METHODOLOGY AND ARCHITECTURE FOR SAFETY MANAGEMENT

Matthieu Carré[1,2], Ernesto Exposito[1] and Javier Ibañez-Guzmán[1,2]

[1]Univ Pau & Pays Adour, E2S UPPA, LIUPPA, EA3000, Anglet, 64600, France

[2]Renault S.A.S, 1 av. du Golf, Guyancourt, 78288, France.

ABSTRACT

The design of complex systems, as in the case of autonomous vehicles, requires a specialized systems engineering methodology and an adapted modelling framework. In particular, the integration of non-functional requirements, as important as the Safety, requires from this methodological framework the well-adapted semantic expression of constraints as well as their traceability during all phases of analysis, design and implementation. This paper focuses on the study of model-based autonomous system design and investigates the design flows and initiatives grasping with this complex computational model. The specialization of the ARCADIA methodology will be illustrated in a real industrial case.

KEYWORDS

Model Based System Engineering, Safety, Autonomous vehicles, System Engineering analysis, System Engineering design.

1. INTRODUCTION

Ensuring the trustworthiness of autonomous systems can not only be based on evidence from some rigorous design methodology and critical system engineering techniques and standards [1]. The enforcement of AV safety when the system is designed, implemented, and operated have led us to the identification of four mandatory non-functional properties for safe autonomy (NFP). Their considerations make the system progress towards safe-by-design architectures as well as ones that are more robust, easier to design, and easier to verify and validate. Their considerations also traduce into providing evidence that the abstract system model fosters the system's nominal behaviors, guarantees its critical system goals and manages any deviations from them. Their goals, coordination and composition may only result of a global model-based analysis with a symbolic and conservative representation both human and machine-readable. Therefore, the whole design process requires a more or less exhaustive analysis to identify all kind of harmful events and their possible effects.

Then, the implementation of DIR (Detection, Isolation, Recovery) mechanisms contributes to detect those symptoms and mitigate them as it keeps or brings the system back to a set of trustworthy states. Moving them from correctness at design time to run-time autonomic management may reduce existing gap between critical and best-effort systems engineering on autonomous systems. They require not only cutting-edge theory but also to find adequate trade-offs between quality of control and performance. Proposing a variety of DIR-type

Natarajan Meghanathan et al. (Eds) : ICCSEA, WiMoA, SCAI, SPPR, InWeS, NECO - 2019
pp. 01-15, 2019. © CS & IT-CSCP 2019 DOI: 10.5121/csit.2019.91801

adaptive processes within the supervised system involve complex decision methods by keeping critical goals and plan best-effort goals according to resource availability. Immediate downsides are the ability to react promptly for timely recovery or the creation possible conflicts between the different control loops.

Shifting the mitigation and assurance to run-time to cope with uncertainty and context complexity constitutes a critical change in System Engineering. This turning point involves new types of theory and architecture to handle these system-level functionalities and new kinds of architecture coping with the different NFP (e.g. large distributed evolvable autonomous systems with non-predicable dynamically changing environments).

This proposed computational model for autonomous systems suggested in [1] can provide a basis for studying model-based autonomous system design. However, such abstract model-based architecture may appear difficult to grasp in its entirety at the first sight since the design and run-time operations have become more complex. Nonetheless, this additional complexity is necessary to appropriately integrate the results from the environment analysis and symbolic representation that contribute to tracking the possible trade-offs and evidence. Design flows involving these solutions are not yet mature and rigorous in the automotive industry, but some initiatives start to provide the different pieces to build up such a solution.

This paper focuses on the study of model-based autonomous system design and investigates the design flows and initiatives grasping with the previous computational model. This paper is organized as following. Section 2 details how System Engineering allows coping with the complexity of designing and integrating the different concerns and quality properties using views, in particular, safety. We also survey self-adaptive software solutions in the literature that contribute to performing safe operations from a quality assurance perspective. Section 3 formulates and deriving safety constraints as conditions or risk measures that cover all relevant hazardous events in regard to the vehicle behaviour. The scope of the work turn around behavioural safety for a level 4-5 Autonomous Vehicle so that functional safety is not directly addressed but remains acknowledged for the later technical design of the supervisor. A abstract model of the vehicle is proposed using the Arcadia methodology involving the expected operational functionalities from the vehicle, the system, the environment and the actors (i.e. stakeholders, actors, regulators, recommendations). Section 4 proposes the analysis and design of the solution independent from any technical solution as logical and technical requirements to be met in the industrial framework proposed for autonomous vehicles. Section 5 summarizes our findings, identifies current limitation and proposes a few perspectives.

2. AV SAFETY AND SYSTEMS ENGINEERING

This section investigates how safety in AV is managed in SE. We pursue the finding and analysis by exploring research works and solutions in the different domains limited to QoS regarding one or more properties.

2.1. System Design Complexity in Autonomy

The combination of multiple functions having different and complementary capabilities enables the emergence of Autonomous Vehicles. Their deployment is limited by the level of complexity they represent together with the challenges encountered in real environments with strong safety concerns. Thus a major concern prior to massive deployment is on how to ensure the safety of autonomous vehicles despite likely internal (e.g. malfunctions) and external (e.g. aggressive behaviors) disturbance they might undergo.

