each interview was around one hour long, we collected field notes and recorded audio. Each interview begins with introduction, clarification about the purpose of the study, asking permission to record and giving assurances of confidentiality of the information.

Moreover, in these years of strong collaboration with Volvo Cars and thanks also to the industrial co-authors (the last two authors of this paper) we had the possibility to further improve the knowledge of to collect a set of insights, as described in the following subsections.

4.1. Gap between prescriptive and descriptive architecture

We identified that there is not always an obvious connection between architecture (or top-level design) and design; it seems also that this connection vanishes during time, once development requires changes on the system design. The architecture is communicated as large documents, which are supposed to be read by stakeholders. However, this not always corresponds to the reality. The companies work also in cross functional teams and other type of communications are also used to communicate the architecture. More analysis is needed to understand the root of the problem. Moreover, maintenance of the architecture, while the design evolves, is demanding and not always performed in all parts. This shows a discrepancy between the planned architecture defined according to a V-Model process, and the architecture that is actually emerging from the system development.

We identified that the architecture has a temporal aspect, that means: at any given point in time the system has only one architecture, however, the architecture will change over time. We call as *prescriptive architecture* the architecture that captures the design decisions made prior to the system's construction. This architecture is the as-conceived or as-intended architecture. We can call as *descriptive architecture* the architecture that describes how the system has been built. This architecture describes the as-implemented or as-realized

architecture. We observed that there is often a discrepancy between the as-intended and the as-implemented architecture. This causes what is called *architecture degradation*. The degradation might shows up in two different ways: (i) *architectural drift* when the descriptive architecture includes changes that are not included in, encompassed by, or implied by the prescriptive architecture, but which do not violate any of the prescriptive architecture's design decisions; (ii) *architectural erosion* is the introduction of architectural design decisions into a system's descriptive architecture that violate its prescriptive architecture.

When looking at the causes of architecture degradation, we can summarize the various reasons in the following three items:

1. During the development some important directives of the as-intended architecture have been violated due to time constraints, mistakes, misunderstanding, etc.
2. Some of the architectural choices of the as-intended architecture were based on assumptions (might be implicit) that then were identified as imprecise or wrong,
3. Some of the architectural choices of the as-intended architecture were made under uncertainty and during development these choices were judged as suboptimal.

4.2. Organization and process challenges

The first item of the causes of architecture degradation discussed in the section above (Section 4.1) might be solved by improving communication as mentioned before or by finding ways to guarantee the preservation of the important directives of the as-intended architecture that should not be violated for any reason.

On the organizational side, we found a need to improve the communication between different groups, for instance by making teams more cross-functional. Today there are several levels between architects and designers/developers and at some of these levels the connection is not very tight.

On one side this seems to be unavoidable since the company is big and there is the need of structuring the organization in sub-organizations (departments, groups, or teams). However, the risk is that sub-organizations will grow up independently and decoupled from other sub-organizations and this might stimulate a silos thinking. For instance, this might lead designers/developers to think that the architects are sitting in their cloud above, without having any connection to the reality. Architects might also feel a frustration because they are not aware of everything that is happening; as a consequence, a big part of their work is to just keep up with what is happening in the construction groups. Espousing the terminology in [**?** **?**], architects should be first of all "Extrovert" architects (conceptually related to the external focus of [**?**]), i.e. devoted to communicating the architectural decisions and knowledge to the other stakeholders.

The different organizations have different competencies, attitude, characteristics. However, new problems emerge. In fact, we can confirm that we found many architecture antipatterns. We found the *Goldplating* antipattern [**?**] since architects seem to be not really engaged with developers. They are doing a good technical job, however, their output is not really aligned with the needs of the developers and in the end they are often ignored. Another antipattern that we found is the *Ivory tower* [**?**]: the architecture team looks

isolated sitting on a separate floor from the development groups and do not engage with the developers and the other stakeholders on a daily basis. This creates tensions in the organization.

It emerges the need to explore both organizational and technical possibilities for tighter cooperation between architecture levels, and to measure effects such improvements would have. On the technical side, one partial solution might be to define a framework able to automatically generate high-level views from the design. The challenge here is to support multiple views, each devoted to showing only what is relevant for respective stakeholders. Moreover, both architects and designers/developers need ways to perform early validation of their solution and to sketch and try different visions of how the future systems should look like, to understand the effect of design decisions affect the architecture. However, this solution cannot solve the architecting problem since this solution only focus to create a "different view" for something that already exist. Other solutions are needed to solve the support just-in-time architecting as mentioned above, as well as to enable stakeholders different from the architects, such as developers, to improve the architecture when actually needed.

4.3. Towards "just-in-time architecture"

The second and third items of the causes of architecture degradation discussed in Section 4.1 call for a "*just-in-time architecture*" or agile architecting [**?**] as well as to enable stakeholders different from the architects, such as developers, to improve the architecture, such as fixing wrong assumptions or making decision deliberately postponed.

One hypothesis made from some practitioners is that when developing large and complex systems "a clear and well defined architecture facilitates and enables agility". This hypothesis implies that some upfront specification is needed when building complex products like cars. However this hypothesis is not completely true when the product to be realized is not clearly defined and companies want to go fast to the market. In these situations, modifiability, support for evolution, etc. are not really main aspects to be considered. The hypothesis seems to be true when the product is well defined. Another aspect to be considered is that agile calls for refactoring, however refactoring often is not performed since the priority is given to what should be realized. Further investigation is needed, however the conclusion we can draw at this point is that there is the need of a "just-in-time architecture" that enables even stakeholders that are different from the architects, such as developers, to improve the architecture, such as fixing wrong assumptions or making decisions deliberately postponed by the architects.

4.4. Towards a Software Ecosystem Perspective

Another interesting finding is that the architecture is not clearly considering the highly complex supplier-network that characterizes automotive engineering.

In a previous study [**? ?**] we found that a holistic strategy for aligning work across the value-chain is currently missing. Specifically, mixing commodity and differentiating components lead to sub-optimal situations, resulting in duplicated work. We argue that

automotive architecture needs to assume a holistic perspective with respect to the whole value-chain and optimize the architecture for facilitating beneficial subcontractor interaction.

- *Commodity Components* require clearly defined technical and organizational interfaces. The goal is to develop them as efficiently as possible, thus reducing coordination overhead. Ideally, of-the-shelf commodity components can be integrated with minimal adjustment.

- *Differentiating Components* should be developed as independent from the commodity components as possible, probably in-house.

