The Journal of Systems & Software 198 (2023) 111604

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

A compositional approach to creating architecture frameworks with an application to distributed AI systems[☆]

Hans-Martin Heyn[a,*], Eric Knauss[a], Patrizio Pelliccione[b]

[a] Department of Computer Science and Engineering, University of Gothenburg and Chalmers, Gothenburg, 41756, Sweden
[b] Gran Sasso Science Institute (GSSI), L'Aquila, 67100, Italy

ARTICLE INFO

Article history:
Received 31 March 2022
Received in revised form 21 December 2022
Accepted 27 December 2022
Available online 3 January 2023

Keywords:
AI systems
Architectural frameworks
Compositional thinking
Requirements engineering
Systems engineering

ABSTRACT

Artificial intelligence (AI) in its various forms finds more and more its way into complex distributed systems. For instance, it is used locally, as part of a sensor system, on the edge for low-latency high-performance inference, or in the cloud, e.g. for data mining. Modern complex systems, such as connected vehicles, are often part of an Internet of Things (IoT). This poses additional architectural challenges. To manage complexity, architectures are described with architecture frameworks, which are composed of a number of architectural views connected through correspondence rules. Despite some attempts, the definition of a mathematical foundation for architecture frameworks that are suitable for the development of distributed AI systems still requires investigation and study.

In this paper, we propose to extend the state of the art on architecture framework by providing a mathematical model for system architectures, which is scalable and supports co-evolution of different aspects for example of an AI system. Based on Design Science Research, this study starts by identifying the challenges with architectural frameworks in a use case of distributed AI systems. Then, we derive from the identified challenges four rules, and we formulate them by exploiting concepts from category theory. We show how compositional thinking can provide rules for the creation and management of architectural frameworks for complex systems, for example distributed systems with AI. The aim of the paper is not to provide viewpoints or architecture models specific to AI systems, but instead to provide guidelines based on a mathematical formulation on how a consistent framework can be built up with existing, or newly created, viewpoints. To put in practice and test the approach, the identified and formulated rules are applied to derive an architectural framework for the EU Horizon 2020 project "Very efficient deep learning in the IoT" (VEDLIoT) in the form of a case study.

1. Introduction

Architectural frameworks provide knowledge structures that allow for the division of architectural descriptions into different architectural views (Pelliccione et al., 2017). An architectural view expresses "the architecture of a system from the perspective of specific system concern" (ISO, 2012). The conventions of how an architectural view is constructed and interpreted is given through a corresponding architectural viewpoint. The design of a system-of-interest needs to account for different concerns of different stakeholders. Therefore, the architecture of the system-of-interest must be expressed through many different architectural views.

Designing a large and distributed system is a hierarchical process (Murugesan et al., 2019). Several views of the architecture of the system-of-interest allow for decomposing the design task into smaller and specialised tasks. This hierarchical design process allows for the co-evolution of requirements and architecture, known as the "twin peaks of requirements and architecture", as described in Nuseibeh (2001), Cleland-Huang et al. (2013).

Developing a complex system, which can include some form of AI, is a highly collaborative act (known as co-design) between many stakeholders. However, the term *co-design* can have two meanings. It can be an acronym for *collaborative design*, which means that all required stakeholders, i.e., developers of different disciplines, customers, business owners, etc, are being heard, and in some form actively involved in system design process (Fitzgerald et al., 2014; Nalchigar et al., 2021). Co-design can also stand for *integrated design* of a system, which means that design aspects of the system, e.g., hardware, software but also quality aspects are closely coupled to each other. In fact, many different concerns need to be satisfied with a system that includes a huge variety of different dimensions which need to be designed in parallel to ensure a "safe by design", "secure by design", "ethical by design", or any other required "quality by design" system.

[☆] Editor: Raffaela Mirandola.
[*] Corresponding author.
E-mail address: hans-martin.heyn@gu.se (H.-M. Heyn).

https://doi.org/10.1016/j.jss.2022.111604
0164-1212/

Architecture frameworks can explicitly support various non-functional quality requirements. Examples include a framework for addressing non-functional requirements in computer vision systems with AI (Fenn et al., 2016), a framework for architectural patterns improving operational stability (robustness) of machine learning systems (Yokoyama, 2019), a framework for architectures of machine learning enabled systems that improve safety (Serban, 2019), and a framework for architecture frameworks of distributed AI systems that emphasise security aspects (Mendhurwar and Mishra, 2021).

Each of these examples presents an architecture for AI systems which is tailored to only one particular quality aspect. In fact, architectural frameworks for AI systems are typically defined for one specific application domain. Examples of domain specific architectural framework include a framework for computer vision and cognition (Kurup et al., 2011), for self-driving vehicles (Schroeder et al., 2015), for smart power grid (Thilakarathne et al., 2020), and for health systems applications with machine learning (Moreb et al., 2020). In a systematic literature review on software engineering patterns for AI, Martínez-Fernández et al. (2021) noted that "at the system level, there are few proposals for patterns, design standards, or reference architectures". Also, in a review on architecture and design patterns for designing machine learning systems, Washizaki et al. (2019) noted that "developers are concerned with the complexity of ML systems and their lack of knowledge of the architecture[...]".

Summarising, we believe that the following three limitations of current architectural frameworks, especially towards AI systems, exist:

- For complex systems, such as for example AI systems distributed in the IoT, there is the need of both interpretations of co-design, i.e., collaborative design and integrated design to arrive at the desired system.
- If quality aspects are considered in a system architecture, mostly only certain specific non-functional requirements are adopted (such as security, or safety, or explainability). Again, a more generalisable approach is needed that can include any number of required quality aspects of the system.
- Many architecture frameworks are limited to specific application domains only and there is a lack of generalisable architectural frameworks.

As an example, distributed AI systems combine properties of AI systems with properties of systems in IoT. This also means that challenges from both AI system design and IoT system design need to be accounted for when designing an architecture framework that can also support distributed AI systems. This article investigates whether a generalisable architecture framework is possible; the architecture framework should be able to cope with any number of challenges relevant for complex systems, such as AI systems in IoT.

A viable approach towards a generalisable architecture framework could be to utilise compositional thinking. Compositional thinking bases on category theory, which is a formalised way of representing structures and orders through directed graphs (Awodey, 2010). Censi (2017) proposed to use compositional thinking to "co-design" functionality aspects of a complex system and the required resources such as hardware components. Zardini et al. (2020) demonstrated how applied category theory supports the co-design of hardware and software for an autonomous driving system, and Bakirtzis et al. (2021) applied compositional thinking to engineering of cyber–physical systems.

The main contribution is a mathematical description for compositional architectural frameworks. We first establish suitable descriptions of the abstractions levels for architectural views, their classification into clusters of concern, and elements from category theory to manage the relation between the views. Then, we show, based on an example, that the compositional approach to architectural frameworks can have the potential to ease co-design of systems that combine "classical" system components, AI components, and concerns when integrating an AI system into the Internet of Things (IoT), and thereby ease the identified challenges. Finally, the proposed compositional architectural framework is demonstrated on a use case taken from a joint industry project.

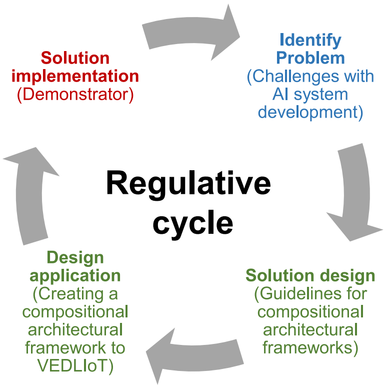

Regulative cycle
Identify Problem (Challenges with AI system development)
Solution design (Guidelines for compositional architectural frameworks)
Design application (Creating a compositional architectural framework to VEDLIoT)
Solution implementation (Demonstrator)

Fig. 1. Regulative cycle of the design science process as proposed in Wieringa (2009).

2. Research method

The study follows the design-science paradigm, outlined, among others, in Hevner et al. (2004) and Peffers et al. (2007). The study contains three major parts: (1) identification and motivation of the problem, (2) design, development, and validation of an artefact, and (3) demonstration of the artefact. The different parts contribute to answering the following research questions:

RQ1: Which challenges are relevant when defining system architectures for AI systems?

RQ2: What guidance can compositional thinking provide to overcome these challenges for the design of architectures for AI systems?

RQ3: How can a compositional framework be defined and applied in a realistic context?

Following the concept *design science as nested problem solving* proposed by Wieringa (2009), the different parts of the study represent a regulative cycle as illustrated in Fig. 1. As reflected by the research questions, our intention was to investigate a problem, designed a solution, validated the solution by applying it to a realistic context, and finally implement the solution in form of a demonstrator. Fig. 2 shows the resulting different elements of this study.

Through literature search and two focus groups with experts, Section 3 produces a list challenges when developing architectures for distributed AI systems. Section 4 presents the artefact of the design science study, which encompasses the theory of compositional architectural frameworks. In Section 5 the theory is tested by creating an architectural framework for the joint industry EU Horizon research project VEDLIoT. Following the design evaluation methods from Hevner et al. (2004), this constitutes an observational case study. Section 6 shows the outcome of the solution implementation on a real world use case. According to Hevner et al. (2004), this constitutes a descriptive evaluation using a detailed scenario around the artefact. Section 7 establishes the relation between the architectural framework theory and requirements engineering.

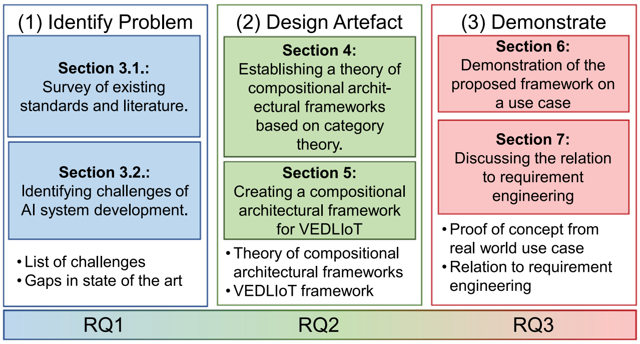

(1) Identify Problem
Section 3.1.: Survey of existing standards and literature.
Section 3.2.: Identifying challenges of AI system development.
• List of challenges
• Gaps in state of the art
(2) Design Artefact
Section 4: Establishing a theory of compositional architectural frameworks based on category theory.
Section 5: Creating a compositional architectural framework for VEDLIoT
• Theory of compositional architectural frameworks
• VEDLIoT framework
(3) Demonstrate
Section 6: Demonstration of the proposed framework on a use case
Section 7: Discussing the relation to requirement engineering
• Proof of concept from real world use case
• Relation to requirement engineering
RQ1
RQ2
RQ3

Fig. 2. Structure of this study including the three parts of the research cycle.

3. Challenges with architectural frameworks for complex systems

We first analysed relevant standards related to architectural frameworks with special attention to those specific for AI systems and IoT systems to derive the challenges with architectural frameworks for distributed AI systems. We complemented this analysis with a literature survey in systems engineering for AI to understand the current state of the art in research. We focused on AI and IoT systems because they represent systems of often high complexity that need to be supported by our proposed compositional approach to an architecture framework. We also conducted a workshop and focus groups with practitioners in order to truly understand and validate the challenges we need to account for in an architectural framework, especially towards our use case for distributed AI systems.

3.1. Surveying existing standards and literature

Based on the architecture ontology and methodology of ISO 42010 (ISO, 2012), IEEE published a standard for architectural frameworks for IoT (IEEE, 2019). The purpose of this standard is to "provide a framework for system designers to accelerate design, implementation, and deployment processes". Good overviews of standardisation attempts for IoT architectures are given by Weyrich and Ebert (2016), Ray (2018), and in the architecture section of Mohd Aman et al. (2020).

IoT is a network of cyber–physical devices and systems, and, although it does not directly address IoT, NATOs architecture framework provides an architectural framework for large distributed systems of intelligent agents (NATO, 2020).

Standardisation of architectural frameworks for AI

In 2021, the international standardisation for architectural frameworks of AI systems is still ongoing. The only published international standard relevant for AI systems is currently ISO/IEC TR 20547, which describes a standardisation of big data reference architectures (ISO, 2020). Table 1 provides an overview of ongoing international standardisation efforts.

Table 1
List of ongoing international standardisation related to architecture frameworks for AI system.

Number	Title	Status
ISO/IEC 5338	AI system lifecycle processes	Preparatory
ISO/IEC 5392	Reference architecture of knowledge engineering	Preparatory
ISO/IEC 5469	Functional safety and AI systems	Proposal
ISO/IEC 23053	Framework for AI systems using machine learning	Under approval
ISO/IEC 24030	Artificial intelligence - Use cases	Under publication
ISO/IEC 24372	Overview of computational approaches for AI systems	In draft

State of the art for AI systems architecture

In a research agenda for engineering AI systems, Bosch et al. (2020) provide a list of challenges when developing architectures for systems with AI components: (i) providing the right (quality of) data used for training, (ii) establishing the right learning infrastructure, (iii) building a sufficient storage and computing infrastructure, and (iv) creating a suitable deployment infrastructure. Indeed, the process of finding the data which provide a ground truth or reference for the AI to train on is often insufficiently documented during requirement engineering (Kondermann, 2013). This is insofar problematic as especially deep learning models depend on a vast amount of ground truth data for training and testing (Ries et al., 2021). Therefore, additional architectural views might be needed to capture the learning perspective of the AI part of the overall system (Muccini and Vaidhyanathan, 2021). Because it might only be possible to detect and correct flaws in an AI systems after deployment, monitoring under operation of the system needs to become part of a suitable deployment infrastructure (Bernardi et al., 2019). Of course, an AI system does not only consist of AI components, but relies also on conventional software and hardware components. The development of AI components and traditional system components must therefore be aligned to avoid unwanted technical debt (Sculley et al., 2015).