System Engineering has partially contributed to respond to the complexity of autonomy, context, system architecture and knowledge. The autonomy involves two challenges in both System Engineering methods and Knowledge representations.

In the first place, System Engineering offers to manage complexity of the AV functional and non-functional decomposition. The System Theory is a way to cope with complexity alongside analytic reduction and statistics. Systematic approach to manage complexity with appearance of new defined needs: the necessity to provide holistic and systematic engineering (through methodology, tools, processes and definition). It offers traceability and flexibility of the architecture (systems, components, features) in the life cycle. In addition, it may provide top-down, bottom-up and middle-out possibilities to enhance and integrate learning processes, updates, new regulations facilitating the acquisition and adaptability of the knowledge mandatory for autonomous systems.

2.2. "Safe-by-Design" Architectures towards run-time Safety Representation and Assurance

Despite the fact that the automotive industry practices a bottom-up approach to the engineering of autonomous driving systems focusing on functions and technologies first, architectures have been including those concerns and architecting towards these properties. They capture both AV functional decomposition, AV non-functional decomposition (i.e. safety) and AV contextual decomposition. The compliance to them in standards and research towards more self-awareness and safety in AV have successively been introduced at different scale of adaptation to the system.

Hybrid architecture: In the first place, the structure of the framework of Albus on 4D/RCS [2] introduces a hybrid control architecture being both deliberative (i.e. actions based on reasoning and planning) and reactive (i.e. fast actions based on direct and simple condition on feedback). On the one hand, the framework structure ensures reasoning and planning processes based on goals and priorities of the decision entities within a control hierarchy. On the other hand, reactive loops that provide a faster and controlled response are introduced at each level of the control hierarchy, and can locally modify planned actions to adapt to new events. This approach is close in its concept and structure composition to the autonomic computing paradigm [3] introduced by IBM, with a hierarchy of autonomic orchestrating managers in IT infrastructure.

Degraded states and modes: In their work to define robust system architectures, Tas et al. [4] emphasizes the relation between architectural design and the use of degraded operation modes, introduced by [5]. The authors refer to the design of a functional and layered system architecture as one of the main focus of autonomous driving to deal with safety-critical challenges in systems engineering. Moreover, they suggest that the use of an effective monitoring system is necessary to give proper feedback to the vehicle about its state and to allow to take well-adapted decisions. Modes and graceful degradations continue to be investigated in the literature [6].

Self-awareness on system's abilities and limitations: In the matter of monitoring, Reschka et al. [7] reports the needs to provide a permanent online monitoring of vehicle capabilities, as well an adequate modelling tool to support appropriate and safe decisions. Accordingly, the framework perceives the current performance of the system during operation, by knowing the range of actions of the system and its limitations. With respect to ISO26262 standard [8], the authors propose the use of ability and skill graphs to design the functional architecture of AV. Those skills reflect the performance feedback of the system and then are used for system self-

perception. Moreover, Colwell et al. [9] also introduce degradation for subsystem functionalities and ODD restriction to grasp the limit of the safe system operation.

Constructivist and layered evolvable architecture: Regarding the long-term evolution of AV in systems engineering, Behere et al. [10] overview the key functional components and orientations needed for autonomous driving, to establish a layered evolvable functional architecture. They report the necessity of constructivist architectures managed by Artificial Intelligence to tackle the limitations of current engineering practices for scaling to more complex systems. These so called constructivist architectures introduce a fundamental shift from manually designed to self-organising architectures that evolve at run-time. The current challenge resides in the possibility of run-time reasoning and run-time verification of the desired properties as safety constraints. The design and implementation of such architectures will involve paradigms and technologies from non-automotive domains. For example, the use of reflective intelligent control systems based on a high-level supervisor on the functional layer that will allow the monitoring and change of its behaviour for better adaptations.

Those approaches have been introduced progressively in the automotive domain in order to propose solutions for well-adapted decisions; to apprehend the range and limitation of actions of the system; to evolve the structure to provide better adaptation behaviours; and finally, to integrate non-functional dimensions. Researchers have started to investigate the application of such models in run-time environments combining safe-by-design and run-time mitigation across the system and its functionalities. This software change beyond typical software evolution approaches which has led to the vision of self-adaptive software.

2.3. Ensuring Safe Operations with Self-adaptive Software

Several run-time models that have been experienced and surveyed to cope with uncertainty have been proposed [11, 12]. They are presented as an appropriate solution to cope with the concerns of safety in AV emphasised previously. This type of architecture involves constructivist or evolvable or dynamic behaviours or systems and are referred as self-adaptive architecture. They act as an orchestration solution that can support acceptable trade-off to pursue the assurance of some non-functional quality such as safety in evolvable and reconfigurable system [13].

2.4. Composable and Flexible Systems

2.4.1. Introduction to microservice

Regarding software architecture, modularization and abstraction have been the first stages to break down the complexity of systems in programs and architectures. However, according to Dragoni et al. [14], those stages still result into the creation of monoliths, where monolithic applications are generally built as a single unit with functionalities spread in modules and components that are not independently executable. The final product and code result in difficulties for scalability, maintainability and evolvability. In order to cope with these considerations it suggests the use of micro-services as an architectural style in software development to create a single application as a suite of small composable services. In order to confront the previous issues, the components are forced to only implement one functionality that results in the expression of one capability. Inspired by service oriented computing, Martin Fowler et al. [15] formalize the terminology of this current movement in software development as alternative to monolithic style applications. The approach suggests to isolate all business capabilities of the systems into simple and small specialized services similar to components but decoupled enough to be independently deployable by an automated

deployment machinery. A micro-service results to be an independent functional process that express one capability, also defined as cohesive [14], interacts with messages and owns its domain logic locally.