- *Innovative Components* naturally require coordination and iterative work between a number of partners. To effectively develop innovative behaviour, could communication channels need to be established.

In traditional software engineering, a software product is often the result of an effort of a single independent software vendor, investing into creating a monolithic product [**?** **?**]. Modern software engineering strongly relies on components and infrastructure from third-party vendors or open source suppliers [**?** **?**]. The emergence of Software Ecosystems (SECOs) is a recent development within the software industry [**?** **?**]. It implies a shift from closed-organizations to open structures where external actors become involved in the development to create competitive value [**?** **?**].

Based on the ecosystem classification model [**?**], we understand the automotive value chain as an ecosystem of cross-organizational collaborations among automotive suppliers [**?**]. In this ecosystem, the Original Equipment Manufacturer (OEM) is in the role of the ecosystem coordinator. The ecosystem is privately owned, and participation to it is based on a list of screened actors.

The automotive ecosystem is characterized by relying heavily on complex supplier networks, and strong dependence on hardware and software development [**?**]. Due to the increasing number of networked components, a level of complexity has been reached which is difficult to handle using traditional development processes [**?**]. The automotive industry addresses this problem through a paradigm shift from a hardware-, component-driven to a requirement- and function-driven development process, and a stringent standardization of infrastructure elements. One central standardization initiative is the AUTomotive Open System ARchitecture (AUTOSAR) [**?**]. The principal aim of the AUTOSAR standard is to master the growing complexity of automotive electronic architectures [**?**]. We refer to the *AUTOSAR ecosystem* as a subset of the *automotive ecosystem*, where different actors participate in value creation (i.e. development of software components) by exchanging products and services based on a technical platform defined by the AUTOSAR standard.

Several challenging areas, including requirements engineering, are reported in the automotive domain [**?**]. OEMs and suppliers need to communicate requirements based on the requirements documents, which are imprecise and incomplete nowadays [**?**]. In this regard, [**?**] reports that the requirements value chain is little understood beyond software projects.

It is unclear which requirements communication, collaboration, and decision-making principles lead to efficient, value-creating and sustainable alignment of interests between interdependent stakeholders across software projects and products [**?**]. This however is important, because the way the interests and expectations of stakeholders of SECOs are communicated is critical for whether they are heard, hence whether the stakeholders are successful in influencing future solutions to meet their needs [**?**]. We note that in the automotive domain, communication, collaboration, and decision-making are cross-organizational challenges related to the requirements viewpoint and requirements engineering.

5. Volvo Cars architecture framework

The starting point for defining an architecture framework is to start from the identification of established stakeholders within the domain of the framework. Stakeholders may be individuals, teams, organizations or classes (of individuals, teams or organizations), while concerns may be fine-grained or very broad in scope [**?**].

Table 1 describes the main stakeholders we have identified; they fall into five major groups:

- *End-users* of the electrical system, like drivers and passengers.
- *Customers* stakeholders, such as paying customers of products and services that depend on the electrical system (i.e. the car) and product planners, who acquire the electrical system as part of the overall product.
- *Management* with responsibility for scheduling, long term quality, groups, departments, and budget.
- *Developers of the electrical system* include engineers throughout the value chain that create the electrical system, its architecture, and the necessary tools as well as that test and integrate the various components.
- *Maintainers of the electrical system* who interact with the electrical system throughout its lifetime.

Then the identified stakeholders motivate the set of concerns on which the architecture framework will focus. This will help the consumer of the architecture framework and of the views and connected modeling tools to understand why they are modeling and when they are done. In order to define the stakeholders concerns we identified a set of challenging scenarios through a number of workshops for elicitation and validation. Figure 1 gives an overview of the scenarios that are strongly connected to the viewpoints we will detail in this paper. These scenarios are described in the following items. **Patrizio** ►*Ensure that there is a mapping between these scenarios and the viewpoints we will discuss. Indeed we need to consider also SoS from WP3*◄ **Magnus** ►*Rogardt and I made a draft picture with the use cases from WP3 added. Color, exact naming, bubble placement etc., TDB. Do you think this is a workable way forward?*◄

Table 1: Overview of Stakeholders

	Stakeholder	Group	Comment	Synonyms
•	Passengers	end-user		
•	Drivers	end-user		
•	Customers	customer	Purchaser of a car or related service	
•	Product planner	customer	Acquirer of electrical system	
•	Purchaser	customer	Purchasers of electrical system	
•	Line managers	management	Has scheduling responsibility, long term quality responsibility, includes group, department	
•	Project managers	management	Owns budget for development	
•	System architects	developers of electrical system		
•	Functional developers	developers of electrical system	Owns functional and non-functional aspects	function owner; function realizer; function developer, function realizer, system developer
•	Component developers	developers of electrical system		
•	SW supplier (internal/external)	developers of electrical system	Can be internal or external from the perspective of the OEM.	
•	HW supplier (internal/external)	developers of electrical system	Can be internal or external from the perspective of the OEM.	
•	Testers	developers of electrical system		
•	Attribute Owners	developers of electrical system	Owns non-functional attributes like performance	
•	Tool Engineers	developers of electrical system	Specifically testing tools, including design tools (e.g. for requirements)	
•	Calibrators	developers of electrical system		
•	Diagnostic method engineers	maintainers of electrical system		
•	Workshop Personnel	maintainers of electrical system		
•	Fault Tracing Specialists	maintainers of electrical system		
•	Technical Specialist	specialists	Support developers and maintainers on specific topics	

- Scenario 2.1 **decision management** aims at exploring how to make, communicate, and manage decisions.

  User Story: *"As a member of the development ecosystem I would like to have a clear understanding on how to take decisions and how to communicate them."*

  Interesting sub-scenarios include decisions about:

  – *Requirements (2.1.1)*: When decisions on requirements are made too early, it will lead to unnecessary changes.