However, as Woods emphasises, traditional architecture frameworks, such as the 4+1 architectural view model by Kruchten (1995), does not account for data and algorithm concerns connected to AI component development (Woods, 2016).

There exist approaches to software architecture that treat data as explicit viewpoints, such as the one described in Clements et al. (2011, Chapter 2.6). These data viewpoints provide models and views for the data flow and data storage/management, which is relevant for distributed and connected systems, such as the IoT.

Context diagrams are an established method to capture the operational domain of the system (Woods and Rozanski, 2009), and architectural frameworks can consider them as own viewpoints (Rozanski and Woods, 2012, Chapter 16). However, in a recent study we identified a number of challenges with context definitions for AI systems; for example, we found that they are often not in sync with the system development, and, therefore, often overly conservative, not well integrated into agile workflows, and disconnected from the function developers and other stakeholders (Heyn et al., 2022).

New stakeholders such as data engineers (Vogelsang and Borg, 2019), or governmental agency overseeing the use of AI in society (European Commission, 2020)) must be integrated in the system design and requirement engineering phases (Altarturi et al., 2017). A common platform needs to be found for communicating design decisions between requirement engineers, data engineers and other new stakeholders (Ahmad et al., 2021). The AI part can also causes new concerns to arise during system design, such as a stronger focus on ethical considerations (Aydemir and Dalpiaz, 2018), fairness or explainability. Aspects such fairness (Habibullah and Horkoff, 2021) and explainability (Chazette and Schneider, 2020) can be considered new non-functional requirements, which eventually require new architectural views describing them as quality aspects of the system (Horkoff, 2019). Developing AI components is a hierarchical, yet also iterative task: (i) prepare training data and/or environment, (ii) create a suitable model, (iii) train and evaluate the model, (iv) tune

Table 2
Participants list for the workshop on challenges relevant for an architectural framework for VEDLIoT.

No	Area of expertise	Industry partner	Academic partner
1	Industrial IoT	✓	
2	Smart home		✓
3	Automotive systems	✓	
4	DL optimisation	✓	
5	AI hardware	✓	
6	Requirement engineering		✓
7	IoT and AI research		✓
8	AI systems development	✓	
9	Secure conc. for IoT and AI		✓
10	AI Hardware Research		✓
11	Systems Safety concepts		✓

and repeat training, and finally, (v) deploy and monitor the runtime behaviour of the trained model (Bosch et al., 2020; Wan et al., 2020). Its design needs to be decomposed into different levels of system design, and consistency needs to be ensured in order to satisfy high level requirements (Giaimo et al., 2010), and to fulfil the stakeholders' goals with a system. In addition, the system design must also allow for "middle-out design", where existing components need to be integrated in the overall system design (e.g. transfer-learning from existing AI models or integration of off-the-shelf components) (Murugesan et al., 2019). Murugesan et al. propose a hierarchical reference model, which supports the appropriate decomposition of requirements to the composition of the system's components. In their model the authors define how components can be decomposed into subcomponents. To ensure consistency between the system architecture and the requirements, they define the terms *consistency*, *satisfaction*, and *acceptability*. One major advantage of their model is that, if decomposition of system components is done correctly, these components can be independently specified and developed.

3.2. Identifying challenges of distributed AI system development in VEDLIoT

Two workshops with industry and academic partners from the VEDLIoT project,[1] were conducted with the aim of identifying and validating concerns relevant for a reference architecture framework for distributed AI systems from a practitioner's point of view. VEDLIoT is an excellent candidate for this study, because the aim of the project is to develop methods and tools for the development of distributed systems with deep learning components by using "real world" use cases.

The first workshop took place in February 2021 and eleven participants joined the discussion through the remote conferencing software Zoom. A list of participants is provided in Table 2.

After explaining the aim of the workshop, the participants were presented with fundamental concepts of Architectural Frameworks for the IoT as described in IEEE 2413-2019 (IEEE, 2019). Table 1 of that standard provides a list of stakeholders for IoT systems which was distributed to the participants before the workshop. After inspecting the table, the participants were asked, if additional stakeholders need to be considered when considering IoT systems with AI components. The participants agreed that the list of common stakeholders from the standards contains most relevant stakeholders, and that additional stakeholder in regards to the AI components are data scientists and legislator and/or policy makers who might impose additional rules, e.g. for data privacy, transparency, or explainability of the AI's decisions.

In a second step, we wanted to identify relevant concerns for systems that are part of the IoT and, at the same time, contain AI components. The list of concerns for IoT systems given in Table 2 of IEEE 2413-2019 (IEEE, 2019) was provided to the participants in advance of the workshop. During the workshop, the participants were asked to list all relevant concerns for IoT systems with AI components that are either on the standard's list of concerns, or that are not mentioned by the standard.

The results were collected in a mind-map and, together with the participants, clustered into what we will call "clusters of concern". The resulted mind-map is reproduced in Figure C.12 in Appendix C[2] and can also be found in a repository holding a replication package and additional material.[3] Concern groups that are not already covered by IEEE 2413-2019 (IEEE, 2019) have received an ID in the Figure and are summarised alphabetically in Table 3, together with the challenges identified from literature. The identified concerns were validated in a second workshop with the same participants in March 2021.

3.3. Problem statement

From literature and the workshop we conclude that when combining architectural aspects for IoT and AI systems, as examples for the development of complex systems, many new concerns arise beyond traditional software engineering. Examples of the new concerns are data quality aspects, heuristic AI modelling, AI learning, and even ethical considerations. New stakeholder such as data engineers enter the stage, and common languages or interfaces need to be managed between the different stakeholders. Architectural views, governed through viewpoints, help to capture the different concerns from different stakeholders. However, typical architectural frameworks, such as the ISO 42010 (ISO, 2012) or the IEEE 2413 (IEEE, 2019) standard cannot cope with the large set of architectural views necessary to satisfy all stakeholders' concerns. One reason is, that certain architectural views were not foreseeable at time of creation of the standards, such as views relating to the explainability or other ethical considerations of AI systems. Another problem of current architectural frameworks is the lack of a clear system development hierarchy, which would support the early identification and mapping of dependencies between different architectural views (Nuseibeh, 2001).

Finally, a major challenge we identified in the workshop is the difficulty to keep track of dependencies, e.g. through correspondence rules, between the different architectural views. Table 3 lists the challenges identified for developing AI systems in IoT. In the following, we present our solution to cope with these challenges and in Section 4.2 we discuss how the proposed solution mitigates the identified challenges.

4. The concept of compositional architectural frameworks

This section proposes the concept and artefact of a compositional architectural framework. In the regulative cycle shown in Fig. 1, this section describes the solution design.

The aim is not to provide a viewpoint catalogue or specific architectural models for AI systems, but instead to establish a structure for ensuring consistency and traceability of an architectural framework applicable to, for example, a distributed AI system. Based on ideas from category theory and the identified challenges with architectural frameworks for AI and IoT systems,

[1] A brief description of the VEDLIoT project is given in Appendix B (accessible at https://doi.org/10.1016/j.jss.2022.111604) and additional information can be found in Heyn et al. (2021).

[2] accessible at https://doi.org/10.1016/j.jss.2022.111604

[3] accessible at https://doi.org/10.7910/DVN/VXFFFU

Table 3
Summary of challenges identified from literature (L) and a workshop (W).

ID	Description	L	W	Sources
#1	Additional views needed for describing the AI model		✓	
#2	Data requirements for ensuring the desired AI's behaviour must be considered	✓	✓	Ries et al. (2021), Woods (2016), Kondermann (2013)
#3	Description of context and design domain		✓	
#4	Describing the learning setting environment	✓		Muccini and Vaidhyanathan (2021), Bosch et al. (2020), Woods (2016)
#5	Integration of additional stakeholders (e.g., data scientists, policy makers)	✓		Ahmad et al. (2021), Vogelsang and Borg (2019), Altarturi et al. (2017), Sculley et al. (2015)
#6	Management of dependencies and correspondences between views	✓		Nuseibeh (2001)
#7	New quality aspects (e.g., explainability, fairness)	✓	✓	Habibullah and Horkoff (2021), European Commission (2020), Horkoff (2019), Aydemir and Dalpiaz (2018)
#8	Run Time monitoring	✓	✓	Wan et al. (2020), Bernardi et al. (2019)
#9	Support of decomposition into different levels of system design	✓		NATO (2020), Giaimo et al. (2010), Nuseibeh (2001)
#10	Support of middle-out design	✓		Murugesan et al. (2019)

propositions are derived which provide "rules" towards building a compositional architectural framework. Each proposition is clearly defined and demonstrated with a running example. We call the framework *compositional*, because it is built up from different "modules", called clusters of concern, at different levels of abstraction.

4.1. Compositional architectural framework theory

Fig. 3 exemplifies three aspects of a system: logical behaviour, hardware, and cybersecurity. Each box represents a view governed by an architecture viewpoint on the final system. The rows represent levels of abstraction. For each aspects, the level of details increases with each additional level of abstractions. Let us call the levels, from top (highest level of abstraction) to bottom (highest level of details), analytical level, conceptual level, design level, and run time level. We chose these four "default" levels of abstraction, because we think they represent typical system development phases. However, depending on the type of systems and development regime, additional levels of abstraction can be added or removed, or different names for the levels can apply. We show that by applying four rules, the architectural views for a system-of-interest can be arranged in a matrix, sorted by *clusters-of-concern* and *levels of abstraction*. During system development, different architectural views of the system-of-interest are created to describe different concerns with the system.

4.1.1. Cluster of concern

We identified in Table 3 that for complex systems, such as for example distributed AI systems, many different concerns need to be considered in the final system's architecture. Clustering the concerns into sets is a starting point towards the architectural framework:

> Cluster of concern
>
> *A cluster of concern is a partially ordered set of architectural views which represent a specific concern of the system at different levels of details.*

Definition 1 (*Cluster of Concern*)**.** Let A, B, and C be architectural views forming a set X called cluster of concern. Assume, that B conveys more or equal details about the system than A, and C conveys more or equal details than B. We define a binary relation $\leq$, such that $A \leq B$. It further holds that:

1. $A \leq A$ (reflexivity);
2. if $A \leq B$ and $B \leq C$, then $A \leq C$ (transitivity);
3. if $A \leq B$ equals $B \leq A$, then A and B are of equal level of detail (equivalent) and we write $A = B$.

The pair $(X, \leq)$ is then a partially ordered relation.

Example. The architectural view "computing resource allocation" contains more details about the final system than the architectural view "logical components". The architectural view "function components" contains the least amount of details. These three architectural views form an ordered set of architectural views considering the concern "logical behaviour". Therefore, they form the cluster of concern "Logical Behaviour".

4.1.2. Levels of abstraction

The further the system design proceeds, the more details about the final system become apparent. This increase in level of details during system development allows for defining levels of abstractions. The views are ordered in the sense that the level of detail increases with each level of abstraction:

> Level of abstraction
>
> *The set of architectural views with equivalent level of details about the system-of-interest constitutes a category. A category is a collection of objects which are related to each other in a consistent way (Perrone, 2019). We call the category* level of abstraction*. Architectural views on a level of abstraction are related to each other through morphisms. A morphism can exist only between architectural views on the same level of abstraction.*

Definition 2 (*Level of Abstraction*)**.** The category $\mathbf{C_{la}}$ describing a level of abstraction has architectural views as its objects. Given two architectural views A and B, then $f \in \mathrm{Hom}_{\mathrm{la}}(A, B) : A \to B$ describes the morphisms between architectural views of different concerns at the same level of abstraction. The identity morphism is the identify function mapping an architectural view to itself. Furthermore, given an additional architectural view C, and defining the additional morphism $h \in \mathrm{Hom}_{\mathrm{la}}(B, C) : B \to C$,[4] there exists a morphism $f \fatsemi h$ such that the composition of f and h is $g \in \mathrm{Hom}_{\mathrm{la}}(A, C) : A \to C$.

Example. In the example shown in Fig. 3, a morphism is the correspondence between the logical components and the system

[4] The morphism is called h here instead of the expected g to ensure consistency with later definitions

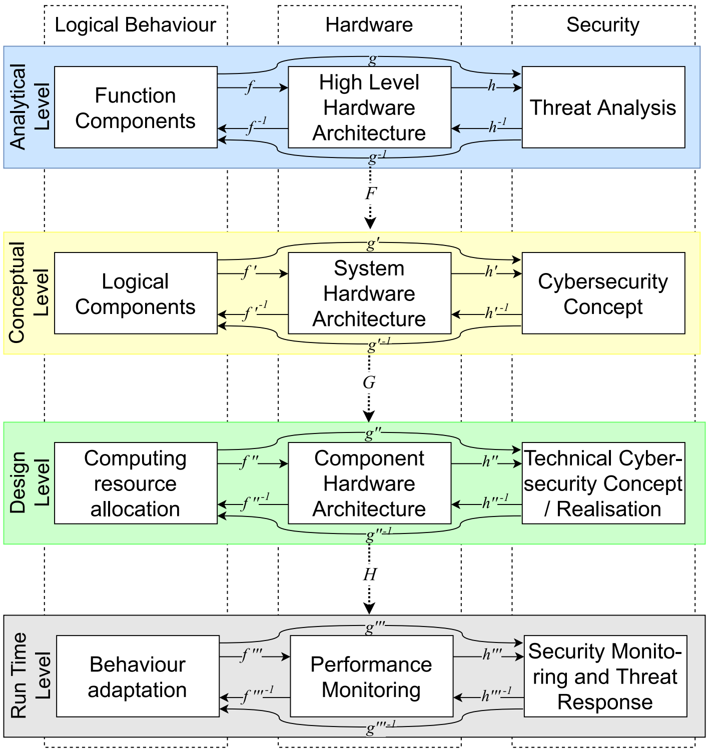


Fig. 3. Order of architecture viewpoints.

hardware architecture on the concept level. This relation can also exist in both directions: The architecture of the system hardware architecture can correspond to the logical components. The morphisms between views that correspond to each other are therefore *isomorphism*, i.e. $f \fatsemi f^{-1} = \text{id}$, where id is the identity relation.