2.4.2. Microservice in self-adaptive or safety-critical systems

[16] is an application of micro-service architecture for aerial unmanned vehicle. The intended approach in its ability to discover new services at run-time and provide an automated deployment machinery requires micro-services to be auto-descriptive by semantically describing their business capability (behaviour), goal (system), logic, structure (component) and interface (communication). The advantages of using microservices in designing is their capacity to fail small. Not only they allow smart adaptation based on observation of such faults, they also make the fault-tolerance mechanisms easier to bound.

3. MODELLING FOR SAFETY ASSESSMENT WITH MBSE IN AV

3.1. Application Scope

Current work in the domain have proposed relevant approaches for the refinement, integration and enforcement of AV safety. Considering safety as a dynamic control problem proposed by STPA/STAMP in [17] has shown promising applications and results in [18, 19], and a complementary approach in [20, 21] towards the AV standards and the traditional failure analysis for hazards coverage. However, most conclusions of these works insist on the difficulty to provide a scalable automation for production-ready analysis processes to the AV context and a flexible enforcement of these safety constraints in parallel. Consequently, they only result in a partially-holistic view of the ADS restricting its ability to argue (e.g. on decisions, observation restrictions). The flexibility and manageability of performing safety assurance at run-time in AV can't be solely addressed with the related work of automotive domains.

Table 1. Mitigation of hazards in safety engineering

Hazard	**Through**
identification	Analysis
elimination	Design
Control	Management

3.1.1. Control structure for safety analysis towards reduce residual risks

The combination with the conventional ISO26262 analysis (FMEA, HARA, Safety case) and STPA allows to perform safety analysis in order to provide a large scope of detection of hazard and mitigation of the residual risk. The goal remains the same as it is to create systems that integrates easily the safety requirements of any types that may have impact on design or can only be mitigated during system operation.

The primary concern of System Safety is the management of hazards as described in Table 1. It presents the 3 ways of managing hazards leading to the reduction of the risk to a manageable and acceptable level.

3.1.2. Composition of the Safety Analysis and Assessment in AV architecture

In our approach, we propose to apply to integrate the awareness of the hazard analysis in the system model via some machine-understandable knowledge. We also consider the strategies of the hazard management in the form of control loops. We propose to call these loops the "safety assessment processes" as they constitute workflows involving different functions like monitoring, classification, evaluation, planning or mitigation. To determine the rightful allocation between the different functions, dataflows and each workflow, we propose to perform a combined safety analysis with STPA/STAMP and ISO.

The combination of both STPA/STAMP and ISO hazard analysis have been performed and shown an extended coverage of the hazard [22]. However, such application requires to consider a specific ODD and reveals to be an iterative process with continuous refinement. Experiments [20] have been applying it to AV and formalized this context-dependant analysis method using a taxonomy (operational keyword to generate a sentence creating the different scenarios).

In the first place, we propose to start the hazard analysis at the high-level representation of our architecture with the behaviour competencies and regulations. Consequently, we start by identifying some general use cases involving the vehicle competencies (e.g. perceive and react near a pedestrian crossing). Then, we determine the hazards, the control structure of the system and identify the unsafe control actions to construct the hazardous scenarios. Finally, safety constraints can be identified and allocated to the different systems or components or functions involved. This integration may result in the creation of a new version of the system architecture (i.e. design constraints). The constraints that suggest some run-time management through control loops are called "safety assessment processes" (SAP) in our research. They will be the focus of our work as we want them to be deployed when the context requires it.
This process is then iteratively performed by applying it these new version but also when the system architecture is continuously refined and more defined in-depth (bottom-up).

3.2. Background on MBSE

The previous point were dealing with the process of designing and maintaining architecture. However, what language or formalization can be used to describe software architecture ?
UML (Unified Modelling Language) is one of object-oriented solutions used in software modelling and design. UML is a language (of object modelling) and therefore proposes a notation and a semantics associated with this language (i.e. models), but no process methodology (i.e. an approach proposing a sequence of steps and activities that lead to the resolution of a problem). Therefore, UML is not a method. UML unifies object methods with a design process that was strongly influenced by its intended use in object programming. UML has a whole approach (i.e. covering the entire development cycle: analysis, design and implementation) that is object-oriented (and not functional): the system is broken down into collaborating objects (rather than tasks broken down into functions that are easier to perform). However, its application is not particularly adapted to modelling complex systems, and suited to system engineering in the first place.