    User Story: *"As a component responsible, I need to write a requirement. Currently, I am forced to write the requirement in a certain document*

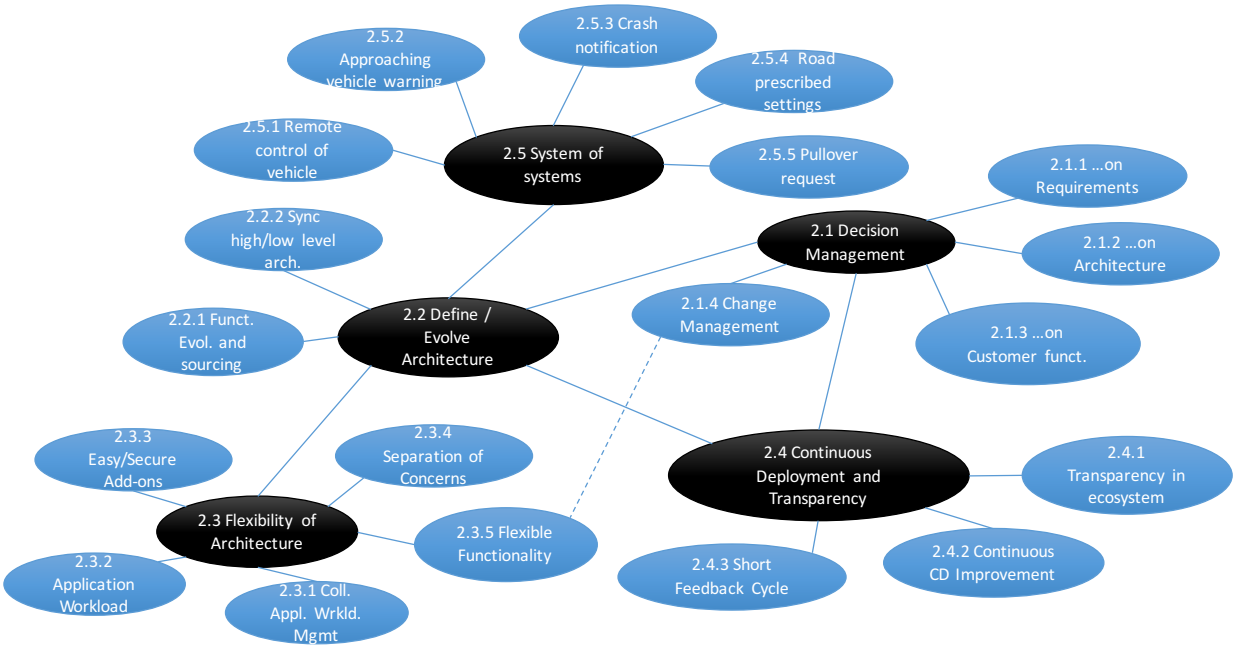

Figure 1: Overview of challenging scenarios

or certain structure. This determines whether the requirement refers to a hw or sw component, whether it will be implemented in-house or by an external supplier. Often, it is too early to make such decisions and changes are necessary later, when more information becomes available. I do not feel comfortable to make this decision so early."

– *Architecture (2.1.2)*: Architectural decision making involves making the right decision, communicating it, ensuring that it is followed, and changing it when needed.

  User Story: *"As an architect* (one of {system architect; functional developer; low level architect}), *I need to make the right decision at the right moment (i.e. when it is useful to make the decision and when the necessary information is available). I need to make this decision on the right level. I need to be introvert to make the best possible decision on the available data and extrovert to communicate it."*

– *Customer functions (2.1.3)*: Electrical architecture is guiding realization of customer functions, but it is not obvious how the architecture support customer functions (with respect to tracing and methodology).

  User Story: *"As a product manager, I want to be supported by the electrical architecture in making decisions about customer functions. Based*

on a wishlist from the Market Analysis Department, I engage in a dialog with the departments that design the system. For this task, I wish for support from an electrical architecture that not only takes into account non-functional aspects, but also the nature of customer functions (which changes over time).”

- *Change management (2.1.4)*: Good flexibility of the architecture allows us to continuously remove assumptions and do changes even late in the process. However, in a weak electrical architecture, such changes will impact the stability of the electrical system because of technical dependencies.

  User Story: *“As a product planner I want to have a flexible Change Management process, allowing me to change or add functions late in the process, often with the goal of removing assumptions. This includes for example defining and changing the allocation of Functions to ECU late in the process.”*

- Scenario 2.2 **define/evolve architecture** explores aspects of long-time evolution of the electrical architecture. This includes:

  - *The impact of long-term sourcing decision on the logical architecture (2.2.1)*: Long term sourcing contracts allow to optimize production cost, but constraint evolution of the architecture.

    User Story: *“As a Function Developer I want to optimise the evolution of functions without considering sourcing agreements. Today I have to discuss with the product planner about the plan for a function two years or more in advance to reflect on existing contracts, e.g. when related nodes are sourced for different time intervals. Problem: Sourcing needs make physical layout dominate logical decisions.”*

  - *The danger of architecture and design evolving in different directions (2.2.2)*: If not actively managed, architecture and design diverge over time. The architecture is then perceived as outdated and not useful, thus it looses its ability to guide design decisions and implementation.

    User Story: *“As a system architect or function developer I want a stringent correlation between architecture and design. Otherwise, one or the other is wrong. ”*

- Scenario 2.3 **flexibility of architecture** addresses different scenarios that emphasize flexibility of the electrical architecture, both on a technical and on a process level. On a technical level, more capacity in the ECUs allows to add functionality late. On a process level, the available resources need to be managed across several contributing partners. This includes:

  - *application workload management from a process perspective (2.3.1)*:

User Story: *"As a functional developer, I want to be flexible when it comes to application workload. Suppliers should be able to acquire and return resources dynamically throughout the lifecycle. "*

- *application workload management from a technical perspective (2.3.2)*: higher capacity of ECUs and Buses will facilitate late or even very late updates (i.e. when the vehicle is on the street), because new functionality could use more resources. This flexibility comes at a cost and it is not trivial to understand where the break-even point is.

  User Story: *"As a system architect I want to balance capacity of ECUs and Buses against cost."*

- *easy and secure add-ons (2.3.3)*: being able to add new functions in a secure and easy way would allow to detach software development to some extend from the development cycle. During the development of a car, an OEM could focus on the critical basic functionality. Convenience features as well as more advanced connected features could be added independent from the start of production. This would allow to develop more continuously and should decrease the peak of trouble reports before critical deadlines observed earlier [**?**].