4.1.3. Consistency of an architecture

Describing an architecture through category theory yields the advantage that one can use mathematical tools to show and proof relations between the elements of an architecture description. For example, the product of two objects X and Y in category theory describe "the most efficient way" to have both X and Y. This can be utilised for finding rules on how to combine architectural views into a consistent system architecture description:

Consistency of architectures

If the product over all architectural views on a level of abstraction is valid,[a] *the architectural views consistently describe the system-of-interest.*

[a] Valid means, informally, that no matter which architectural view one "starts at", it is always possible to "transit" via the correspondence rules to another architectural view, and still see the same system (from a different perspective).

Definition 3 (*Consistency of Architectures*)**.** Let $\mathbf{C_{la}}$ be a category and let A and B be two architectural views as objects of $\mathbf{C_{la}}$. The product of A and B consists of a new object Z (this is the product), and two morphisms $\pi_1 : Z \to A$ and $\pi_2 : Z \to B$. The product is valid if the following diagrams commutes[5]:

$$\begin{tikzcd}
& P \arrow[dl, "f"'] \arrow[d, "\varnothing_{f,g}"] \arrow[dr, "g"] & \\
A & Z \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & B
\end{tikzcd}$$

Example. Two architectural views A and B, for example a system hardware architecture and a cybersecurity concept, can be combined to a new view $Z = A \times B$ which unites, in "the most efficient way", the hardware architecture and the cybersecurity concept. P now is any other view, for example the logical components. In addition to the already existing relations f and g between the views, there must be a relation $\phi_{f,g}$ from the logical components P to the newly combined "secure hardware architecture" Z. Furthermore, there must be relations π_1 and π_2, such that one can go always from the "secure hardware architecture" Z back to the origin system hardware architecture A, or to the origin cybersecurity concept B.

The product is valid, if, and only if, it does not matter if one first uses correspondence rules to go from the logical components to the new "secure hardware architecture", and from there to the origin system hardware architecture ($\phi_{f,g} \fatsemi \pi_1$); or if one commutes from the logical components directly to the system hardware architecture (f). If it is not possible to commute between views at the same level of abstraction, one or several views are not describing completely all necessary aspects of the system-of-interest.

4.1.4. Mapping of relations

Each level of abstraction is another category but Definitions 2 and 3 only describe relations between architectural views of the same category (i.e., on the same level of abstraction). However, another concept from category theory called functors allow for mappings between different categories (i.e., different levels of abstraction) that preserve and respect the relations between the objects (Perrone, 2019):

[5] Successfully commuting between different views means to "look" at the system with different architectural views, without encountering inconsistencies in the system.

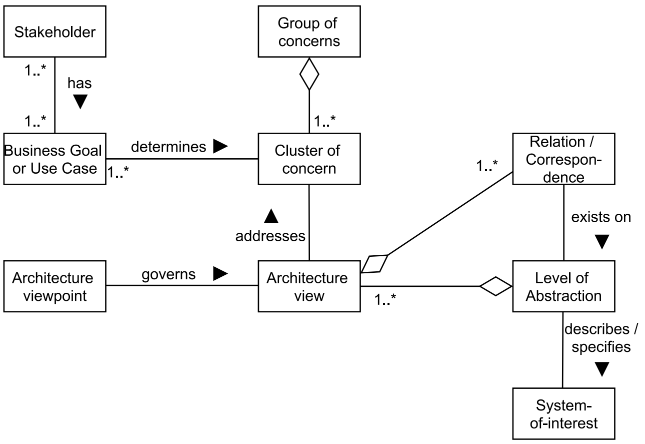


Fig. 4. Conceptual model of a compositional architecture framework.

Mapping of relations

Functors map all views of one level of abstraction to corresponding views of the next lower level of abstraction, and all relations between views to corresponding relations of the next lower level of abstraction.

Definition 4 (*Mapping of Relations*)**.** Let $\mathbf{C_{la}}$ and $\mathbf{C_{lb}}$ be categories representing two levels of abstractions in an architectural description. A functor F from $\mathbf{C_{la}}$ to $\mathbf{C_{lb}}$ maps each object of $\mathbf{C_{la}}$ to a corresponding object of $\mathbf{C_{lb}}$; and maps each morphism between the objects in $\mathbf{C_{la}}$ to corresponding morphisms in $\mathbf{C_{lb}}$. Furthermore, unitality holds, i.e. identities id are mapped into identities, and compositionality holds, i.e. let f and g be morphisms, then $F(f \fatsemi g) = F(f) \fatsemi F(g)$.

Example. If there exists a relation (morphism) between the Cybersecurity Concept and the System Hardware Architecture on the conceptual level, a corresponding relation (morphism) must exists on the design level between the technical cybersecurity concept and the component hardware architecture. Note, however, that the reverse is not necessarily true. If there exists a relation between two views on a lower level of abstraction, this relation does not necessarily exist on a higher abstraction level too.

The final conceptual model of a compositional architecture framework based on the stated definitions is illustrated in Fig. 4. Note that an additional object called "Group of concerns" has been introduced. It serves as a way to order the clusters of concern into different major aspects of the system.

4.2. Connecting the theory to the challenges

Table 3 listed the challenges identified for developing AI systems in IoT. The idea of a compositional architectural framework can solve these challenges as follows:

- *#1: Additional views needed for describing the AI model* — Architectural views describing e.g., the configuration and hyperparameter settings of the AI model can be explicitly taken into consideration through an own cluster of concern.
- *#2: Data requirements must be considered* — Depending on what the data concerns, data aspects can be integrated into views of different clusters of concern (e.g., training data can be handled as an own cluster of concern with views representing the data strategy for AI training). Data concerns regarding communication and information can be separated into another cluster of concern with own architectural views.
- *#3: Description of context and design domain* — The context and operation design domain can be treated as own clusters of concern, thus making the treatment of the context and design domain explicit during system development.
- *#4: Describing the learning setting/environment* — The learning setting can be integrated as an own cluster of concern, with independent architectural views. The views can include objectives of the learning, learning scenarios selection, and a view describing the specific learning settings.
- *#5: Integration of additional stakeholders* — The proposed architectural framework is scalable in the sense that additional clusters of concern can easily be added. This allows for additional stakeholders to add their concerns and (architectural) views for the system in own clusters of concern.
- *#6: Management of dependencies and correspondences between views* — A compositional architectural framework provides rules on how to handle dependencies between architectural views. The existence of correspondence rules in a compositional architecture is limited by Definitions 2 and 4. Specifically, correspondence rules should only exist between architectural views on the same level of abstraction.
- *#7: New quality aspects (e.g., explainability, fairness)* — New (quality) concerns such as explainability can be explicitly integrated in the architectural framework as clusters of concern. The concept of compositionality allows for an easy scalability, because any number of additional clusters of concern can be added to the framework.
- *#8: Run time monitoring* — Run time concerns, such as run time monitoring, can be made explicit with an own level of abstraction, as suggested in the default setup of a compositional architectural framework.
- *#9: Support of decomposition into different levels of system design* — A compositional architectural framework explicitly supports the decomposition into different levels of system design. The number of levels of system design is flexible, and can be adapted to the needs of the system and company.
- *#10: Support of middle-out design* — Middle-out design is supported because it is not required to "fill" the framework with architectural views from top to bottom. For example, it is possible to define a component hardware architecture first (because a certain hardware component must be used in the system), and then build the system hardware architecture "around it". Definition 3 can be used to ensure that no inconsistencies between architectural views on the same level of abstraction exist after finishing the middle-out design.

5. A compositional architecture framework for VEDLIoT

The idea of a compositional architectural framework is tested by studying the artefact in depth in the VEDLIoT project. The VEDLIoT project is a suitable test-ground because it combines conventional systems with deep learning and the IoT. This requires a variety of design decisions with different architecture views on the resulting systems. Additionally, VEDLIoT aims at explicitly supporting all necessary non-functional aspects of the system, such as data privacy, safety, but also non-functional aspects particularly relevant to deep learning such as explainability. The success criteria of the study were that the proposed framework should result in an applicable architecture framework for VEDLIoT, and that guidelines are found on how the theoretical framework, described in Section 4, can be applied in a practical setting.

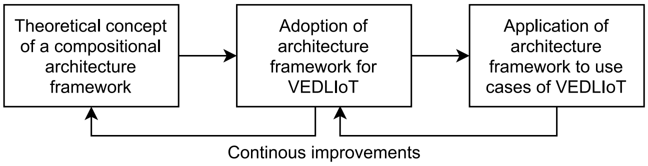

Theoretical concept of a compositional architecture framework
Adoption of architecture framework for VEDLIoT
Application of architecture framework to use cases of VEDLIoT
Continous improvements

Fig. 5. Iterative process of adopting and applying the compositional architecture framework concept to VEDLIoT.

Table 4
Participating roles in the development of the compositional architectural framework for VEDLIoT.

No	Role	Industry partner	Academic partner	Years of experience
1	Researcher		✓	11
2	Research engineer	✓		4
3	Researcher	✓		20+
4	Researcher		✓	30
5	Developer	✓		1
6	PhD Student		✓	3
7	Professor		✓	15
8	Deep learning developer	✓		3
9	Developer	✓		5
10	Project Lead	✓		15
11	Researcher		✓	1
12	Researcher		✓	4
13	Researcher		✓	2
14	Professor		✓	20+
15	Professor		✓	6
16	Developer	✓		10

5.1. Methodology of the case study

Participants from all involved companies and academic partners met in bi-weekly meetings to analyse the use cases, to adopt the architectural framework to the needs of the VEDLIoT project, and ultimately to apply the compositional architectural framework to the use cases. Fig. 5 shows the iterative character of this process: Initially, the participants were introduced to a preliminary version of the theoretical idea of a compositional architectural framework. A first step was to find a general adoption of the architecture framework for VEDLIoT, which was then continuously applied to the use cases. If the theoretical framework or the general adoption did not fit to the use cases, changes in the theoretical concept of the architectural framework would be proposed and implemented. As is common practice in workshops related to solution design,[6] the authors of this article were participants in the meetings, and therefore influenced the development of the architectural framework for VEDLIoT. However, the majority of participants were not involved in the academic observations, and therefore could validate or rebut the ideas brought forward by the authors. A list of the roles and experience of workgroup's participants is given in Table 4. The entire version history and evolution of the VEDLIoT architectural framework can be found in the supplement material.[7]

5.2. Clusters of concern

The challenges of architectural frameworks for distributed AI systems, summarised in Table 3, were used as starting point for discussions on the necessary clusters of concern during bi-weekly meetings of the VEDLIoT partners. First, the group identified four major groups of concerns for the architecture framework. Then, for each of the groups of concern, the participants of the bi-weekly meetings analysed the use cases of VEDLIoT and determined the required clusters of concern. The overall aim was to create as many clusters of concern as the VEDLIoT use cases require (none missing), yet trying to minimise the amount of clusters of concern (no cluster of concern can be removed). The resulting clusters of concern are summarised in Table 5. Note, that other development project might require different clusters of concerns. The decision which clusters are required is based on the use case and business goals, as highlighted in the top row of the framework.

Some of the clusters of concern are especially relevant towards distributed AI systems. For example, the *Context and Constraints* cluster of concern covers views on the system that define the context and limits the design domain. An example of a context viewpoint is described in Rozanski and Woods (2012). For AI systems, it is beneficial, sometimes even required, to explicitly state the desired context and to define views on the constraints and the (operational) design domain of the system. (Ali et al., 2010) for example state, that the desired context is often ignored when defining requirements for a system, and Berry (2022) underlines the need of context information for the assessment of the AI system's performance. In addition, Knauss et al. argue that run time uncertainty can be removed by making the context, in which requirements are valid, explicit (Knauss et al., 2016). The context, in which the system operates, will influence architectural decisions (and vice versa), and thus should be made explicit during the design process.

For distributed AI systems, the concern *AI models* is relevant, because it contains views that describe the setup and configuration of the required AI models. Classification of objects in an optical videostream for example requires a different deep neural network setup then recognising natural language or predicting trajectories of other vehicles. Choosing the right AI model is a design decision which requires suitable views on the AI model in relation to the overall system. Also, the learning strategy of the AI model has paramount impact on the final behaviour of the AI system. Trained with a flawed data sets (e.g. biased data), the behaviour of the AI system will exhibit the flaws learned during the learning process (e.g., the trained system will exhibit a bias). The learning process for the AI model is therefore integral part of the system design process. For VEDLIoT we decided to describe the learning process in two clusters of concerns: The first learning specific cluster of concern, titled *Data Strategy*, contains views that allow for the specification of the collection and preparations of the required training, testing, and run time data. The second learning-specific cluster of concern is titled *Learning*. It covers views that allow for the definition of the learning environment, for example views that describe what the AI model is supposed to learn, and how it can learn, i.e., it can contain views elaborating the optimiser and learning settings for the training phase. The concerns of AI model and learning have many dependencies between each other, which will be expressed through correspondences (morphisms).

Unlike previous architectural frameworks for the IoT, such as IEEE 2413-2019 IEEE (2019), the compositional thinking in the architectural framework allows for co-designing the system to fulfil the explicitly identified quality concerns, such as safety, security, but also energy efficiency and ethical concerns. It means that already early in the system development, correspondences between the views regarding the quality concerns and other views in the architecture description are established. The final system can then be said to be "Safe by design", "Secure by design", "Efficient by design", or "Fair by design". Recent legislation shows that the ethical aspects become a central concern when developing AI systems (European Commission, 2020).