The Architecture View Model designed by Philippe Kruchten, also called 4+1 view model represents the functional and non-functional requirements of software-intensive systems. They are represented in the different views that offer and describe the system from the viewpoints of different stakeholders (e.g. end-users, developers, system engineer, and standards). The four views of the model consist of the logical, development, process and physical views. Each one achieve a set of specific concerns regarding the system in addition to a use case or scenario view that describes the sequences of interactions between objects

and between processes involving the system and guiding the other views. The logical view is high-level and focuses on abstraction and encapsulation, it models the main elements and mechanisms of the system. It identifies the elements of the domain, as well as the relationships and interactions between these elements "notions of classes and relationships". The component view expresses the physical perspective of the code organization in terms of modules, components and especially the concepts of the language or implementation environment. From this perspective, the architect is mainly concerned with the aspects of code management, compilation order, reuse, integration and other pure development constraints. To represent this perspective, UML provides adapted concepts such as modules, components, dependencies, interface. The process view is very important in multitasking environments; it expresses the perspective on concurrent and parallel activities. The deployment view expresses the distribution of the system through a network of computers and processing logic nodes. This view is particularly useful for describing the distribution of a distributed system.

SysML is the new modeling language defined by the OMG. It can be seen as an extension of UML for modelling a broad spectrum of complex systems. It is based on UML and replaces class and object modeling with block modeling for a vocabulary more adapted to System Engineering. A block includes any software, hardware, data, process, and even people management concept.

Much more than a simple modeling tool, Capella is a model-based engineering solution that has been successfully deployed in a wide variety of industrial contexts. Based on graphical modeling, it provides system, software and hardware architects with rich methodological advice based on Arcadia, a complete model-based engineering method. The DSML Arcadia/Capella is based on the UML/SysML and NAF standards, and shares many concepts with these languages. It is the result of an iterative definition process driven by software systems and architects working in a wide range of business areas (transport, avionics, space, radar, etc.). Many industrial companies, such as Airbus, Areva, Thales, Continental and Renault are currently interested in using Capella and running pilot modelling projects with this tool. It exists because Arcadia allows you to:

- Ensure collaboration at the engineering level by sharing the same reference architecture.
- Manage the complexity of systems and architectures.
- Define the best optimal architectures through compromise analysis.
- Manage different levels of engineering and traceability through automated transition and refinement of information.

The couple Capella/Arcadia associates both the tool and the language: referring to MBSE’s three well-known pillars, one could say that ARCADIA provides both a modeling language and a modeling approach, and that Capella knows the language and method perfectly.

With a similar scope around adaptation, EUREMA is an integrated MDE approach as it rigorously and consistently uses models for engineering feedback loops. The models are used (1) to represent the adaptable software to achieve self-adaptation as proposed by the idea of models at run-time (Models@run.time), as well as to design (2) individual adaptation activities for feedback loops, (3) feedback loops as a whole, and (4) coordination between these loops. Finally, (5) these models are used throughout the life cycle of the self-adaptive system, i.e. to specify, execute and evolve feedback loops with their adaptation activities. This approach introduces a model-driven architecture approach that guide the design and development of complex and evolving systems while claiming to guarantee the portability, the interoperability and the reusability of the final system.

3.3. Modeling with Arcadia/Capella

ARCADIA (ARChitecture Analysis and Design Integrated Approach) is a model-based engineering method for the architectural design of systems, hardware and software. It was developed by Thales between 2005 and 2010 through an iterative process involving operational architects from all Thales businesses (transport, avionics, space, radar, etc.). It applies a structured approach over successive engineering phases that establishes a clear separation between requirements (operational needs analysis and system needs analysis) and solutions (logical and physical architectures), in accordance with the IEEE 1220 standard. ARCADIA recommends three mandatory interdependent activities at the same level of importance: Needs analysis and modeling; Construction of architectures and validation; and Requirements engineering.

Arcadia DSML (Domain-Specific Modeling Language) is based on UML/SysML standards and NAF and shares many concepts with these languages. But a modeling language was preferred in order to facilitate ownership by all parties stakeholders. Arcadia is mainly based on functional analysis, then the allocation of functions to the components. The richness of Arcadia DSML is comparable to SysML with about ten types of diagrams: data flow diagrams, scenario diagrams, state and mode diagrams, component distribution diagrams, component distribution diagrams, component distribution diagrams functional distribution, etc. There are also diagrams available from ARCADIA at different levels that are detailed in Figure 3.1.

The Capella tool provides the different set of tools to carry out the system modelling and automated transition steps from one level to the next providing traceability in the refinement of the different element of the system model architecture.

It is noticeable that the Arcadia method is presented in a top-down manner by nature. However, the representation can also be bottom-up if we start from an existing system for instance. In the next sections, our example will assume the extension of the ADCC architecture. For this purpose, the Arcadia methodology relates more about levels providing key representations objectives rather than phases or steps. In addition, all the architectural levels are not stated as mandatory. Since the methodology aims to guide the analysis and modelling processes, some levels may be skipped depending the system complexity and model expectations (e.g. operational analysis, logical architecture, EPSB are optional).

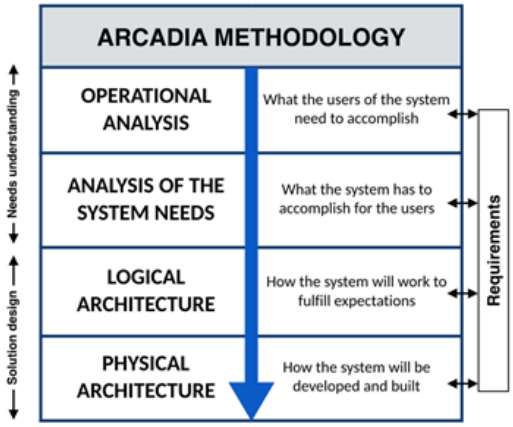

Figure 1. The main engineering levels of Arcadia and transition steps [23]

3.3.1. Operational Analysis

The Operational Analysis represents the highest level of the Arcadia method. It focuses on answering the question “what the users of the future system need to accomplish?”. At this level, the future system is not yet represented as a modeling element but the identification of the needs and objectives of the system start in this step. This level constitutes the first representation of the system environment by modelling the jobs of future users as activities, roles to fulfil while precising the operational conditions of the system. A series of activities and of interactions constitutes a process and contributes toward an operational capability. The creation of models and scenarios representing the different elements allows to check the adequacy of the system to these operational needs.