  User Story: *"As a functional developer I want to be able to add new functions (very) late. Such add-ons could originate from third parties and should be added even after the vehicle has been put into use. "*

- *separation of concerns (2.3.4)*:

  User Story: *"As a system architect, I want to balance separation of concerns on two levels: between domains (e.g. safety vs. infotainment) and levels of abstractions (e.g. architecture vision vs architecture implementation)."*

- *flexible functionality (2.3.5)*:

  User Story: *"As a functional developer, I want to be flexible about the functions that are running in the car and allow for very late deployment. "*

- Scenario 2.4 **continuous deployment and transparency** includes:

  - *the need for openness and transparent information through out the different value chains in the automotive ecosystem (2.4.1)*: Good transparency in the value chain supports flexibility, since all partners can take appropriate action to reach a (changing) goal quickly. The *cone of uncertainty* is a good example for this [**? ?**]: In the beginning of a project, not much data is available and decisions are very uncertain. As the project proceeds, assumptions are removed and uncertainty is reduced, but as a consequence, it becomes harder to make decisions that change a lot. Good transparency can help to reduce the uncertainty quicker and can allow to decide fast in order to learn fast. m

User Story: *"As any developer of the electrical system (internal or external), I want to have access to relevant decision, to the status of relevant assumptions, and knowledge generated during the development. This allows me to participate in the fast learning throughout the value chain and enables me to be effective and flexible in my work."*

- *the need for using this information to continuously improve an continuous integration, delivery, and deployment flow (2.4.2)*: Cross-organizational continuous integration, delivery, and deployment facilitates fast feedback and rich learning. This learning should also help to improve the interaction of organizations and the integration flow between them.

  Have a culture of continuous improvement between VCC and partners Apply Continuous Deployment to the CD tool chain

  User Story: *"As a Project Manager, I want to have a culture of continuous improvement within the OEM and with partners in the value-chain. As any developer of the electrical system (internal and external) I want to benefit from and take responsibility in improving the continuous delivery flow."*

- *the need for establishing short feedback cycles (2.4.3)*: While developing new functionality, basic software, and hardware, one should plan on how to receive feedback, which data to collect, and how to use it in the development.

  User Story: *"As any developer of the electrical system (internal or external) I want to have quick feedback on how my contribution will work on the various levels of integration. As a Functional Developer, I want to have fast and defined feedback cycles. As a tester, I want quick updates on all levels of tests and continuous improvement of functionality."*

- Scenario 2.5 **System of Systems** includes:
  - *Communication*
  - *Interoperability*
  - *Degree of autonomy and readiness to be at the service of the SoS*
  - *Dealing with uncertainty and functional safety*
  - *Cyber security and privacy*

The identified architecture-related concerns determine the choice of viewpoints and view to be included. It is important to note that almost each viewpoint detained in the following is already handled within the current architecture of Volvo Cars. However, we are investigating the definition of proper viewpoints that will create the architecture framework. **Patrizio** ▶*here we have to explain that there are the viewpoints in the litareture, other important viewpoints for Volvo Cars, but that in this paper, due to space restrictions we only focus on three viewpoints*◀ Therefore, in addition to the viewpoints summarized in Section 2 we foresee the following viewpoints:

- *Continuous integration and deployment* (detailed in Section **??**) - OEMs are increasingly interested to reduce the development time, to increase flexibility, to have early feedback on decisions made, and to add new functionalities incrementally even after production.
- *Connected cars and safety* (detailed in Section **??**) - Future scenarios of collaborating autonomous vehicles, like platooning, will require to extend the vehicle safety architecture across the classical boundaries of single vehicles and will ask for an open and adaptive architecture able to support runtime assessment of safety.
- *Security and privacy of connected cars* - Connected cars open new important challenges from the point of view of security and privacy.
- *Ecosystem and transparency* - related to the value net viewpoint of AAF [**?** **?**]. The ecosystem around the OEM can be seen as a virtual organization consisting of the OEM, its suppliers and other partners involved in the process of creating customer value.
- *Autonomous cars* - autonomous cars require special architecture solutions, e.g. inspired to autonomous and self-adaptive systems [**?**].
- *Modes management* - a mode viewpoint is needed to design the different modes of a vehicle as well as the transitions from one mode to another.
- Special viewpoints and views might be conceived to enable dissemination and communication of the architecture to developers and other stakeholders.

The natural consequence of the use of multiple viewpoints and views in architecture descriptions is the need to express correspondences and consistency rules between those views. The mechanisms introduced in [**?**] is called model correspondences and it allows the definition of relations between two or more architecture models. Since architecture views are composed of architecture models [**?**], model correspondences can be used to relate views to express consistency, traceability, refinement or other dependencies [**?**]. These mechanisms allow an architect to impose constraints between types of models and then demonstrate that them are satisfied by the architecture.

6. Continuous Integration and Deployment viewpoint

6.1. Continuous Integration and Deployment

⋆ Provide the name for the viewpoint.

If there are any synonyms or other common names by which this viewpoint is known or used, record them here.

6.2. Overview

Agile approaches and practices such as continuous integration and deployment promise to help reducing development time, to increase flexibility, and to generally shorten the feedback cycle time. However, the complex supplier network, and typical setup with a large number of ECUs, pose specific challenges to these practices.

First, dependencies between ECUs raise multiple concerns, regarding organization, versioning and testing: (i) organization - identifying the recipient of a given software change; (ii) versioning - the question is related to the compatibility of the software version of specific ECUs; and (iii) testing - compatible combinations need to be validated. Second, support for continuous deployment has to face with safety concerns. Should, for instance, the software of an ECU responsible for a safety critical function be modifiable at runtime?

Dependencies between ECUs are a property of the architecture. As mentioned, the emergent architecture may differ from the intended architecture, and continuous integration and deployment of software may entail architectural changes. This highlights both the need for collaboration between parts of the organization working on different architectural levels, and the need of a proper support for agile and flexible architecting.

Addressing these concerns suggests two architectural views and viewpoints: (i) one covering architecture as an enabler of continuous integration and deployment, facilitating variant handling and coordination of updates, and (ii) another considering continuous integration and deployment on the architecture level, facilitating reasoning about modifications to the architecture itself.

6.3. Concerns and stakeholders

Architects looking for an architecture viewpoint suitable for their purposes often use the identified concerns and typical stakeholders to guide them in their search. Therefore it is important (and required by the Standard) to document the concerns and stakeholders for which a viewpoint is intended.

6.3.1. Concerns

- How can we reduce the number of architectural assumptions?

The more of the architecture one make up front the more assumptions are needed to be added. It is hard to guarantee that these assumptions are true, moreover, if there are too many assumption one might loose the overviews as well.

- How can we avoid building the wrong architecture?

It is no use in making a good architecture if it is for the wrong purpose.

- How can we reduce the number of architectural assumptions?
- How can we avoid building the wrong architecture?
- How can we respond quicker to change in the market?

- How can we deal with changing interfaces

⋆ Provide a listing of architecture-relevant concerns to be framed by this architecture viewpoint per ISO/IEC/IEEE 42010, 7a.

Describe each concern.

Concerns name "areas of interest" in a system.