[6] see Hevner and Chatterjee (2010, Chapter 10).

[7] accessible at https://doi.org/10.7910/DVN/VXFFFU

Table 5
Description of clusters of concern in the VEDLIoT framework.

Concern	Description
Behaviour and context (Group)	Aspects that concern the static and dynamic behaviour of the system, as well as the context and constraints for the desired behaviour.
Logical behaviour	Views that are concerned with the static behaviour of the system.
Process behaviour	Views concerned with the dynamic behaviour of the system.
Context and constraints	Contains views on the system that define the context and limit the design domain.
Means and resources (Group)	Contains views on aspects of the system that enable the desired behaviour.
Hardware	Includes views on the hardware architecture and component design of the system.
AI models	Contains views that describe the setup and configuration of the required AI model. Views can include model design, e.g., neural network setup or views detailing the configuration of the AI model.
Data strategy	Views that support collection and selection for training, validation, and run time data of the AI model. Views can describe methods for data creation, data selection, data preparations, and run time monitors of data used by the AI.
Learning	Covers views on the system that allow for defining and setting up the learning environment of the AI model. This can include the definition of training objectives and views that outline the chosen optimiser for training.
Communication (Group)	Contains views of data, connectivity and communication between nodes or components of the desired system.
Information	Accumulates views on the system that model the information and data exchanged in and through the system-of-interest.
Connectivity	Contains views on the means of communication available to the system and its resources.
Quality concerns (Group)	Encompass quality aspects which can be described through non-functional requirements which affect the architecture of the system.
Ethics	Views that regulate ethical aspects, such as fairness or transparency of the system.
Security	Views that ensure the security aspects of the system.
Safety	Contains views governing the safety aspects of the system. The views can stem from standards such as ISO 26262.
Energy efficiency	This cluster of concern contains views ensuring energy efficiency, especially for mobile devices.
Privacy	Here views can be contained that ensure privacy requirements, such as for example requested by regulatory authorities.

Table 6
Description of the levels of abstraction (LoA).

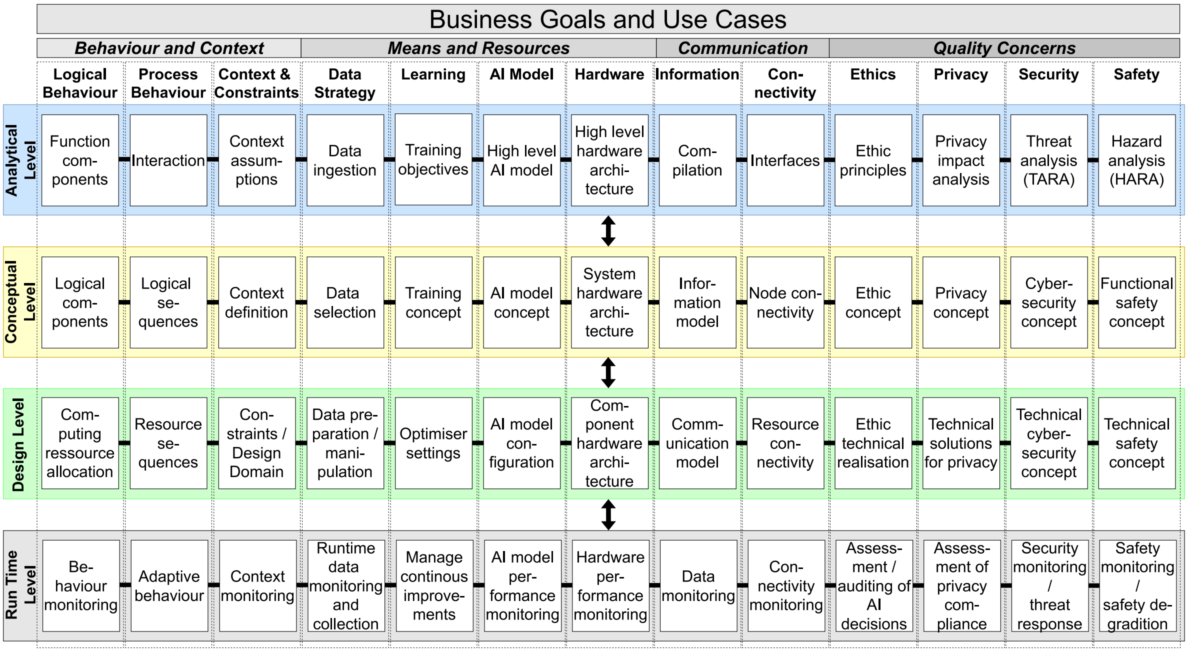
LoA	Description
Analytical level	The first level of abstraction includes architectural views that provide an abstract and high level view on the system-of-interest. On that level, all views provide a way to describe the system and context on a knowledge level, which provides information for further, more concrete system development. For example, the high level AI model view could elaborate on which functions should be fulfilled through an AI.
Conceptual level	On the next level of abstraction, the views provide a more concrete description of the overall system-of-interest. Components are not detailed yet, but the overall system composition becomes clear and the context of operation is clearly defined. For example, the AI model could be concretely shaped as a Deep Learning Network with the required amount of layers. All views on this level combined provide a system specification that sets the system-of-interest in context and elaborates on how the desired functionality is fulfilled.
Design level	The most concrete level at design time of the system is the design level, which includes views that concretely shape the final system-of-interest. Resources are allocated to components, the AI model is configured to work most efficiently in the given environment, and the concrete component hardware architecture is defined. The solution specification describes the final embodiment of the system-of-interest.
Run time level	Complex systems, both AI driven and conventional, often require forms of monitoring and operations control. The purpose of the run time monitoring can be manifold: On one hand, monitoring of a deployed system at run time provides valuable feedback about its performance and reliability to developers and product owners. DevOps is an essential component of an agile development framework, and early detection of issues in a deployed system allows for a swift response from the developers.

5.3. Levels of abstraction

The architectural views are not only sorted by clusters of concern but also by their represented level of abstraction, as was discussed in Section 4.1.

For VEDLIoT, the workgroup decided to follow the four proposed default levels of abstraction introduced in Section 4. They are the *analytical level* providing a high level view, the *conceptual level* providing a more concrete but not too detailed description of the system, the *design level* detailing concrete design decisions, and the *run time level*. Detail descriptions for each level are given in Table 6.

The *run time level of abstraction* is of special interest because some requirements of an AI system might not be exhaustively testable before deployment. Russel describes in his book *Human Compatible: AI and the Problem of Control* the example of an AI algorithm commonly found in social media that maximises *click-through*, i.e., "the probability that the user clicks on the presented items" (Russel, 2020). Russel highlights that such an algorithm not necessarily "presents items that the user likes to click on", but instead could (inadvertently) change the user's behaviour in a manner to make him or her more predictable in his preferences e.g., by favouring extreme political views (Russel, 2020). By constantly monitoring the decisions of the AI algorithm, such deviations from the intended behaviour can be detected and mitigated, e.g., through retraining or by "pulling the plug". Most AI systems are not "adaptive". They are trained and tested with a data set representing the desired context in which the AI system is intended to operate in under the assumption of stationarity in the probability distribution of the data. In reality, the assumption of stationarity of the probability distributions does not hold in most case, for example when the context, in which the AI operates in, can change over time. Concepts like *continual learning* allow the AI to handle drifts in data distributions (Lesort et al., 2021). However, continual learning requires run time monitoring concepts to detect deviations from the currently learned context, and automatic data collection (and labelling) for autonomous retraining of the AI model.



	Business Goals and Use Cases												
	Behaviour and Context			Means and Resources				Communication		Quality Concerns			
	Logical Behaviour	Process Behaviour	Context & Constraints	Data Strategy	Learning	AI Model	Hardware	Information	Connectivity	Ethics	Privacy	Security	Safety
Analytical Level	Function components	Interaction	Context assumptions	Data ingestion	Training objectives	High level AI model	High level hardware architecture	Compilation	Interfaces	Ethic principles	Privacy impact analysis	Threat analysis (TARA)	Hazard analysis (HARA)
Conceptual Level	Logical components	Logical sequences	Context definition	Data selection	Training concept	AI model concept	System hardware architecture	Information model	Node connectivity	Ethic concept	Privacy concept	Cyber-security concept	Functional safety concept
Design Level	Computing ressource allocation	Resource sequences	Constraints / Design Domain	Data preparation / manipulation	Optimiser settings	AI model configuration	Component hardware architecture	Communication model	Resource connectivity	Ethic technical realisation	Technical solutions for privacy	Technical cyber-security concept	Technical safety concept
Run Time Level	Behaviour monitoring	Adaptive behaviour	Context monitoring	Runtime data monitoring and collection	Manage continous improvements	AI model performance monitoring	Hardware performance monitoring	Data monitoring	Connectivity monitoring	Assessment / auditing of AI decisions	Assessment of privacy compliance	Security monitoring / threat response	Safety monitoring / safety degradation

Fig. 6. An architecture framework for VEDLIoT categorising views in different clusters of concern on different levels of abstraction.

5.4. Architectural views

Finally, the workgroup populated the matrix with architectural views for the different clusters of concern at different levels of abstraction. The final matrix of architectural views for VEDLIoT is given in Fig. 6.

Systems using the VEDLIoT toolchain will contain a significant amount of "traditional" system components around the AI components in order to facilitate the desired behaviour. In this paper, we will not detail architectural viewpoints corresponding to these concerns since they are well covered in literature, e.g., in IEEE (2019) or the extensive viewpoint catalogue in Rozanski and Woods (2012, Part III). However, other architectural viewpoints that aim to facilitate the design of AI components for the system are novel and will be the focus of this section. Table F.8 in Appendix F[8] provides a list of viewpoints, which govern architectural views in the architecture framework for VEDLIoT, that we assume to be relevant specifically towards the AI components of the system. Note that, we do not specify pre-defined models or diagrams suitable for the views because it is not the intention of a compositional architectural framework to provide a catalogue of architecture viewpoints and their models. Instead, the proposed framework aims at providing a method for constructing a coherent architectural framework with architectural views defined through the concerns relevant to fulfil all necessary requirements of the use case. It is the choice of the system architects, developers, and other stakeholders which viewpoints and models are suitable to create the views. Section 6 provides some example of diagrams for the AI specific views of one particular use case, but other use cases might want to use different viewpoints with different models.

In the proposed architectural framework, quality concerns are explicitly represented by architectural views on each level of abstraction. From a traditional system architecture point of view, one might question how, for example, a hazard analysis in form of a *Hazard and Risk Assessment (HARA)*, can be constituted an architectural view. Tekinerdogan and Sözer (2011) proposed an approach to explicitly include *quality architectural views* into architectural frameworks to overcome challenges when matching quality concerns with architectural elements representing functional aspects of the system.

The main challenge is that one cannot meaningfully analyse different quality aspects of the system without some prior knowledge of the system's architecture. On the other hand, improving quality aspects often entail changes in the system's functional architecture.

To give an example, we revisit the need of including the hazard analysis in the architectural framework: A work product that serves as input to a hazard analysis in accordance with ISO 26262 is the *Item Definition*. The item definition contains a boundary diagram that depicts the elements of a system which are "in scope" of the safety relevant system. This includes all functional components. Defining which elements are included in the boundary diagram is an architectural decision, and requires information provided through other, mostly functional, views of the system (i.e. there exist morphisms from functional views towards the hazard analysis view). The hazard analysis will return a set of safety goals that the "item" has to achieve. How the safety goals are achieved is conceptualised in the *Functional Safety Concept*. A common solution is to distribute the safety risk through safety decomposition, which often leads to the introduction of redundancies in the system. Again, introducing redundancies is a clear architectural decision, which has direct consequences e.g., for the system hardware architecture (i.e., we create a morphism from the Functional Safety Concept towards the system hardware architecture view).

Fig. 6 presents the compositional architectural framework for VEDLIoT that aims at mitigating the identified concerns for distributed AI systems presented in Section 3.

5.5. Defining a compositional architectural framework

Based on the experience of applying a compositional architectural framework to VEDLIoT, the following guidelines can be provided:

[8] accessible at https://doi.org/10.1016/j.jss.2022.111604

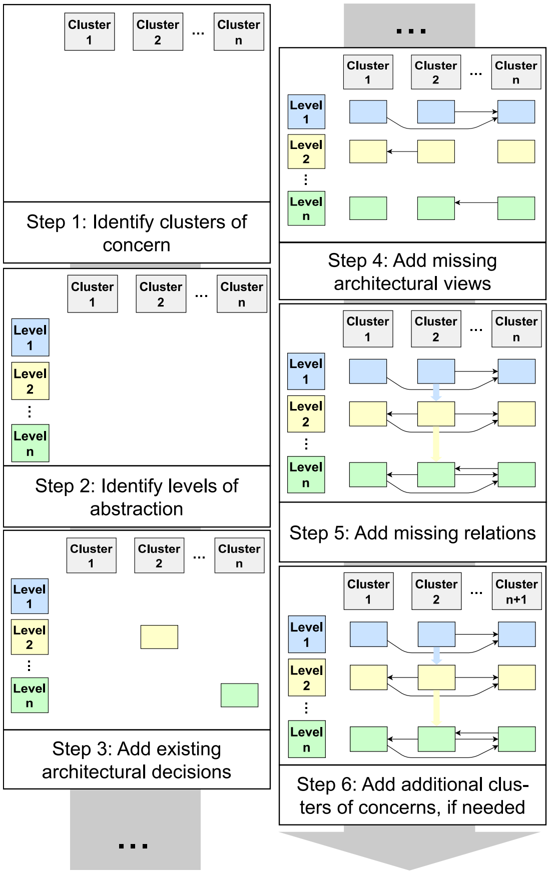

Cluster 1
Cluster 2
...
Cluster n
Step 1: Identify clusters of concern
Level 1
Level 2
Level n
Step 2: Identify levels of abstraction
Step 3: Add existing architectural decisions
Step 4: Add missing architectural views
Step 5: Add missing relations
Cluster n+1
Step 6: Add additional clusters of concerns, if needed

Fig. 7. Steps for defining a compositional architectural framework.