This level of Operational Analysis consists in:

- Capture and formalize the operational needs from the different stakeholders.
- Define what the vehicle (System) need to accomplish and how the other actors contribute, influence or impact.
- Also identify the different entities in each separate domain (entities, actors, roles, activities and concepts).

3.3.2. System Analysis

The System Analysis places the system as the central element of the dataflow representations in interaction with the external actors. It focuses on answering the question “what the system must accomplish for the users?”. The System Analysis involves the identification of system capabilities and functions that satisfy the operational needs defined in the previous level. At this point, the system is still considered as a black-box as no internal structure should be decided yet. Only function and communication exchanges may be allocated during this external functional analysis. Consequently, this analysis consists in refining the system functions with the decomposition of the top-level functions and adding the constraints of non-functional properties. For these aspects, these diagrams on Capella provide rich mechanisms for managing complexity: simplified links calculated between high-level functions, categorization of exchanges, etc. Functional chain can also be displayed as highlighted paths. They represents a specific path involving a of the system functions and exchanges and are relevant for assigning constraints (latency, criticality, etc.), as well as organizing tests on a specific system features.

This level of System Analysis consists in:

- Define what the system have to accomplish for the users
- Identify the boundary of the black-box system, consolidate the requirements(sourced from the actors)
- Model functional data-flows and dynamic behaviours

3.3.3. Logical Architecture

The Logical Architecture aims to identify the different logical components that form the component structure inside the system. It focuses on answering the question “how the system will work to fulfil expectations?”. This level provides tools to represent the relations between components and their content, independently of any considerations of technology or implementation.

The previous model described in the System Analysis can be imported via an automated transition to Logical Architecture model. Thus, it keeps all the previous definition and description of the top-level function and exchanges. On this basis, the Logical Architecture carries out the internal functional analysis where all sub-functions subsumed by the previously identified top-level functions need to be identified and allocated to a specific logical component while keeping tracks of the integration of the non-functional constraints .

This level of Logical Architecture consists in:

- Provide a white-box vision of the system identifying how the system is fulfilling the expectations
- Propose a first step for trade off-analysis

3.3.4. Physical Architecture

This level of Physical Architecture possesses the same objective than the Logical Analysis but defines the final architecture of the system providing information on how the system will be built. This level reflects the technical choices and defines the different components (software and hardware).

This level of Physical Architecture consists in:

- How the system will be developed and built
- Software vs. hardware allocation, specification of interfaces, deployment configurations, trade-off analysis

3.3.5. EPBS and integration contracts

An additional level EPBS (End Product Breakdown Structure) is available to define the "conditions that each component must fulfil to satisfy the architecture design constraints and limitations, established in the previous phases" [23]. This last level answers to the question "what is expected from the provider of each component" and is usually used with sub-contracting.

4. INTEGRATING DYNAMIC SAFETY ANALYSIS AND MANAGEMENT IN ARCADIA/CAPELLA

4.1. Overview of the Composed Methodology

We propose to design a methodology based on the Arcadia to appropriately represents the safety-related adaptive processes by synchronizing the step of the Arcadia methodology with the STPA analysis for the set of defined scenarii. For each step of Arcadia, we propose to associate an STPA analysis for any representation that contains the equivalent of a control loop.

4.2. Process of Integration for Safety Analysis

In the first place, it is necessary to base our analysis on what the behavioural requirements the AV system and vehicle needs to accomplish and the source of these requirements. In fact, multiple actors are the sources of requirements that can highly impact how the vehicle need to behave, how it is design and implemented. Standards, recommendations, guidelines, state-of-the-art, car manufacturer, regulators, renting or end customers impose diverse requirements to

the AV system that can be both functional or dysfunctional coping with totally distinct disciplines (e.g. functions, safety, ethics, social acceptance, security, etc).

4.2.1. Operational analysis

The operational analysis of the Arcadia methodology contribute to capture those operational needs from the different stakeholders. It defines what the vehicle need to accomplish and how the other actors contribute, influence or impact. Indeed, we model the whole process of actors and requirements that identify the different entities in each separate domain with a specific vocabulary of entities, actors, roles, activities and concepts. The functional part definition of the system takes as inputs the different behavioural competencies, OEDR, ODD, DDT and requirements from different shareholders available in the literature.

The actors of the non-functional part also takes place as they require specific operational capabilities to manage properly conditions or risk measures that cover all relevant hazardous events in regard to the vehicle behaviour. The following figures contribute to illustrate how high-level capabilities, entities, actors, interactions, activities and process. It is important to notice that the creation of diagrams contributes to refine step-by-step the model of the system. The next figures are views (or windows) to observe the semantic model that have been incrementally built using the Capella tool and following the Arcadia methodology.