NOTE: *Following ISO/IEC/IEEE 42010,* ***system*** *is a shorthand for any number of things including man-made systems, software products and services, and software-intensive systems such as "individual applications, systems in the traditional sense, subsystems, systems of systems, product lines, product families, whole enterprises, and other aggregations of interest".*

Concerns may be very general (e.g., *Reliability*) or quite specific (*e.g., How does the system handle network latency?*).

Concerns identified in this section are critical information for an architect because they help her decide when this viewpoint will be useful.

When used in an architecture description, the viewpoint becomes a "contract" between the architect and stakeholders that these concerns will be addressed in the view resulting from this viewpoint.

It can be helpful to express concerns *in the form of questions* that views resulting from that viewpoint will be able to answer. E.g.,

- *How does the system manage faults?*
- *What services does the system provide?*

NOTE: *"In the form of a question" is inspired by the television quiz show, Jeopardy!*

ISO/IEC/IEEE 42010, 5.3 contains a candidate list of concerns that must be considered when producing an architecture description. These can be considered here for their relevance to the viewpoint being specified:

- What are the purpose(s) of the system-of-interest?
- What is the suitability of the architecture for achieving the system-of-interest's purpose(s)?
- How feasible is it to construct and deploy the system-of-interest?
- What are the potential risks and impacts of the system-of-interest to its stakeholders throughout its life cycle?
- How is the system-of-interest to be maintained and evolved?

See also: ISO/IEC/IEEE 42010, 4.2.3.

6.3.2. Typical stakeholders

⋆ Provide a listing of the typical stakeholders of a system who are in the potential audience for views of this kind, per ISO/IEC/IEEE 42010, 7b.

Typical stakeholders would include those likely to read such views and/or those who need to use the results of this view for another task.

Stakeholders to consider include:

- users of a system;
- operators of a system;
- acquirers of a system;
- owners of a system;
- suppliers of a system;
- developers of a system;
- builders of a system;
- maintainers of a system.

6.3.3. "Anti-concerns" (optional)

It may be helpful to architects and stakeholders to document the kinds of issues for which this viewpoint is *not appropriate or not particularly useful*.

Identifying the "anti-concerns" of a given notation or approach may be a good antidote for certain overly used models and notations.

6.4. Model kinds+

⋆ Identify each model kind used in the viewpoint per ISO/IEC/IEEE 42010, 7c.

In the Standard, each architecture view consists of multiple architecture models. Each model is governed by a *model kind* which establishes the notations, conventions and rules for models of that type. See: ISO/IEC/IEEE 42010, 4.2.5, 5.5 and 5.6.

Repeat the next section for each model kind listed here the viewpoint being specified.

6.5. Continuous Integration and Deployment

⋆ Identify the model kind.

6.5.1. Continuous Integration and Deployment conventions

⋆ Describe the conventions for models of this kind.

Conventions include languages, notations, modeling techniques, analytical methods and other operations. These are key modeling resources that the model kind makes available to architects and determine the vocabularies for constructing models of the kind and therefore, how those models are interpreted and used.

It can be useful to separate these conventions into a *language part*: in terms of a metamodel or specification of notation to be used and a *process part*: to describe modeling techniques used to create the models and methods which can be used on the models that result. These include operations on models of the model kind.

The remainder of this section focuses on the language part. The next section focuses on the process part.

The Standard does not prescribe *how* modeling conventions are to be documented. The conventions could be defined:

I) by reference to an existing notation or language (such as SADT, UML or an architecture description language such as ArchiMate or SysML) or to an existing technique (such as $M/M/4$ queues);

II) by presenting a metamodel defining its core constructs;

III) via a template for users to fill in;

IV) by some combination of these methods or in some other manner.

Further guidance on methods I) through III) is provided below.

Sometimes conventions are applicable across more than one model kind – it is not necessary to provide a separate set of conventions, a metamodel, notations, or operations for each, when a single specification is adequate.

I) Model kind languages or notations (optional)

Identify or define the notation used in models of the kind.

Identify an existing notation or model language or define one that can be used for models of this model kind. Describe its syntax, semantics, tool support, as needed.

II) Model kind metamodel (optional)

A metamodel presents the AD elements that constitute the vocabulary of a model kind, and their rules of combination. There are different ways of representing metamodels (such as UML class diagrams, OWL, eCore). The metamodel should present:

entities What are the major sorts of conceptual elements that are present in models of this kind?

attributes What properties do entities possess in models of this kind?

relationships What relations are defined among entities in models of this kind?

constraints What constraints are there on entities, attributes and/or relationships and their combinations in models of this kind?

NOTE: *Metamodel constraints should not be confused with architecture constraints that apply to the subject being modeled, not the notations used.*

In the terms of the Standard, entities, attributes, relationships are *AD elements* per ISO/IEC/IEEE 42010, 3.4 4.2.5 and 5.7.

In the *Views-and-Beyond* approach [**?**], each viewtype (which is similar to a viewpoint) is specified by a set of elements, properties, and relations (which correspond to entities, attributes and relationships here, respectively).

When a viewpoint specifies multiple model kinds it can be useful to specify a single viewpoint metamodel unifying the definition of the model kinds and the expression of correspondence rules. When defining an architecture framework, it may be helpful to use a single metamodel to express multiple, related viewpoints and model kinds.

III) Model kind templates (optional)

Provide a template or form specifying the format and/or content of models of this model kind.

6.5.2. Continuous Integration and Deployment operations (optional)

Specify operations defined on models of this kind.

See §8.5 for further guidance.

6.5.3. Continuous Integration and Deployment correspondence rules

⋆ Document any correspondence rules associated with the model kind.

See §8.6 for further guidance.

6.6. Operations on views

Operations define the methods to be applied to views and their models. Types of operations include:

construction methods are the means by which views are constructed under this viewpoint. These operations could be in the form of process guidance (how to start, what to do next); or work product guidance (templates for views of this type). Construction techniques may also be heuristic: identifying styles, patterns, or other idioms to apply in the synthesis of the view.

interpretation methods which guide readers to understanding and interpreting architecture views and their models.

analysis methods are used to check, reason about, transform, predict, and evaluate architectural results from this view, including operations which refer to model correspondence rules.

implementation methods are the means by which to design and build systems using this view.

Another approach to categorizing operations is from Finkelstein et al. [**?**]. The *work plan* for a viewpoint defines 4 kinds of actions (on the view representations): *assembly actions* which contains the actions available to the developer to build a specification; *check actions* which contains the actions available to the developer to check the consistency of the specification; *viewpoint actions* which create new viewpoints as development proceeds; *guide actions* which provide the developer with guidance on what to do and when.