- *Step 1:* Clusters of concern are identified based on the use case and business goals. Initially, larger groups of concerns (such as functionality, hardware, communication, quality) can be defined, which are then refined into atomic clusters of concern (Definition 1).
- *Step 2:* Levels of abstractions are identified. The number of required levels, and the level of detail on each of these levels depend on the size and complexity of system-of-interest and the development settings of the company. Three to four different levels of abstraction seem a good default (Definition 2).
- *Step 3:* Known architectural decisions are entered into the matrix. Most development projects do not start from scratch, but instead have to reuse or integrate into existing architectures. Prior knowledge, such as an existing component architecture, can be entered into the appropriate clusters of concern and levels of abstraction in the architecture matrix.
- *Step 4:* Architectural views are added. Relations (morphisms) are created between the architectural views at each level of abstractions (Definition 2) such that no inconsistencies occur when looking at the system-of-interest from different architectural views (Definition 3).
- *Step 5:* All relations between architectural views must be mapped onto corresponding views of the next lower level of abstraction (Definition 4). If a relation between two architectural views on a higher level of abstraction does not have a correspondence on the next lower level of abstraction, the relation might be unnecessary and can be removed, or a corresponding relation needs to be created.
- *Step 6:* During the system development, additional clusters of concern might be discovered and they are iteratively added.

All steps are illustrated in Fig. 7.

6. Demonstration of the proposed framework on a use case

This section presents the results from applying the proposed compositional architectural framework for VEDLIoT on the development of an automatic emergency braking system, one of the use cases of VEDLIoT. The aim of this demonstration is to answer Research Question RQ3, which asks how a compositional framework can be applied in a realistic context.

Systems that mitigate frontal collisions are considered standard equipment in road vehicles today and standards exists that help design and test these systems (ISO, 2013). However, today's system yet do not exploit all the opportunities given by advanced AI. For example today's systems are very limited in their capability of differentiating what specific object is detected as obstacles, in which context the vehicle is operating in, and how the hardware in the entire vehicle, and beyond on the edge and in the cloud, can be used more efficiently for processing-intensive tasks.

The use case serves as a detailed scenario to demonstrate the utility of the proposed artefact, i.e., the compositional architectural framework. Specifically, the aim is to demonstrate how the different challenges outlined in Table 3 can be solved in a real-world scenario with the proposed compositional approach to an architectural framework. We demonstrate how the logical behaviour, the distribution over different processing notes, and the hardware components can be planned concurrently using the compositional architectural framework approach. The demonstrator also shows how deep learning models can be designed concerted to other design decisions, such as context definition, data strategy, and learning concept. Lastly, we demonstrate how a quality concern such as safety influences other architectural views through morphisms between views. The presented demonstrator entails only a small part of the use case development in VEDLIoT and does not cover the entire system development, because this would be outside the scope of this study. For a more detailed description of the use case development in VEDLIoT, the reader can refer to the VEDLIoT project and its deliverables, such as Meierhöfer et al. (2021).

6.1. Evolution of logical components and hardware architecture

Fig. 8 illustrates a simple example of the idea of co-design using the compositional architectural framework concept. The figure is an extract of the overall system architecture and depicts the logical behaviour design, the hardware architecture, and connectivity. On the highest level of abstraction, the function components view contains four main logical components: Obstacle detection, road characterisation, warning and brake request, and performance monitoring. Also, a high level view of the hardware architecture and connectivity is provided. On the next level of abstraction, the conceptual level, the function components are refined into logical components. Concurrently, the hardware architecture is refined into more details and first correspondences/morphisms are created: E.g., the need to use a visual camera in the system hardware architecture will cause a morphism towards the "find objects in FOV" component, because the required algorithms for finding objects can be technology depending. A concept of connectivity between the hardware components is drafted; it outlines the desired network interfaces and connections. There are now morphisms between the system hardware architecture and the connectivity concept because

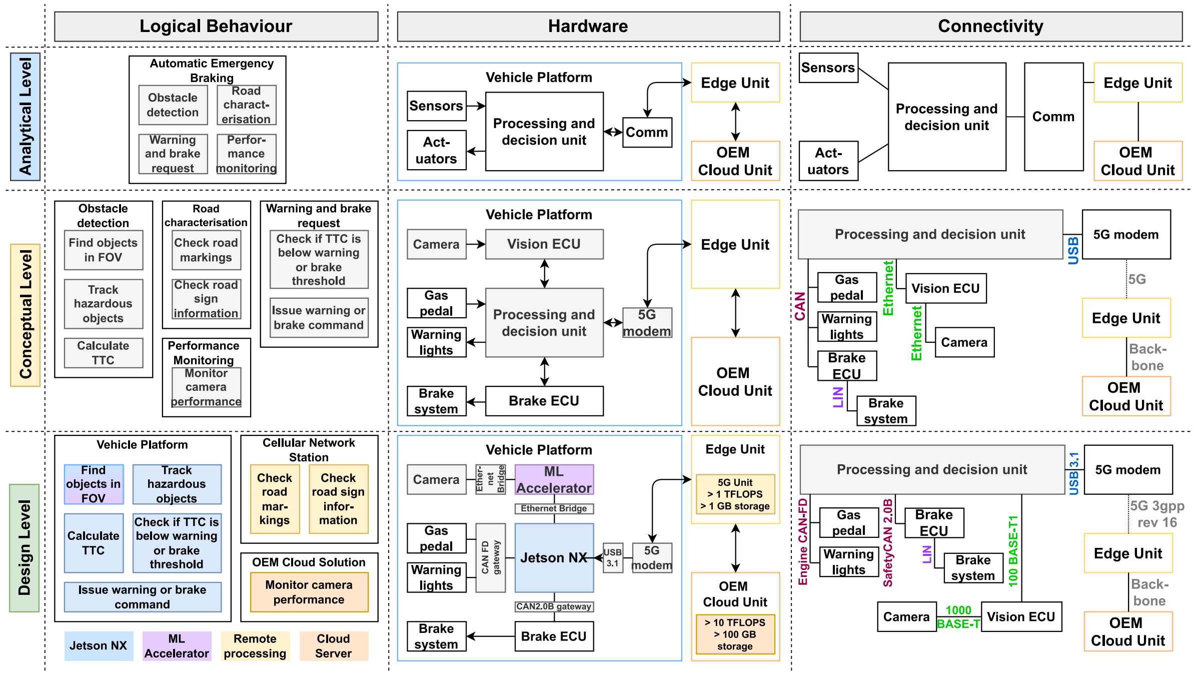


Fig. 8. Co-Design of logical components, hardware design, and connectivity. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

changes of hardware components can require changes in the node connectivity. On the design level of abstraction, the component hardware architecture provides detailed information on the hardware components. This is required, because the computing resource allocation view assigns the logical components to the different available hardware components. Several morphisms exist now between the computing resource allocation view and the component hardware architecture. A consistent system solution can only be obtained if the logical components are mapped to the hardware components represented in the component hardware architecture view. If logical components were mapped to hardware component that do not exist (let us say an additional ARM processor core), the morphism between the component hardware architecture and the computing resource allocation would be broken, and a consistent system solution could not be obtained. Furthermore, knowing the component allocation to hardware allows for specifying bandwidths and latency needs in the design level connectivity view. This example showed how the identified challenges #6: *Management of dependencies* and #9: *Support of different levels of system design*, depicted in Table 3, are solved by the application of the compositional architectural framework.

6.2. Correspondences between context, data strategy, learning, AI model, and hardware

The second example, provided in Fig. 9, illustrates the parallel evolution of the context and constraints of the system, the data strategy, the learning concept, and the AI model. This example demonstrates how the development of an AI component can be seen as a hierarchical process which needs to stay in synchronisation with the remaining concerns of the system development.

On the highest level of abstraction, the context assumptions take direct influence on the required learning objectives (i.e., there exists a morphism from the assumption ("Pedestrians can either be in lane, or on the road but not in the lane, or the road is empty" to the learning object of classification of objects into three classes).

On the conceptual level, the context definition clarifies the earlier context assumptions. This provides input to the data selection view in which feature attributes are specified for the data. By establishing morphisms from the system hardware architecture view, the context definition can also take into account limitation of the hardware, e.g., the mounting position of the camera influences the minimum height of a human (e.g., 1.00 m) to be detected. Now, the AI model can be conceptualised because input data, required output data and objective of the AI model are known.

On the design level, the design domain view provides clear constraints for the system's operability. Furthermore, the learning procedure's and AI model's configuration are set. This also includes a view on necessary data preparations or data manipulations. For example, after some time in operation the camera might suffer from random pixel failures. These pixel failures can be simulated by randomly disturbing pixels in the training data. The result is an AI model that is more robust against random pixel failures. Finally, the run time level provides views that explain which monitoring concepts and run time reconfiguration might be required during operation of the system. We can take the AI model run time view as an example: Two monitors can check the feature map activity in the feature extraction section of the deep neural network, and uncertainty monitoring can be applied to the classification output of the network.

In summary, this example showed how the challenges (see Table 3) #1: *Additional views for AI modelling*, #2: *Consideration of data requirements*, #3: *Context definition*, #4: *Description of the AI's learning environment*, and #8: *Run time monitoring* are mitigated by the architectural framework.

6.3. Example of correspondences between a quality concern and the remaining architecture concerns

This example demonstrates how a quality concern influences via morphisms other architectural views. This is an example on how quality aspects (even novel aspects such as explainability) can be explicitly handled in the architectural framework (Challenge #7 in Table 3). A common quality concern for automotive

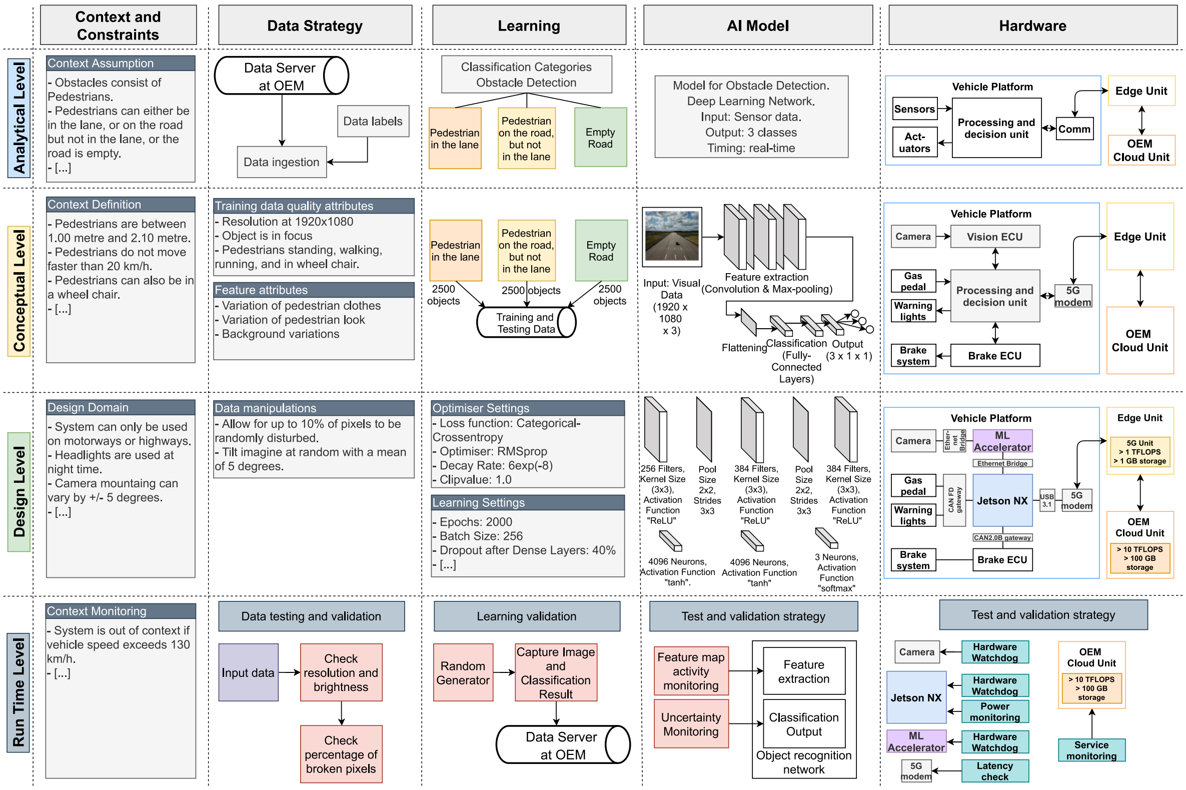
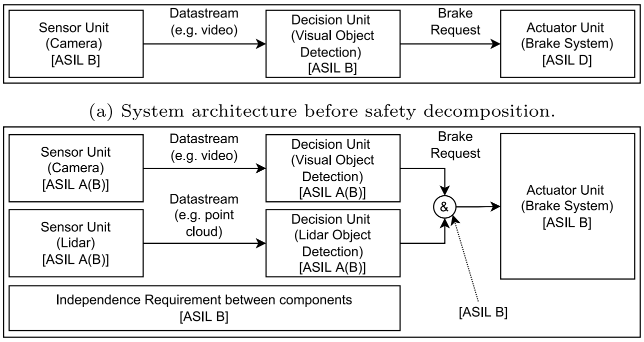