Figure 2 defines the operational entities and the capabilities involving both the functional and the safety concerns. At this level, we start to represent the “real” needs of the various shareholders without the architecture, design or technical implementation in mind. We describe the model with Operational Capabilities (OC), Operational Entities and Operational Actors (OA) and the relation between them. For autonomous vehicle, shareholders often decompose the needs between functional and non-functional concerns. We also propose to distinguish functional and behavioral safety strategies from the functional part of driving to initiate the technical design of some sort of the core of the supervisor (both in the model and in the future physical architecture).

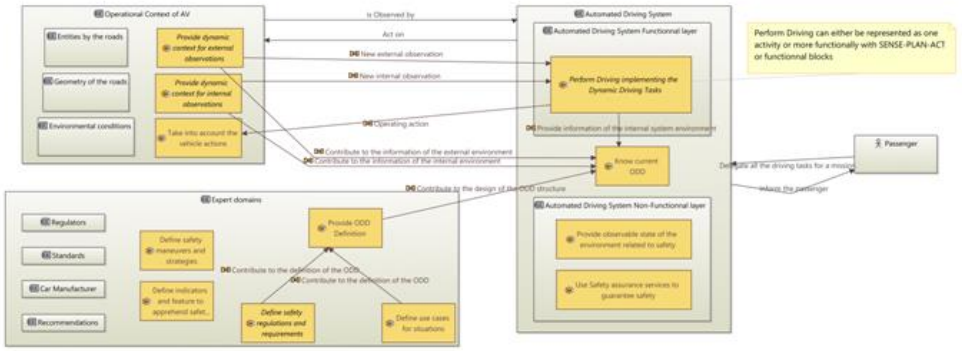

Figure 2. Safety Operational Analysis for Autonomous Vehicles

For this purpose, we have identify some functional oriented capabilities within the driving automation and some non-functional strategies (safety and social dimension). In addition, we have performed a additional step in the definition by refining each OC into some subsumed OCs. To illustrate, the DDTs are represented as OC as they can not be translated into activities and later as functions since they remain high-level definition of general service for operational objectives. In our case, their description and traceability is performed with some involved activities and operational processes (e.g. SENSE, PLAN, ACT).

The operational analysis phase defines also the architecture between the operational activities and the interactions between the different actors to grasp the operational context of the system. The activities (in yellow) correspond to the different process steps carried out in order to reach a precise objective for the entities. They might need to use the future system in order to do so, or the system might use it (e.g. Provide ODD definition). For example, we allocated to the operational context of the autonomous vehicle as the surrounding environment it may perceive (external observations) as well as its internal observations (system states). These observation are then consumed by the driving automation activity of the functional layer. This diagram also represents activities related to the ODD as it is a major source of the environment representation and formalisation of needs from the different stakeholders with expressions (e.g. rules or contracts to require behaviour X in situation Y, or to prevent X in Y). The construction (structure) and definition (content) of the ODD play a major role in our definition as well as the understanding of the current situations in which the system is involved. Finally, additional activities are also represented in interaction with the previous activities to address the non-functional concerns and strategies to use. To define with more detail the activities and the respective interactions for each capability (OC), Capella proposes to create an OAIB view (Operational Analysis Interaction Blank).

4.2.2. System analysis

The role of this next level is to identify the different functions or system services necessary to its users and specify them with the possible constraint from the non-functional properties. First, we have started with a definition of the system as a black-box and creating functions that realize the operational activities from the previous level. Figure 3 is a synthesis view of the system functions we have identified as parent functions: to perform the driving automation, to obtain access to the ADCC resources (Layer 1), to supervise them with specific loops of adaptation (Layer 2), to perform runtime adaptions to fit the ODD and safety concerns (Layer 3), to store this information for possible update or learning purposes (KB), and finally to display some system internal information. At this step, we have already made some choices regarding the design and how we wanted to structure and organize the control loops.

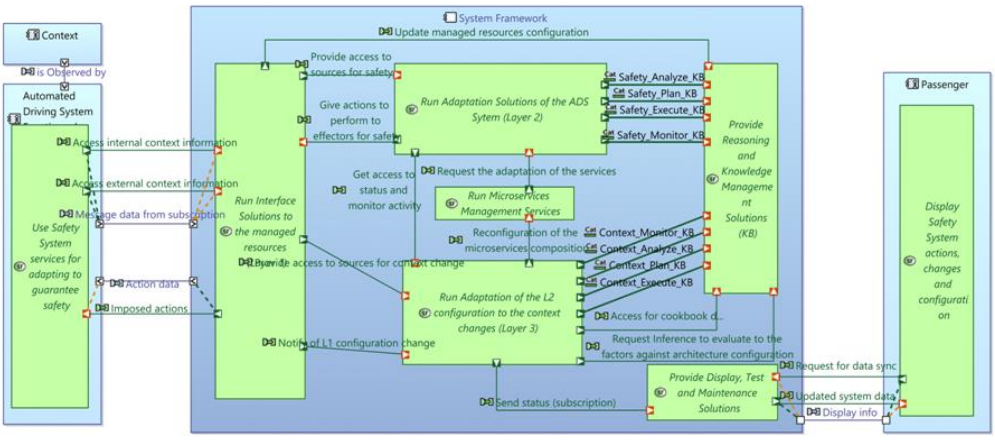

Figure 3. Safety High-Level System Analysis for Autonomous Vehicles

Figure 4 provides further details about the design choices for the architecture with the introduction of MAPE-K loops with an external knowledge base and service orientation.