6.7. Correspondence rules

⋆ Document any correspondence rules defined by this viewpoint or its model kinds.

Usually, these rules will be across models or across views since, constraints within a model kind will have been specified as part of the conventions of that model kind.

See: ISO/IEC/IEEE 42010, 4.2.6 and 5.7

6.8. Examples (optional)

Provide helpful examples of use of the viewpoint for the reader (architects and other stakeholders).

6.9. Notes (optional)

Provide any additional information that users of the viewpoint may need or find helpful.

6.10. Sources

⋆ Identify sources for this architecture viewpoint, if any, including author, history, bibliographic references, prior art, per ISO/IEC/IEEE 42010, 7e.

7. Ecosystem and transparency viewpoint

7.1. Ecosystem and transparency

⋆ Provide the name for the viewpoint.

If there are any synonyms or other common names by which this viewpoint is known or used, record them here.

7.2. Overview

Provide an abstract or brief overview of the viewpoint.

Describe the viewpoint's key features.

7.3. Concerns and stakeholders

Architects looking for an architecture viewpoint suitable for their purposes often use the identified concerns and typical stakeholders to guide them in their search. Therefore it is important (and required by the Standard) to document the concerns and stakeholders for which a viewpoint is intended.

7.3.1. Concerns

⋆ Provide a listing of architecture-relevant concerns to be framed by this architecture viewpoint per ISO/IEC/IEEE 42010, 7a.

Describe each concern.

Concerns name "areas of interest" in a system.

NOTE: *Following ISO/IEC/IEEE 42010,* ***system*** *is a shorthand for any number of things including man-made systems, software products and services, and software-intensive systems such as "individual applications, systems in the traditional sense, subsystems, systems of systems, product lines, product families, whole enterprises, and other aggregations of interest".*

Concerns may be very general (e.g., *Reliability*) or quite specific (*e.g., How does the system handle network latency?*).

Concerns identified in this section are critical information for an architect because they help her decide when this viewpoint will be useful.

When used in an architecture description, the viewpoint becomes a "contract" between the architect and stakeholders that these concerns will be addressed in the view resulting from this viewpoint.

It can be helpful to express concerns *in the form of questions* that views resulting from that viewpoint will be able to answer. E.g.,

- *How does the system manage faults?*
- *What services does the system provide?*

NOTE: *"In the form of a question" is inspired by the television quiz show, Jeopardy!*

ISO/IEC/IEEE 42010, 5.3 contains a candidate list of concerns that must be considered when producing an architecture description. These can be considered here for their relevance to the viewpoint being specified:

- What are the purpose(s) of the system-of-interest?
- What is the suitability of the architecture for achieving the system-of-interest's purpose(s)?
- How feasible is it to construct and deploy the system-of-interest?
- What are the potential risks and impacts of the system-of-interest to its stakeholders throughout its life cycle?
- How is the system-of-interest to be maintained and evolved?

See also: ISO/IEC/IEEE 42010, 4.2.3.

7.3.2. Typical stakeholders

⋆ Provide a listing of the typical stakeholders of a system who are in the potential audience for views of this kind, per ISO/IEC/IEEE 42010, 7b.

Typical stakeholders would include those likely to read such views and/or those who need to use the results of this view for another task.

Stakeholders to consider include:

- users of a system;
- operators of a system;
- acquirers of a system;
- owners of a system;
- suppliers of a system;
- developers of a system;
- builders of a system;
- maintainers of a system.

7.3.3. "Anti-concerns" (optional)

It may be helpful to architects and stakeholders to document the kinds of issues for which this viewpoint is *not appropriate or not particularly useful*.

Identifying the "anti-concerns" of a given notation or approach may be a good antidote for certain overly used models and notations.

7.4. Model kinds+

⋆ Identify each model kind used in the viewpoint per ISO/IEC/IEEE 42010, 7c.

In the Standard, each architecture view consists of multiple architecture models. Each model is governed by a *model kind* which establishes the notations, conventions and rules for models of that type. See: ISO/IEC/IEEE 42010, 4.2.5, 5.5 and 5.6.

Repeat the next section for each model kind listed here the viewpoint being specified.

7.5. Ecosystem and transparency

⋆ Identify the model kind.

7.5.1. Ecosystem and transparency conventions

⋆ Describe the conventions for models of this kind.

Conventions include languages, notations, modeling techniques, analytical methods and other operations. These are key modeling resources that the model kind makes available to architects and determine the vocabularies for constructing models of the kind and therefore, how those models are interpreted and used.

It can be useful to separate these conventions into a *language part*: in terms of a metamodel or specification of notation to be used and a *process part*: to describe modeling techniques used to create the models and methods which can be used on the models that result. These include operations on models of the model kind.

The remainder of this section focuses on the language part. The next section focuses on the process part.

The Standard does not prescribe *how* modeling conventions are to be documented. The conventions could be defined:

I) by reference to an existing notation or language (such as SADT, UML or an architecture description language such as ArchiMate or SysML) or to an existing technique (such as $M/M/4$ queues);

II) by presenting a metamodel defining its core constructs;

III) via a template for users to fill in;

IV) by some combination of these methods or in some other manner.

Further guidance on methods I) through III) is provided below.

Sometimes conventions are applicable across more than one model kind – it is not necessary to provide a separate set of conventions, a metamodel, notations, or operations for each, when a single specification is adequate.

I) Model kind languages or notations (optional)

Identify or define the notation used in models of the kind.

Identify an existing notation or model language or define one that can be used for models of this model kind. Describe its syntax, semantics, tool support, as needed.

II) Model kind metamodel (optional)

A metamodel presents the AD elements that constitute the vocabulary of a model kind, and their rules of combination. There are different ways of representing metamodels (such as UML class diagrams, OWL, eCore). The metamodel should present:

entities What are the major sorts of conceptual elements that are present in models of this kind?

attributes What properties do entities possess in models of this kind?

relationships What relations are defined among entities in models of this kind?

constraints What constraints are there on entities, attributes and/or relationships and their combinations in models of this kind?

NOTE: *Metamodel constraints should not be confused with architecture constraints that apply to the subject being modeled, not the notations used.*

In the terms of the Standard, entities, attributes, relationships are *AD elements* per ISO/IEC/IEEE 42010, 3.4 4.2.5 and 5.7.