Context and Constraints
Data Strategy
Learning
AI Model
Hardware
Analytical Level
Conceptual Level
Design Level
Run Time Level
Context Assumption
- Obstacles consist of Pedestrians.
- Pedestrians can either be in the lane, or on the road but not in the lane, or the road is empty.
- [...]
Data Server at OEM
Data labels
Data ingestion
Classification Categories Obstacle Detection
Pedestrian in the lane
Pedestrian on the road, but not in the lane
Empty Road
Model for Obstacle Detection. Deep Learning Network. Input: Sensor data. Output: 3 classes Timing: real-time
Vehicle Platform
Sensors
Act-uators
Processing and decision unit
Comm
Edge Unit
OEM Cloud Unit
Context Definition
- Pedestrians are between 1.00 metre and 2.10 metre.
- Pedestrians do not move faster than 20 km/h.
- Pedestrians can also be in a wheel chair.
- [...]
Training data quality attributes
- Resolution at 1920x1080
- Object is in focus
- Pedestrians standing, walking, running, and in wheel chair.
Feature attributes
- Variation of pedestrian clothes
- Variation of pedestrian look
- Background variations
Pedestrian in the lane
Pedestrian on the road, but not in the lane
Empty Road
2500 objects
2500 objects
2500 objects
Training and Testing Data
Input: Visual Data (1920 x 1080 x 3)
Feature extraction (Convolution & Max-pooling)
Flattening
Classification (Fully-Connected Layers)
Output (3 x 1 x 1)
Vehicle Platform
Camera
Vision ECU
Gas pedal
Warning lights
Processing and decision unit
5G modem
Brake system
Brake ECU
Edge Unit
OEM Cloud Unit
Design Domain
- System can only be used on motorways or highways.
- Headlights are used at night time.
- Camera mountaing can vary by +/- 5 degrees.
- [...]
Data manipulations
- Allow for up to 10% of pixels to be randomly disturbed.
- Tilt imagine at random with a mean of 5 degrees.
Optimiser Settings
- Loss function: Categorical-Crossentropy
- Optimiser: RMSprop
- Decay Rate: 6exp(-8)
- Clipvalue: 1.0
Learning Settings
- Epochs: 2000
- Batch Size: 256
- Dropout after Dense Layers: 40%
- [...]
256 Filters, Kernel Size (3x3), Activation Function "ReLU"
Pool Size 2x2, Strides 3x3
384 Filters, Kernel Size (3x3), Activation Function "ReLU"
Pool Size 2x2, Strides 3x3
384 Filters, Kernel Size (3x3), Activation Function "ReLU"
4096 Neurons, Activation Function "tanh".
4096 Neurons, Activation Function "tanh"
3 Neurons, Activation Function "softmax"
Vehicle Platform
Camera
Ether-net Bridge
ML Accelerator
Ethernet Bridge
Gas pedal
Warning lights
CAN FD gateway
Jetson NX
USB 3.1
5G modem
CAN2.0B gateway
Brake system
Brake ECU
Edge Unit
5G Unit > 1 TFLOPS > 1 GB storage
OEM Cloud Unit
> 10 TFLOPS > 100 GB storage
Context Monitoring
- System is out of context if vehicle speed exceeds 130 km/h.
- [...]
Data testing and validation
Input data
Check resolution and brightness
Check percentage of broken pixels
Learning validation
Random Generator
Capture Image and Classification Result
Data Server at OEM
Test and validation strategy
Feature map activity monitoring
Uncertainty Monitoring
Feature extraction
Classification Output
Object recognition network
Test and validation strategy
Camera
Hardware Watchdog
Jetson NX
Hardware Watchdog
Power monitoring
ML Accelerator
Hardware Watchdog
5G modem
Latency check
OEM Cloud Unit
> 10 TFLOPS > 100 GB storage
Service monitoring

Fig. 9. Co-Design of context and constraints, data strategy, learning concept, AI model, and hardware.

systems is safety. Assume the system shall trigger the brakes whenever a person is detected in the lane in front of the vehicle. Assume further that, through the HARA, the following safety goal has been identified: *"The system shall not trigger the emergency brake unintentionally (ASIL*[9] *B)"* An extract of the high-level system architecture is illustrated in Fig. 10(a). While the brakes are designed to a high safety integrity level, the camera and object detection algorithm might not be able to achieve the required ASIL B. Therefore, a safety decomposition in accordance with ISO 26262 results in redundancy in the sensing system, and a lower ASIL on each component: In the functional safety concept, an additional lidar sensor, together with a second object detection algorithm specifically designed for detecting objects in lidar point clouds, allow for the reduction of the required safety integrity level of all redundant components to ASIL A(B). The additional sensing system must be independent from the first object detection algorithm. The final high level system architecture after safety decomposition is illustrated in Fig. 10(b).

By introducing the safety decomposition in the functional safety concept, correspondences (morphisms) to the system hardware architecture view and the logical components view were established. On the next level of abstraction, the technical safety concept establishes a view on the overall system's architecture that allows the fulfilment of the functional safety concept. For example, the technical safety concept provides a view on the system architecture that requires the logical component "Visual object detection" to be deployed on safety certified hardware components, which creates a correspondence to the computing resource allocation view; or that the object detection algorithm only works at daylight, which creates a correspondence to the constraints/design domain view.


Sensor Unit (Camera) [ASIL B]
Datastream (e.g. video)
Decision Unit (Visual Object Detection) [ASIL B]
Brake Request
Actuator Unit (Brake System) [ASIL D]
Sensor Unit (Camera) [ASIL A(B)]
Datastream (e.g. video)
Decision Unit (Visual Object Detection) [ASIL A(B)]
Brake Request
Sensor Unit (Lidar) [ASIL A(B)]
Datastream (e.g. point cloud)
Decision Unit (Lidar Object Detection) [ASIL A(B)]
&
Actuator Unit (Brake System) [ASIL B]
Independence Requirement between components [ASIL B]
[ASIL B]

(a) System architecture before safety decomposition.

(b) System architecture after safety decomposition.

Fig. 10. Safety decomposition in the system architecture for an automatic emergency brake system.

A major advantage of the compositional architectural framework is the ability to trace morphisms (i.e., links or correspondences) between the different architectural views. Assume, that after years of service the system's hardware shall be upgraded and it is decided to replace the two independent processing units with a more powerful single unit. Then, the morphism between the system hardware architecture and the functional safety concept reminds the system designer of the safety concern that triggered the original design decision of having separated and independent processing units for the two independent object detection algorithms, even years after the original development.

6.4. Feedback from industry partners in vedliot

In this article, we presented only one of four use cases to which the compositional architectural framework approach was

[9] Automotive Safety Integrity Level.

applied to. The other use cases were a fault detection system for high voltage switching, a system for electric motor condition classification, and a smart mirror as part of a smart home setup. Because it is beyond the scope of this article to detail all demonstrator, we used semi-structured interviews to collect feedback on the usability, the consistency of the compositional architecture framework for VEDLIoT, as well as feedback on possible improvements, how well the approach fits with their use case, and comparisons to current system architecture frameworks applied in the company. We interviewed four use case developers, one from each use case. The interviews allowed us to gain opinions and feedback from participants who had experienced the compositional architecture framework and to explore yet un-identified issues with the subject (Hancock, 2006). The interviews contained three sets of questions: (I) Questions about the interviewee's role and experience with architecting complex systems; (II) A set of feedback questions on the usefulness and applicability of the compositional architecture framework; (III) The role of run time monitoring in the VEDLIoT systems and their architectures. The interview guide with all questions is available in the replication package of this article.[10] Due to data privacy reasons, contact information to the use case owners is not included in this article. But general contact information to the companies can be found on the website of the VEDLIoT project.[11]

Background of the interviewees. The four interviewees have all a background in system architecting with different years of experience. One interviewee was a PhD candidate in charge of developing a prototype system, two interviewees work as project leader and system developer for a multinational company ($>$10.000 employees) with 4 and 10 years of experience, and one interviewee is a research specialist for system design in an automotive supply company ($>$1.000 employees) with more than 20 years of experience in system development.

Challenges solved through the architectural framework. We presented the challenges discussed in Section 3 to the interviewees and asked which of the challenges they encountered and in what degree the architectural framework helped mitigating them. All interviewees mentioned that the explicit treatment of quality aspects as part of the architectural framework was helpful. It eased finding trade-offs between competing quality aspects, and it allowed for better cooperation between different teams dealing with different quality aspects of the system.

Having explicit cluster of concerns for data strategy and learning helped all partners in defining data sets and training configurations. One partner mentioned, that, in contrast to previous in-house processes they used, the architectural framework established a much better connection between the context description, the hardware architecture, the AI model and the data sets used for training. Establishing connections early between the context and the data strategy reduced the need for data creation in a laboratory, which reduced the costs of the project.

We asked in what degree the guidelines of the architectural framework helped them in finding the right architecture for their system, compared to other approaches or processes they usually apply in their respective company. A common answer was that, compared to previous architectural framework they used, the guidelines helped in keeping an overall structure and overview of the system development. The differentiation into different level of abstractions prevented that design decisions are taken too early, which could have created boundaries in the later system development. Also, one interviewee highlighted that the proposed framework fit better into an agile development environment, because the guidelines can help different teams in keeping a consistent architecture.

Furthermore, according to two interviewees, the guidelines established an easier traceability of design decisions, which helped in documenting causes for architectural decisions and in fulfilling documentation requirements for certification (e.g., for fulfilling safety standards, or ethical aspects of AI as governed by new EU regulations) of the products.

All use case owners agreed that the explicit treatment of run time behaviour in the architectural framework is helpful. One use case owner used the views in the run time level to design explicit feedback mechanisms allowing the user to report on the run time experience of the system. Two use cases created run time views that ensured that legal regulations are met and controlled at run time.

Missing aspects or negative experiences. We asked if any information or aspects were missing in the architectural framework. One interviewee mentioned that it was difficult to decide on the importance of the cluster of concerns and where to start the system design process. The interviewee suggested to highlight those clusters of concern that are most likely compulsory for each system development (e.g., logical behaviour, context and constraints, and hardware) and use them as starting point for the system development. Another use case owners said he wished for more software architecture concerns, such as views on the operation system and middleware. Furthermore, the same use case owner said that, although the guidelines helped significantly in establishing traceability of design decision for certification of e.g., safety aspects, an explicit treatment of certification and standardisation aspects in the framework would be appreciated. Lastly, one use case owner mentioned that a cluster of concern could be established on human–machine-interfaces, containing views on the interaction with the users or operators of the system. All use case owners agree that some form of tool support for the architecture framework would be highly appreciated.

7. Discussing the relation to requirement engineering

The three examples of applying the architectural framework given in Section 6 show an interdependence between the systems engineering and requirements engineering. For example, the functions selected for deployment on the Jetson NX (highlighted in blue on the Design Level in Fig. 8) set performance requirements on the processing unit (Jetson NX). Indeed, based on the twin peaks model, Nuseibeh (2001), emphasise in their hierarchical requirements reference model the importance of the interrelation between system design and requirements. We realised that the compositional architectural framework provides a refinement structure, which complements and supports requirement engineering.

7.1. Traceability of design decisions

Based on the feedback from the use case owners on applying the compositional architectural framework in VEDLIoT, we learnt that the framework helped in establishing traceability in design decisions through morphisms between the architectural views. Fig. 11 shows how the relations between the architectural views evolved for the example discussed in Section 6.2. It illustrates that design decisions made in one architectural view can cause requirements on elements in another architectural view. Through formalising the correspondences in the architectural framework, traceability of the design decisions can be established. Therefore, it extends existing architectural frameworks, such as Viewpoints + Perspectives (Rozanski and Woods, 2012), by providing a mathematical foundation to establish traceability of design decisions in the architectures through the rules outlined in Section 4.

[10] accessible at https://doi.org/10.7910/DVN/VXFFFU

[11] www.vedliot.eu

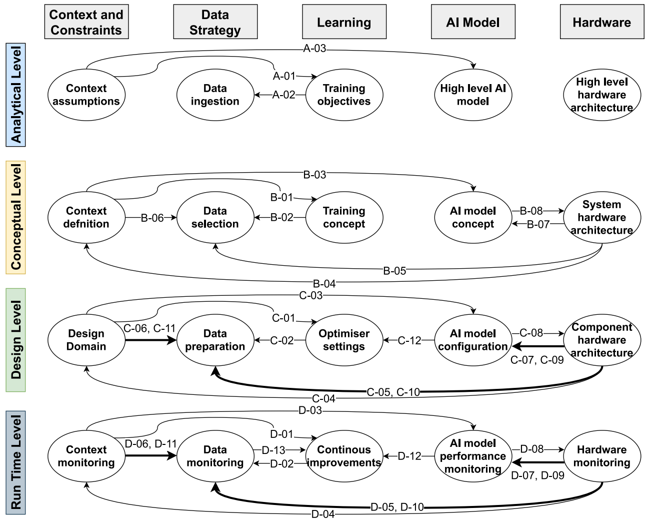


Fig. 11. Directed graph illustrating the evolution of relations between architectural views in the compositional architectural framework. A table of all relations for this example is available in Appendix G[12]

7.2. Support of middle-out design

Traditionally, requirement engineering would be organised in a top-down fashion. However, the architectural framework supports middle-out systems engineering, which is a widely common practice, combining traditional top-down systems design with integration of designated lower level hardware, software, AI models, or other components (Davis and Yen, 2019). The need for middle-out design has also been identified as Challenge #10 in Table 3. Knowledge can become available on all levels of the architectural framework at any time. For example, a new desired function shall utilise an existing AI model and run on a predefined hardware platform. Thus, the aim of requirement engineering is to ensure completeness of missing views on a conceptual or analytical level based on existing design level views.