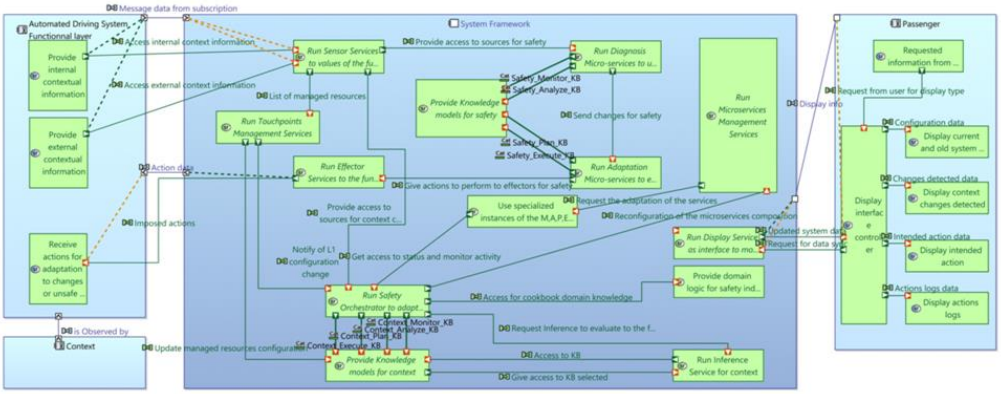

Figure 4. Safety Low-Level System Analysis for Autonomous Vehicles

Based on this operational and system analysis, the design phases can be started, based on the available technologies from the cyber-physical systems represented by autonomous vehicles.

For the design, several questions need to be answered:

- How the transition between the functional view to the logical and physical architecture can be achieved? We need to determine what types of architectural solutions can support the safety management loops and mitigation within the system. In addition, we also need to address the management of the constraints resulting from the AV context analysis. Determining the different operational contexts needs to be open for iterative and thorough safety analysis. Finally, we need to guarantee the constraints as defined distributed autonomic processes.

- How to assess the safety that remain behavioural? Hence, we want to ensure the safety and similar system attributes than in the design by managing the complexity of the systems for observability, traceability and flexibility. It may not be possible to solve by only tweaking profile or configuration: it is required to provide guarantees at any system or sub-system levels of adaptation.

CONCLUSIONS, LIMITATIONS AND PERSPECTIVES.

In this article, we have explored the way to study and build L4-L5 AV using MBSE with the key functionalities and safety considerations as design and run-time mitigation. We focus on the key functionalities for providing autonomy. Technological and technical solutions should not be pushed over them to provide manageable system complexity with sufficient level of safety coverage (quality and quantity) and NFP. It also enables the possibilities to perform trade-off between the various alternatives, combinations, variants and implementations of the various architecture. To conclude, the proposed computational model for autonomous systems can provide a basis for studying model-based autonomous system design. Nonetheless, we are far from ensuring that the conditions are in place to develop rigorous design flows. Indeed, the detailed structural and behavioural specification of the system needs to be provided in order to extend our current solution to include validation of the design based on functional and non-functional requirements. Future work will focus in developing extensions of our modelling approach, mainly based on new viewpoints aimed at integrating and validating non-functional properties, such as performance and safety.

ACKNOWLEDGEMENTS

This work has been financed by Renault and by FUI 23 under the French TORNADO research project focused on the interactions between autonomous vehicles and infrastructures for mobility services in low-density areas. Further details of the project are available at https://www.tornado-mobility.com.

REFERENCES

[1] Joseph Sifakis. Autonomous systems – an architectural characterization. In Models, Languages, and Tools for Concurrent and Distributed Programming, pages 388–410. Springer International Publishing, 2019. doi: 10.1007/ 978-3-030-21485-2_21.

[2] Gizem, Aksahya & Ayese, Ozcan (2009) Coomunications & Networks, Network Books, ABC Publishers.

[3] James S Albus, Hui-Min Huang, Elena R Messina, Karl Murphy, Maris Juberts, Alberto Lacaze, Stephen B Balakirsky, Michael O Shneier, Tsai Hong Hong, Harry A Scott, et al. 4d/rcs version 2.0: A reference model architecture for unmanned vehicle systems. NIST Interagency/Internal Report (NISTIR)-6910, 2002.

[4] J Kephart, D Chess, Craig Boutilier, Rajarshi Das, and William E Walsh. An architectural blueprint for autonomic computing. IBM White paper, June 2006. doi: 10.1.1.150.1011. URL https://pdfs.semanticscholar.org/0e99/ 837d9b1e70bb35d516e32ecfc345cd30e795.pdf.

[5] Ö. S¸ Tas¸ F. Kuhnt, J. M. Zöllner, and C. Stiller. Functional system architectures towards fully automated driving. In 2016 IEEE Intelligent Vehicles Symposium (IV), pages 304–309, June 2016. doi: 10.1109/IVS.2016.7535402.