In the *Views-and-Beyond* approach [**?**], each viewtype (which is similar to a viewpoint) is specified by a set of elements, properties, and relations (which correspond to entities, attributes and relationships here, respectively).

When a viewpoint specifies multiple model kinds it can be useful to specify a single viewpoint metamodel unifying the definition of the model kinds and the expression of correspondence rules. When defining an architecture framework, it may be helpful to use a single metamodel to express multiple, related viewpoints and model kinds.

III) Model kind templates (optional)

Provide a template or form specifying the format and/or content of models of this model kind.

7.5.2. Ecosystem and transparency operations (optional)

Specify operations defined on models of this kind.

See §8.5 for further guidance.

7.5.3. Ecosystem and transparency correspondence rules

⋆ Document any correspondence rules associated with the model kind.

See §8.6 for further guidance.

7.6. Operations on views

Operations define the methods to be applied to views and their models. Types of operations include:

construction methods are the means by which views are constructed under this viewpoint. These operations could be in the form of process guidance (how to start, what to do next); or work product guidance (templates for views of this type). Construction techniques may also be heuristic: identifying styles, patterns, or other idioms to apply in the synthesis of the view.

interpretation methods which guide readers to understanding and interpreting architecture views and their models.

analysis methods are used to check, reason about, transform, predict, and evaluate architectural results from this view, including operations which refer to model correspondence rules.

implementation methods are the means by which to design and build systems using this view.

Another approach to categorizing operations is from Finkelstein et al. [**?**]. The *work plan* for a viewpoint defines 4 kinds of actions (on the view representations): *assembly actions* which contains the actions available to the developer to build a specification; *check actions* which contains the actions available to the developer to check the consistency of the specification; *viewpoint actions* which create new viewpoints as development proceeds; *guide actions* which provide the developer with guidance on what to do and when.

7.7. Correspondence rules

⋆ Document any correspondence rules defined by this viewpoint or its model kinds.

Usually, these rules will be across models or across views since, constraints within a model kind will have been specified as part of the conventions of that model kind.

See: ISO/IEC/IEEE 42010, 4.2.6 and 5.7

7.8. Examples (optional)

Provide helpful examples of use of the viewpoint for the reader (architects and other stakeholders).

7.9. Notes (optional)

Provide any additional information that users of the viewpoint may need or find helpful.

7.10. Sources

⋆ Identify sources for this architecture viewpoint, if any, including author, history, bibliographic references, prior art, per ISO/IEC/IEEE 42010, 7e.

8. System of Systems viewpoint: vehicle point of view

8.1. System of Systems: vehicle point of view

The viewpoint "System of Systems: vehicle point of view" focus on the car as a constituent of the SoS. Therefore, this viewpoint aims at giving an answer to this question: How to engineering a car so to be part of a system of systems?

8.2. Overview

Connected cars will benefit from Intelligent Transport Systems (ITS), Smart Cities and IoT to provide new application scenarios like smart traffic control, smart platooning coordination, collective collision avoidance, etc. Vehicles will combine data collected through its sensors with external data coming from the environment, e.g. other vehicles, road, cloud, etc.

Connected vehicles will face new challenges and opportunities related to safety issues. Current regulations for safety aspects, like the ISO 26262 standard, do not account for scenarios in which the vehicle is part of a more complex system; the challenge is on how to manage new hazards that coming from the environment could jeopardize safety. At the same time, connected cars open new opportunities for safety, called "connected safety" within Volvo Cars[7]: e.g., this new technology will allow a connected car to be aware of a slippery road, of cyclists on the road, etc., so to initiate all the actions needed to avoid accidents and collisions.

These scenarios are posing new requirements to the architecture. We foreseen two different viewpoints and views: (i) from the point of view of a single connected car, which has to offer the functionalities needed to realize the scenario, and (ii) from the point of view of the system of systems (i.e. cars connected with other entities of their environment), spanning from the agreements between the different systems affected, e.g. different OEMs, cloud providers, road infrastructures,

[7] `https://goo.gl/mIWWS3`

etc., to the definition of the scenario that the system of systems has to realize, like the slippery road mentioned above.

In general, the main issue with safety guarantees in connected cars is that a full analysis of the system is not possible at design time. When moving from a single vehicle to a cooperative system, a new safety analysis is required to handle uncertainties at runtime. Some approaches have been proposed to deal with certification at runtime, e.g. [**? ?**], but a clear framework that can be used to define the connected safety requirements is still missing.

8.3. Concerns and stakeholders

8.3.1. Concerns

In this section we focus on the concerns that are essential to enable functions in the systems of systems for cars. We express the concerns in the form of questions as suggested by the ISO/IEC/IEEE 42010 standard.

- Once the car is part of a SoS, how to guarantee functional safety requirements?
- Once the car is part of a SoS, what are the implication on system design and functional distribution for functional safety?
- Once functional safety requirements involve devices that are outside of the vehicle (other constituent systems of the SoS), how to ensure that these requirements will be guaranteed?
- How the methods and processes for end-to-end function development and continuous delivery of software need to evolve to be suitable in a systems of systems setting?
- How to enable a reliable and efficient communication between the vehicle and heterogeneous entities, like other vehicles, road signals, pedestrians, etc.?
- How to be sure that the vehicle and other constituent systems of the SoS will be able to exchange information and to use the information that has been ex-changed?
- How to guarantee that the security of the vehicle is preserved once the vehicle becomes connected?
- How to identify the right tradeoff between shared data and users' privacy?
- How to keep the data shared within the SoS (and possible replication of data) sufficiently updated or synchronized?
- How to manage the age of available information?
- Which functions in the car are allowed to make use of data coming from other constituents?

8.3.2. Typical stakeholders

Since it this viewpoint we take the vehicle point of view, potentially every stakeholder described in Section 5 and summarized in Table 1 will be part of the audience of this viewpoint. Moreover, additional stakeholders external to the company should be considered as stakeholders. Examples of them are standard authorities for communication means used by the SoS, stakeholders of other OEMs, suppliers, road authorities, etc.

8.4. Model kinds+

There are not obvious model kinds that can be used for this viewpoint. UML and general purpose architecture description languages seem to be not appropriate to architecting SoSs. There are some attempts in the literature to define architectural languages (Als) for SoS, such as... Patrizio ►*Add Oquendo and others*◄ , however, a deep analysis of these languages need to be performed in order to precisely understand whether the ADL can exactly satisfy the needs of this viewpoint and of Volvo cars. Results of an initial analysis suggest that this language is might be too formal and heavy. As it is stated in [**?** **?**] often ALs are not adopted by practitioners since they are (i) too complex and require specialized competencies, (ii) the return on investment is insufficient perceived, (iii) they require over-specification and at the same time practitioners are not able to model design decisions explicitly in the AL, and (iv) there is a lack of integration in the software and system life cycle, lack of mature tools, and usability issues.