Another example is that of user stories: User stories, can mix problem and solution descriptions. For example, *"as a driver, I want the vehicle to brake automatically so that I do not hit an obstacle on the road"*, transmits a problem to be solved. However, *"as product owner, I want the automatic emergency braking function to execute on an existing hardware platform"*. transmits a solution description. Because the compositional architectural framework supports middle-out development, the solution descriptions would influence the design level, while the problem descriptions affect the analytical level of the architecture description.

7.3. Defining the scope of the system

The concrete design domain of a solution is found by iterating between the problem and solution space. The analytical level of the architectural framework accumulates information about the problem space. The further the design proceeds, the more information from a solution space perspective is added to the system architecture and its design domain. This became evident in the example shown in Section 6.1: Concrete hardware solutions which are part of the solution space are defined only on the design level of the architecture (e.g., the selection of a Jetson NX platform in Fig. 8). The conceptual level is mostly, and the analytical level is completely hardware agnostic and therefore they represent information from the problem space. Finally, in the design phase, it mostly will be the solution space that provides additional information. As an example, assume a certain hardware component can only operate correctly in a temperature range of −30 degrees to 60 degrees. The component hardware thus limits the design domain in regards of allowed temperature range. This would also create a correspondence to the *"Constraints/Design Domain"* view in the *"Context & Design"* cluster of concern, and thus would make this constrain, and its relation to the hardware component, explicit in the architecture (Need of context and design domain descriptions, Challenge #3 in Table 3).

7.4. Ensuring the desired behaviour of an AI system

The behaviour of an AI system depends on the available data, and the context. Rao et al. (2021) therefore propose to reference the context and data definitions together with the use cases. We saw in the use case demonstration in Section 6.2 that all elements of the information model proposed by Rao et al. can be represented by the compositional architectural framework: Context, and Context Elements are described in the *Context and Constraints* cluster of concern; data requirements and data sources are described through *Information* cluster of concern, and the Learning cluster of concern; and quality attributes are represented through the group of *Quality* clusters of concern. Explicit knowledge of the necessary data quality and context of operations is a prerequisite for defining run time monitors, which was depicted as Challenge #8 in Table 3.

7.5. Defining quality goals

Quality goals can be distinguished into quality attributes (such as performance requirements, or specific quality requirements) and constraints (Glinz, 2007). A first step to determine which quality attributes need to be represented in the architectural framework explicitly in the form of clusters of concern, a quality grid analysis can be formed (Lauesen, 2002). The quality grid is essentially a table with relevant quality attributes in the first column and then an indication on whether this attribute is more or less important than in comparable products. Quality attributes that rank high in this analysis can then be made a cluster of concern in the compositional architectural framework. By representing a quality attribute as explicit quality concern in the architectural framework, the fulfilment of that quality attribute becomes part of the overall system design process (i.e., safety-by-design, security-by-design, etc.). This eases the inclusion of novel quality concerns, such as explainability and the integration of additional stakeholders, such as social scientists (ethical requirements) or public authorities (privacy requirements) in the system development (Challenges #5 and #7 in Table 3). Their concerns are explicitly reflected as clusters of concern in the architecture of the system.

Open targets for quality requirements

An additional advantage of including quality attributes as concerns with different levels of abstraction is the ability to support open metrics, or open targets. According to Lauesen (2002), it is good practice to defer the selection of metrics and its target values for quality attributes to a late stage in the systems development. This strategy avoids a false sense of accuracy and eventually over-designed solutions. The quality concerns can contain target values as late as in the design level of the compositional architectural framework. At the highest level, the analytical level, it is only important to understand what quality attributes are relevant for the general system design.

[12] accessible at https://doi.org/10.1016/j.jss.2022.111604.

8. Conclusion

Answer to RQ1: Which challenges are relevant when defining system architectures for AI systems?

The article outlined challenges and concerns when developing systems that can include AI components. Together with industry partners, these concerns were collected for distributed systems with deep learning components. Specifically, the empirically identified challenges include concerns related to data quality aspects, AI modelling, or setting up a correct learning environment for the AI. Additionally, existing quality aspects such as security and safety have to be combined with novel aspects such as ethical considerations or explainability. New stakeholders enter the stage of systems engineering, such as data engineers which bring their own views (on the system) and language. This creates a huge variety of different architectural views, for which existing standards for architectural frameworks are not sufficient. Especially identifying and mapping of dependencies between the architectural views has been identified as a critical challenge.

Answer to RQ2: What guidance can compositional thinking provide to overcome these challenges for the design and management of architectures for AI systems?

For the purpose of defining and managing architectural views, a theoretical approach to a compositional architectural framework based on ideas from category theory was proposed. Four suggested propositions can be translated into rules that provide guidance for the creation and management of a compositional architectural framework, see the *Rules for compositional architecture* box on this page.

Unlike previous approaches to architectural frameworks for the IoT and AI systems, such as the IEEE 2413 standard (IEEE, 2019), the proposed idea for compositional architectural frameworks allow for system architectures that are scalable to the number of architectural views required for complex systems such as distributed AI systems. Furthermore, it provides guidance for managing dependencies between the architectural views, something that for example the 4+1 view model by Kruchten (1995) is not considering. Rules based on mathematical considerations help in organising different views with different levels of details, keeping consistency between all views, and they help in ensuring traceability of design decisions. This extends existing architectural frameworks such as Viewpoints + Perspectives (Rozanski and Woods, 2012). We showed how the idea of a compositional architectural framework supports requirement engineering efforts for complex systems by explicitly supporting context descriptions of the system, middle-out system engineering, and any number of required quality goals.

Answer to RQ3: How can a compositional framework be defined and applied in a realistic context?

By following the proposed rules, a workgroup of 16 participants from academia and industry created a compositional architectural framework for the VEDLIoT project. The group started by identifying the necessary clusters of concern, levels of abstractions, and finally necessary architectural views for VEDLIoT. The created framework for VEDLIoT was demonstrated on a use case from the automotive industry. The demonstration showed that the framework can handle different architectural views for concerns such as logical behaviour, hardware, context, data strategy, learning environment, AI model, and safety aspects. The compositional framework integrates and complements requirements activities in a favourable way, thus supporting middle-out design and decomposition of requirements for AI systems. The feedback from four use case owners who applied the architectural framework in their projects was favourable. They highlighted that, compared to previously used frameworks and processes, the compositional architecture framework helped their development in keeping a more structured and better overview of the architecture, in finding trade-offs easier between quality aspects, in establishing traceability of design decisions, and in supporting an agile development of the system architecture.

Rules for compositional architectures

A cluster of concern is partially ordered set of architectural views which represent a specific concern of the system at different levels of abstraction.

Rule 1: Clusters of concern shall contain architectural views with different levels of details of a certain aspect of the system-of-interest.

The set of architectural views with equivalent level of details about the system-of-interest constitutes a category called level of abstraction. Architectural views on a level of abstraction are related to each other through morphisms. Morphisms exist only between architectural views on the same level of abstraction.

Rule 2: Architectural views shall be sorted into levels of abstraction. Views on the same level of abstraction shall have an equivalent level of details about the system-of-interest. Correspondences shall exist only between architectural views on the same level of abstraction.

If the product over all architectural views on a level of abstraction is valid, the architectural views consistently describe the system-of-interest.

Rule 3: By using correspondence rules, it shall be possible to arrive at different architectural views of the system-of-interest without encountering inconsistencies.

Functors map all views of one level of abstraction to corresponding views of the next lower level of abstraction, and all relations between views to corresponding relations of the next lower level of abstraction.

Rule 4: Architectural views, and relations between them, on a higher level of abstraction shall be mapped onto corresponding views and relations on the next lower level of abstraction.

8.1. Threats to validity

While we assume that the theoretical idea behind the compositional architecture framework is valid due to the mathematical definitions of the underlying principles, the application of these principles to VEDLIoT can contain threats to validity. First of all, VEDLIoT concentrates on the application of deep learning in distributed systems, which only represents one of many possible applications of the compositional framework. Furthermore, the composition of the workgroup and focus groups involved in the creation and discussion of the architectural framework for VEDLIoT contains mostly people from both academia and industry dealing with advanced research projects. Out of 16 participants, four participants work as developers for systems with deep learning in industry. While all other participants have significant knowledge in system architecture design, system development, and requirement engineering, experience "from the field" of deep learning development could only be provided by 25% of the participants. This potentially can be a cause of mismatch in the case study. By conducting review interviews with additional use case

Table 7
Additional suggestions for validation, including hypotheses and actions.

ID	Description	Relates to section
1	The construction of the proposed architecture framework followed a clearly defined research methodology described in Fig. 2. The framework is built on awareness on the state of the art through a survey of published literature and standards. Furthermore, we identified the state of practice by conducting workshops within the VEDLIoT project, because the aim of the case study was to develop an architecture framework suitable for VEDLIoT. Validating the awareness of the state of practice outside of VEDLIoT requires empirical data from practitioners external to the project.	3.2
→	**Hypothesis 1:** The state of practice and specifically the challenges identified within VEDLIoT can also be representative for other project. **Action 1:** Validate the state of practice with practitioners external to the VEDLIoT project.	
2	A mapping between the theory of compositional architecture frameworks and the identified challenges of constructing systems within the VEDLIoT project is derived from both the state of the art as defined through literature and standards, and empirical data collected within VEDLIoT. The identified challenges themselves, and the mapping towards the theory of compositional architecture frameworks need to be validated outside of the VEDLIoT project in order to establish better generalisability of the mapping.	3.2, 4.2
→	**Hypothesis 2:** The mappings between the challenges and the theoretical framework can also be applied to other projects. **Action 2:** Validate the mapping between the theory of compositional architecture frameworks and the challenges through independent experts. A possible experiment setup could take the form of different surveys that validate the challenges listed in Table 3 and elicits additional challenges. An additional survey can then check the mapping between the theory and the challenges.	
3	In order to validate the instantiation of the compositional architecture framework within the VEDLIoT project, the participants of the project are most suitable, because they are the ones knowing the needs of the project and, therefore, they can assess the applicability and usefulness. This can depend on prior knowledge and usage of architecture frameworks, and therefore different projects might evaluate the applicability and usefulness of the proposed approach differently.	5.5, 6.4
→	**Hypothesis 3:** The applicability and usefulness of the proposed approach to apply compositional architecture frameworks in system design can also be shown in other projects. **Action 3:** Instantiate the compositional architecture framework to further use cases and validate the success of this instantiation with the practitioners responsible for the new use cases based on their prior usage of architecture framework.	
4	For validating the generalisability of the guidelines on how the theoretical framework is instantiated, independent of the case company and VEDLIoT project, there is the need of involvement of practitioners outside the project.	5.5
→ →	**Hypothesis 4:** The guidelines on how to build a compositional architecture framework can be transferable to other projects. **Action 4-a:** Apply the guidelines towards building a compositional architecture framework to another case. **Action 4-b:** Scrutinise each guideline by external experts, both via questionnaires and interviews to collect quantitative and qualitative aspects that might lead to identify risks and benefits of each guideline.	
5	For evaluating the applicability of the compositional approach towards architecture frameworks in a realistic context, there is a need of experts of the domain of that context. In Section 6, we demonstrated the applicability on the development of an automatic emergency braking system. We used experts from the automotive domain to apply and validate the approach. Arguing for the applicability of the approach in other contexts requires additional use cases.	6
→	**Hypothesis 5:** A compositional architecture framework built on the described theory can also be applied in other realistic contexts outside the automotive industry. **Action 5:** Apply the proposed approach towards compositional architecture frameworks in additional realistic contexts outside of VEDLIoT.	

owners, we received feedback on the utility of the framework by additional industry practitioners. The interviews were also an effort to reduce the confirmation bias by including participants not involved in the creation of the framework.

8.2. Additional steps towards validation

The theoretical model of a compositional architectural framework was tested in practice through a case study conducted within a company from the automotive industry in the context of distributed deep learning systems. Additional practitioners outside of the case company confirmed through interviews the usefulness of the compositional approach described through the proposed mathematical model towards defining system architectures. Further validation steps can help in increasing external validity of the findings presented in this article. Especially selection bias can be a concern because we only applied the proposed mathematical concept of compositional architectures to one concrete case in the context of automotive distributed deep learning. Table 7 details additional validation steps, including references to different sections of this article and proposed actions for validation.

8.3. Further research

Besides the additional steps towards validating the proposed theoretical contribution to compositional architecture frameworks, further research can be conducted in how to apply category theory to system and software architectures. Category theory is a very broad field of mathematics, which is why it is difficult to find a starting point to develop ideas from it for a given problem. However, looking at the concepts of profunctors and monoidal categories, there are many interesting concepts to be found in applied category theory that could solve problems we today face in requirement engineering and systems engineering. This paper gives a starting point by introducing categorification to an architectural framework. The article's main focus was on evaluating this starting idea by creating a compositional architectural framework for systems with deep learning components. The VEDLIoT project includes an open call with the aim to apply, among other tools, the VEDLIoT toolchain to new use cases. The open call can serve to further validate the idea of a compositional architectural framework and to collect best practices on how to work with and apply the VEDLIoT architectural framework to new use cases. Further research is proceeding in developing software tools that support the ideas of a compositional architectural framework. One such tool could be built upon the idea of using textual architectural description and a distributed version control system such as *git*, see for example Knauss et al. (2018).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

A link to the data stored at Harvard's Dataverse is contained in the manuscript.

Acknowledgements

This project has received funding from the EU's Horizon 2020 research and innovation program under grant agreement No 957197 (vedliot). This work was also supported by the Centre of EXcellence on Connected, Geo-Localised and Cybersecure Vehicles (EX-Emerge), funded by the Italian Government under CIPE resolution n. 70/2017 (Aug. 7, 2017). We are thankful to all interviewees and companies who supported us in this research.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.jss.2022.111604.