[6] J. Lygeros, D. N. Godbole, and M. E. Broucke. Design of an extended architecture for degraded modes of operation of ivhs. In American Control Conference, Proceedings of the 1995, volume 5, pages 3592–3596 vol.5, Jun 1995. doi: 10.1109/ACC.1995.533806.

[7] Yrvann Emzivat, Javier Ibanez-Guzman, Herve Illy, Philippe Martinet, and Olivier H. Roux. A formal approach for the design of a dependable perception system for autonomous vehicles. In 2018 21st International Conference on Intelligent Transportation Systems (ITSC). IEEE, nov 2018. doi: 10.1109/itsc.2018.8569903.

[8] Andreas Reschka, Jürgen Rüdiger Böhmer, Tobias Nothdurft, Peter Hecker, Bernd Lichte, and Markus Maurer. A surveillance and safety system based on performance criteria and functional degradation for an autonomous vehicle. In 2012 15th International IEEE Conference on Intelligent Transportation Systems, pages 237–242. IEEE, 2012.

[9] ISO 26262 road vehicles – functional safety, 2018. URL http: //www.iso.org/iso/home/store/catalogue_tc/catalogue_ detail.htm?csnumber=43464.

[10] Colwell, B. Phan, S. Saleem, R. Salay, and K. Czarnecki. An automated vehicle safety concept based on runtime restriction of the operational design domain. In 2018 IEEE Intelligent Vehicles Symposium (IV), pages 1910–1917, June 2018. doi: 10.1109/IVS.2018.8500530.

[11] Sagar Behere and Martin Törngren. Systems Engineering and Architecting for Intelligent Autonomous Systems, chapter 13, pages 313–351. Springer International Publishing, Cham, 2017. ISBN 978-3-319-31895-0. doi: 10.1007/978-3-319-31895-0_13. URL https://doi.org/10.1007/ 978-3-319-31895-0_13.

[12] Holger Giese, Nelly Bencomo, Liliana Pasquale, Andres J. Ramirez, Paola Inverardi, Sebastian Wätzoldt, and Siobhán Clarke. Living with Uncertainty in the Age of Runtime Models, pages 47–100. Springer International Publishing, Cham, 2014. ISBN 978-3-319-08915-7. doi: 10.1007/978-3-319-08915-7_3. URL https://doi.org/10.1007/978-3-319-08915-7_3.

[13] Milos Ojdanic. Systematic literature review of safety-related challenges for autonomous systems in safety-critical applications. Master's thesis, MÃlardalen University, School of Innovation Design and Engineering, VÃsterÃs, Sweden, 2019.

[14] Sagar Behere and Martin Törngren. Systems Engineering and Architecting for Intelligent Autonomous Systems, chapter 13, pages 313–351. Springer International Publishing, Cham, 2017. ISBN 978-3-319-31895-0. doi: 10.1007/978-3-319-31895-0_13. URL https://doi.org/10.1007/ 978-3-319-31895-0_13.

[15] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch-Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. CoRR, abs/1606.04036, 2016. URL http://arxiv. org/abs/1606.04036.

[16] Martin Fowler and James Lewis. Microservices: a definition of this new architectural term. ThoughtWorks. http://martinfowler.com/articles/microservices.html [last accessed on July 06, 2016], 2014. URL http://martinfowler.com/ articles/microservices.html.

[17] D. Rodrigues, R. de Melo Pires, E. A. Marconato, C. Areias, J. C. Cunha, K. R. L. J. Castelo Branco, and M. Vieira. Service-oriented architectures for a flexible and safe use of unmanned aerial vehicles. IEEE Intelligent Transportation Systems Magazine, 9(1):97–109, Spring 2017. ISSN 1939-1390. doi: 10.1109/MITS.2016. 2611038.

[18] Nancy G. Leveson and John P. Thomas. STPA Handbook. MIT Partnership for a Systems Approach to Safety (PSAS, March 2018. URL http://psas.scripts.mit.edu/home/get_file.php?name= STPA_handbook.pdf.

[19] Gerrit Bagschik, Torben Stolte, and Markus Maurer. Safety analysis based on systems theory applied to an unmanned protective vehicle. Procedia Engineering, 179:61 – 71, 2017. ISSN 1877-7058. doi: http://dx.doi.org/10.1016/j.proeng.2017. 03.096. URL http://www.sciencedirect.com/science/article/ pii/S1877705817312122. 4th European {STAMP} Workshop 2016, {ESW} 2016, 13-15 September 2016, Zurich, Switzerland.

[20] Shawn A. Cook, Hsing-Hua Fan, Krzysztof Pennar, and Padma Sundaram. Building behavioral competency into stpa process models for automated driving systems. March 2018.
GiedreSabaliauskaite,LinShenLiew,andJinCui.Integratingautonomousvehicle
safety and security analysis using stpa method and the six-step model. International Journal on Advances in Security, 11:160–169, July 2018

[21] Mark A. Vernacchia. Gm presentation for introducing stamp/stpa
tools into standards. March 2018.

[22] Asim Abdulkhaleq, Stefan Wagner, Daniel Lammering, Hagen Boehmert, and Pierre Blueher. Using STPA in compliance with ISO 26262 for developing a safe architecture for fully automated vehicles.

[23] Pascal (Consultant) Roques. Systems Architecture Modeling with the Arca- dia Method. ISTE Press Ltd - Elsevier Inc, 2017. ISBN 9781785481680.