This suggests that the language to be used should be defined and/or customized within the company, so that it can precisely implement the needs of the company and users and at the same time can be integrated with the company life-cycle. An interesting approach to customize existing ALs according to specific needs might be found in [**?**].

The model kind for this viewpoint should enable the specification of the following concepts that are essential building blocks for engineering a car that will be part of a system of systems. These building blocks have been identified within the NGEA project and are documented in the deliverable D3.2 Systems of Systems for Cars Concepts.

- **Heterogeneity of communication channels**: Today cars already use several communication channels for communicating with other systems. Communication means include also GPS receivers, sound signals, visual break lights, and turn indicators, which can be used to communicate to other systems the intention of the driver, as well as cameras able to detect other vehicles intentions or to read road signs, which are then highlighted to the driver. In the near future we can expect several new communication channels. Examples are IMT2020 (5G) the successor of the LTE (4G) mobile networks, Vehicle to vehicle (V2V) and Vehicle to infrastructure (V2I) communication over IEEE 802.11p, BLE (Bluetooth low energy), Proprietary communication channels used by single or a few vendors, standard WiFi versions, etc.

- **Connectivity**: Connectivity is essential to enable the expected service level in different places of the SoS. Connectivity may be possible through many different channels as discusses in the previous item. However there has to be on-board functions that handle graceful degradation of services when connectivity is limited or not available at all. The user shall experience a robust behaviour of the available functions. The quality of service (QoS) of both data and functions must be made sufficiently clear to the users. Acceptable QoS typically requires very good connectivity and availability of real-time information. However, since connectivity cannot always be guaranteed, other forms of fall-back solutions or redundancy are usually needed. The QoS available in a specific location and time need to be communicated to the user in a user-friendly and easy to grasp way.

- **Large and unreliable information**: low quality information could lead to disastrous events, especially when functions are implemented to rely heavily on external information. Therefore, it is important to be able to determine whether the received information's quality

is high in term of correctness and timeliness. Multi-sensor data fusion provides an answer by combining perceived data from sensors to make the resulting information more reliable.

- **Interoperability**: Interoperability is the ability of diverse systems to work together. This general definition has been conjugated in many different ways based on the reference application area and on the many different factors and aspects characterizing them. Interoperability involves standards, protocols, and integration and adaptation of interfaces to enable the effective and efficient communication between constituent systems. Interoperability is the ability of two or more constituent systems that are part of SoS to exchange information and to use the information that has been ex-changed. Unambiguous interpretation of shared data between systems is necessary for interoperation, but it is not sufficient. Despite standards for shared data provides specification with the objective to enhance the functionality and interoperability, the data encoded using these standards are not necessarily interoperable. For instance, concepts that have the same labels, and somehow even the same meaning, can be used completely differently in different applications. This is for instance the case of speed within a car that can have different meanings.

- **Cyber security and privacy**: Connectivity of cars opens problems of both security and privacy. On one side cars become exposed as any other device that is connected to Internet. A representative example is given by Jeep Patrizio ►*find reference*◄ . For what concerns privacy, as mentioned above cars will need to share several information, however sensitive information should be properly protected.

- **Distributed end-to-end functionality**: End-to-end functionalities should be provided not only between nodes in the vehicles but also outside the vehicle, and then involving other constituent systems of the SoS, e.g., clouds, other cars, pedestrians, road infrastructure and signals, etc. Connected vehicles can benefit a lot from access to cloud computing based data and services. Services implemented in distributed end-to-end functions will benefit from the possibility to dynamically load software to the on-board electrical architecture. Dynamically loaded software may be executed in one or several physical nodes, and this will open challenges in terms of cyber-security, functional safety and compatibility.

- **Functional safety**: Functional safety requirements are expected to apply for functions outside the car. To be able to handle safety once the car becomes a constituent system of a SoS, one can expect that operational and key functions will be protected. However, connected safety[8] promises important and interesting improvements for what concerns the overall safety of the car. A typical example is a car that receives information about some ice on the street from the cloud as reported from another trusted vehicle. These types of scenarios might implicitly create expectations to the driver of a car that will start relying on a service that is however dependent on potentially uncontrollable connections, devices, sensors, etc.

8.5. Operations on views

An interesting way to define model kinds is to properly define the metamodel of the model kind through the use of model-driven engineering techniques like ecore Patrizio ►*write properly*◄ . This

[8] Patrizio ►*add reference to connected safety of Volvo cars*◄

will help defining operations to define views (model) that are compliant to the viewpoint's rules and constraints (conforms to metamodel). The interested reader can refer to the work described in [**?** **?**].

Other operations should help and guide the user in interpreting the views. The starting point should be a serious study on understanding exactly both the users and the purpose of the views. This study will then define the requirements of the syntax (e.g., graphical, textual, tree-like) to be used for the language. Operations should be also provided to evaluate architecture descriptions and decisions. For instance, analysis methods might be defined to evaluate the interoperability level, the ability of the system to cope with heterogeneity of communication channels, security and privacy when the car become part of the SoS, etc. Finally, the view should be connected to the software and system life-cycle through methods helping the design and build of the system.

8.6. Correspondence rules

⋆ Document any correspondence rules defined by this viewpoint or its model kinds.

Usually, these rules will be across models or across views since, constraints within a model kind will have been specified as part of the conventions of that model kind.

See: ISO/IEC/IEEE 42010, 4.2.6 and 5.7

- security, privacy, the other SoS viewpoint

9. Discussion and Concluding remarks

In this paper we described the current evaluation of Volvo Cars towards the definition of an architecture framework. In this stage we are identifying potential viewpoints of the vehicles of the near future. We will then identify the modeling languages to be used to describe the architecture. According to the actual needs and to the consumers of the particular view, we will identify the modeling language to be used and we will find the right tradeoff between a language (i) able to support the level of formality and precision required by disciplined development processes, or (ii) simple and intuitive enough to communicate the right message to stakeholders and to promote collaboration [**?** **?**].

For what concerns the realization of the architecture framework we will investigate MEGAF [**?** **?**], which is an infrastructure that enables software architects to realize their own architecture frameworks specialized for a particular application domain or community of stakeholders and compliant to the ISO/IEC/IEEE 42010 standard [**?**].