References

Ahmad, K., Bano, M., Abdelrazek, M., Arora, C., Grundy, J., 2021. What's up with requirements engineering for artificial intelligence systems? In: 2021 IEEE 29th International Requirements Engineering Conference (RE). IEEE, pp. 1–12.

Ali, R., Dalpiaz, F., Giorgini, P., 2010. A goal-based framework for contextual requirements modeling and analysis. Requir. Eng. 15 (4), 439–458. http://dx.doi.org/10.1007/s00766-010-0110-z.

Altarturi, H.H., Ng, K.-Y., Ninggal, M.I.H., Nazri, A.S.A., Abd Ghani, A.A., 2017. A requirement engineering model for big data software. In: 2017 IEEE Conference on Big Data and Analytics. ICBDA, IEEE, pp. 111–117.

Awodey, S., 2010. Category Theory. In: Oxford Logic Guides, Oxford University Press.

Aydemir, F.B., Dalpiaz, F., 2018. A roadmap for ethics-aware software engineering. In: 2018 IEEE/ACM International Workshop on Software Fairness (FairWare). IEEE, pp. 15–21.

Bakirtzis, G., Subrahmanian, E., Fleming, C., 2021. Compositional thinking in cyberphysical systems theory. Computer 54 (12), 50–59.

Bernardi, L., Mavridis, T., Estevez, P., 2019. 150 successful machine learning models: 6 lessons learned at Booking.com. In: Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining. pp. 1743–1751. http://dx.doi.org/10.1145/3292500.3330744.

Berry, D.M., 2022. Requirements engineering for artificial intelligence: What is a requirements specification for an artificial intelligence? In: International Working Conference on Requirements Engineering: Foundation for Software Quality. Springer, pp. 19–25.

Bosch, J., Crnkovic, I., Olsson, H.H., 2020. Engineering AI systems: A research agenda. arXiv, arXiv:2001.07522, URL http://arxiv.org/abs/2001.07522.

Censi, A., 2017. Uncertainty in monotone codesign problems. IEEE Robot. Autom. Lett. 2 (3), 1556–1563.

Chazette, L., Schneider, K., 2020. Explainability as a non-functional requirement: challenges and recommendations. Requir. Eng. 25 (4), 493–514.

Cleland-Huang, J., Hanmer, R.S., Supakkul, S., Mirakhorli, M., 2013. The twin peaks of requirements and architecture. IEEE Softw. 30 (2), 24–29. http://dx.doi.org/10.1109/MS.2013.39.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J., 2011. Documenting Software Architectures: Views and Beyond, second ed. SEI Series in Software Engineering.

Davis, W.S., Yen, D.C., 2019. General systems design principles. In: The Information System Consultant's Handbook. CRC Press, pp. 577–584.

European Commission, 2020. Regulation of the european parliament and of the councillaying down harmonised rules on artificial intelligence (artificial intelligence act) and amending certain union legislative acts.

Fenn, S., Mendes, A., Budden, D., 2016. Addressing the non-functional requirements of computer vision systems: a case study. Mach. Vis. Appl. 27 (1), 77–86.

Fitzgerald, J., Larsen, P.G., Verhoef, M., 2014. Collaborative design for embedded systems. Academic Press 10, 3–14.

Giaimo, G., Anderson, R., Wargelin, L., Stopher, P., 2010. Will it Work? Transp. Res. Rec.: J. Transp. Res. Board 2176 (1), 26–34. http://dx.doi.org/10.3141/2176-03.

Glinz, M., 2007. On non-functional requirements. In: Proc. of 15th IEEE Int. RE Conf.. RE, New Delhi, India, pp. 21–26.

Habibullah, K.M., Horkoff, J., 2021. Non-functional requirements for machine learning: Understanding current use and challenges in industry. In: 2021 IEEE 29th International Requirements Engineering Conference. RE, IEEE, pp. 13–23.

Hancock, B., 2006. An introduction to qualitative research au t hors. Qual. Res. 4th, 504.

Hevner, A., Chatterjee, S., 2010. Design Science Research in Information Systems. Springer US, Boston, MA, pp. 9–22.

Hevner, A., March, S.T., Park, J., Ram, S., et al., 2004. Design science research in information systems. MIS Q. 28 (1), 75–105.

Heyn, H.-M., Knauss, E., Muhammad, A.P., Eriksson, O., Linder, J., Subbiah, P., Pradhan, S.K., Tungal, S., 2021. Requirement engineering challenges for ai-intense systems development. In: 2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI. WAIN, IEEE, pp. 89–96.

Heyn, H.-M., Subbiah, P., Linder, J., Knauss, E., Eriksson, O., 2022. Setting AI in context: A case study on defining the context and operational design domain for automated driving. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. Springer, pp. 199–215.

Horkoff, J., 2019. Non-functional requirements for machine learning: Challenges and new directions. In: 2019 IEEE 27th International Requirements Engineering Conference. RE, IEEE, pp. 386–391.

IEEE, 2019. IEEE Std 2413: Architectural Framework for the Internet of Things (IOT). IEEE Computer Society.

ISO, 2012. ISO/IEC/IEEE 42010:2012: Systems and Software Engineering — Architecture Description. Swedish Standards Institute, Stockholm, URL https://www.iso.org.

ISO, 2013. Intelligent Transport Systems — Forward Vehicle Collision Mitigation Systems — Operation, Performance, and Verification Requirements. International Organization for Standardization, Geneva, URL https://www.iso.org.

ISO, 2020. ISO/IEC TR 20547:2020: Information Technology — Big Data Reference Architecture. International Organization for Standardization, Geneva, URL https://www.iso.org.

Knauss, A., Damian, D., Franch, X., Rook, A., Múller, H.A., Thomo, A., 2016. Acon: A learning-based approach to deal with uncertainty in contextual requirements at runtime. Inf. Softw. Technol. 70, 85–99. http://dx.doi.org/10.1016/j.infsof.2015.10.001.

Knauss, E., Liebel, G., Horkoff, J., Wohlrab, R., Kasauli, R., Lange, F., Gildert, P., 2018. T-reqs: Tool support for managing requirements in large-scale agile system development. In: 2018 IEEE 26th International Requirements Engineering Conference. RE, IEEE, pp. 502–503.

Kondermann, D., 2013. Ground truth design principles: an overview. In: Proceedings of the International Workshop on Video and Image Ground Truth in Computer Vision Applications. pp. 1–4.

Kruchten, P., 1995. Architecture blueprints—the "4+1" view model of software architecture. IEEE Softw. 12 (November), 540–555. http://dx.doi.org/10.1145/216591.216611.

Kurup, U., Bignoli, P., Scally, J., Cassimatis, N., 2011. An architectural framework for complex cognition. Cogn. Syst. Res. 12 (3–4), 281–292.

Lauesen, S., 2002. Software Requirements. Pearson / Addison-Wesley.

Lesort, T., Caccia, M., Rish, I., 2021. Understanding continual learning settings with data distribution drift analysis. arXiv preprint arXiv:2104.01678.

Martínez-Fernández, S., Bogner, J., Franch, X., Oriol, M., Siebert, J., Trendowicz, A., Vollmer, A.M., Wagner, S., 2021. Software engineering for ai-based systems: A survey. arXiv preprint arXiv:2105.01984.

Meierhöfer, F., Weiss, R., Kucza, N., Brunnegard, O., vor dem Berge, M., 2021. Specification for selected pilots / use cases. Technical Report 957197, Horizon 2020 Research Framework, URL https://vedliot.eu/deliverable/deliverable-d23/.

Mendhurwar, S., Mishra, R., 2021. Integration of social and IoT technologies: architectural framework for digital transformation and cyber security challenges. Enterp. Inf. Syst. 15 (4), 565–584.

Mohd Aman, A.H., Yadegaridehkordi, E., Attarbashi, Z.S., Hassan, R., Park, Y.-J., 2020. A survey on trend and classification of internet of things reviews. IEEE Access 8, 111763–111782. http://dx.doi.org/10.1109/ACCESS.2020.3002932, URL https://ieeexplore.ieee.org/document/9119087/.

Moreb, M., Mohammed, T.A., Bayat, O., 2020. A novel software engineering approach toward using machine learning for improving the efficiency of health systems. IEEE Access 8, 23169–23178.

Muccini, H., Vaidhyanathan, K., 2021. Software architecture for ML-based systems: What exists and what Lies ahead. In: Proceedings of the 43rd International Conference on Software Engineering. URL http://arxiv.org/abs/2103.07950.

Murugesan, A., Rayadurgam, S., Heimdahl, M., 2019. Requirements reference models revisited: Accommodating hierarchy in system design. In: Proceedings of the IEEE International Conference on Requirements Engineering. 2019-September, IEEE, pp. 177–186. http://dx.doi.org/10.1109/RE.2019.00028.

Nalchigar, S., Yu, E., Keshavjee, K., 2021. Modeling machine learning requirements from three perspectives: a case report from the healthcare domain. Requir. Eng. 26 (2), 237–254.

NATO, 2020. NATO Architecture Framework. (September 2020), NATO Science and Technology Organization.

Nuseibeh, B., 2001. Weaving together requirements and architectures. Computer 34 (3), 115–119. http://dx.doi.org/10.1109/2.910904.

Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S., 2007. A design science research methodology for information systems research. J. Manage. Inf. Syst. 24 (3), 45–77.

Pelliccione, P., Knauss, E., Heldal, R., Magnus Ågren, S., Mallozzi, P., Alminger, A., Borgentun, D., 2017. Automotive architecture framework: The experience of volvo cars. J. Syst. Archit. 77, 83–100. http://dx.doi.org/10.1016/j.sysarc.2017.02.005.

Perrone, P., 2019. Notes on Category Theory with examples from basic mathematics. http://dx.doi.org/10.48550/arXiv.1912.10642, arXiv preprint arXiv:1912.10642.

Rao, S., Knauss, E., Mamun, M.A.A., Muhammad, A.P., 2021. Managing requirements-knowledge for developing cloud-based support of autonomous vehicles and transportation as a service: A design science research. Syst. Softw. In review.

Ray, P.P., 2018. A survey on Internet of Things architectures. J. King Saud Univ. - Comput. Inf. Sci. 30 (3), 291–319. http://dx.doi.org/10.1016/j.jksuci.2016.10.003.

Ries, B., Guelfi, N., Jahic, B., 2021. An MDE method for improving deep learning dataset requirements engineering using alloy and UML. In: Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development. SCITEPRESS, pp. 41–52.

Rozanski, N., Woods, E., 2012. Software Systems Architecture: Working with Stakeholders using Viewpoints and Perspectives. Addison-Wesley.

Russel, S., 2020. Human Compatible: AI and the Problem of Control. Penguin Books.

Schroeder, J., Holzner, D., Berger, C., Hoel, C.-J., Laine, L., Magnusson, A., 2015. Design and evaluation of a customizable multi-domain reference architecture on top of product lines of self-driving heavy vehicles-an industrial case study. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. Vol. 2, IEEE, pp. 189–198.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.F., Dennison, D., 2015. Hidden technical debt in machine learning systems. Adv. Neural Inf. Process. Syst. 2015-January, 2503–2511.

Serban, A.C., 2019. Designing safety critical software systems to manage inherent uncertainty. In: 2019 IEEE International Conference on Software Architecture Companion (ICSA-C). IEEE, pp. 246–249.

Tekinerdogan, B., Sözer, H., 2011. Defining architectural viewpoints for quality concerns. In: European Conference on Software Architecture. Springer, pp. 26–34.

Thilakarathne, N.N., Kagita, M.K., Lanka, D., Ahmad, H., et al., 2020. Smart grid: a survey of architectural elements, machine learning and deep learning applications and future directions. arXiv preprint arXiv:2010.08094.

Vogelsang, A., Borg, M., 2019. Requirements engineering for machine learning: Perspectives from data scientists. In: 2019 IEEE 27th International Requirements Engineering Conference Workshops. REW, IEEE, pp. 245–251.

Wan, Z., Xia, X., Lo, D., Murphy, G.C., 2020. How does machine learning change software development practices? IEEE Trans. Softw. Eng. http://dx.doi.org/10.1109/TSE.2019.2937083.

Washizaki, H., Uchida, H., Khomh, F., Guéhéneuc, Y.-G., 2019. Studying software engineering patterns for designing machine learning systems. In: 2019 10th International Workshop on Empirical Software Engineering in Practice. IWESEP, IEEE, pp. 49–495.

Weyrich, M., Ebert, C., 2016. Reference architectures for the internet of things. IEEE Softw. 33 (1), 112–116. http://dx.doi.org/10.1109/MS.2016.20.

Wieringa, R.J., 2009. Design science as nested problem solving. In: Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology. Association for Computing Machinery (ACM), pp. 1–12.

Woods, E., 2016. Software architecture in a changing world. IEEE Softw. 33 (6), 94–97. http://dx.doi.org/10.1109/MS.2016.149.

Woods, E., Rozanski, N., 2009. The system context architectural viewpoint. In: 2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture. IEEE, pp. 333–336.

Yokoyama, H., 2019. Machine learning system architectural pattern for improving operational stability. In: 2019 IEEE International Conference on Software Architecture Companion (ICSA-C). IEEE, pp. 267–274.

Zardini, G., Milojevic, D., Censi, A., Frazzoli, E., 2020. A Formal approach to the co-design of embodied intelligence. arXiv e-prints arXiv:2011.10756.

Further reading

Fong, B., Spivak, D.I., 2018. Seven sketches in compositionality: An invitation to applied category theory. arXiv preprint arXiv:1803.05316.

Mitra, T., 2015. Practical Software Architecture: Moving from System Context To Deployment. IBM Press.

Richards, M., Ford, N., 2020. Fundamentals of Software Architecture: An Engineering Approach. O'Reilly Media.