Transform safety-critical Cyber-Physical Systems into distributed solutions for end-users and partners

D38 (D2.3)

Techniques and Methodology to Analyse and Transformation towards DCPS

This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 101007260. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Netherlands, Finland, Germany, Poland, Austria, Spain, Belgium, Denmark, Norway.

Document Information

Project	TRANSACT
Grant Agreement No.	101007260
Work Package No.	WP2
Task No.	T2.3
Deliverable No.	D38
Deliverable No. in WP	D2.3
Deliverable Title	Techniques and Methodology to Analyse and Transformation towards DCPS
Nature	Report
Dissemination Level	Public
Document Version	1.0
Date	28/05/202430/05/2024
Contact	Axel Saalbach
Organization	PFLH
E-Mail	axel.saalbach@philips.com

Authors Table

Name	Company	E-Mail
Axel Saalbach	PFLH	axel.saalbach@philips.com
Thomas Amthor	PFLH	thomas.amthor@philips.com
Ken Sharman	ITI	ken@iti.es
Jesus Avila	NUN	jesus.avila@nunsys.com
Joan J. Valls	ITI	jvalls@iti.es
F. Javier Fernández-Bravo	ITI	fjfernandez@iti.es
Javier Coronel	ITI	jcoronel@iti.es
Sergio Sáez	ITI	ssaez@iti.es
Wolfram Ratzke	AVL	wolfram.ratzke@avl.com
Koen de Laat	PMS	koen.de.laat@philips.com
Astrid Rakow	DLR	astrid.rakow@dlr.de
Geoffrey Wielingen	PST	geoffrey.wielingen@ps-medtech.com
Abel Gómez Llana	UOC	agomezlla@uoc.edu
Javier Ferrer	KUM	javier@kumori.systems
Josep M. Bernabé Gisbert	KUM	jbgisbert@kumori.systems
Egbert Jaspers	VIN	egbert.jaspers@vinotion.nl

Reviewers Table

Version	Date	Reviewer
0.2	23/04/2024	Axel Saalbach (PFLH)
0.3	21/05/2024	Astrid Rakov (DLR)
0.4	24/05/2025	Robert Hofsink (PMS)

Change History

Version	Date	Reason for Change	Affected pages
0.1	22/02/2024	First draft of the table of content	All
0.2	18/04/2024	Draft version	All
0.3	17/05/2024	Internal revision	All
0.4	21/05/2024	Revised version	All
1.0	28/05/2024	Final version	All

Contents

LIST OF FIGURES

List of Tables

1 Glossary

Table 1: Terms and definitions

Term	Definition
Allocation (*aka* task allocation)	Task allocation refers to the decision of task placement and scheduling associated with the resource management.
Application migration	Application migration is the process of moving a software application from one computing environment to another. You might, for instance, migrate an application from one data centre to another, from an on-premises server to a cloud provider's environment, or from the public cloud to a private cloud environment.
Architecture	The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.
Architecture framework	Conventions, principles and practices for the description of architectures established within a specific domain of application and/or community of stakeholders.
Cloud service migration	[Cloud] service migration is a concept used in cloud computing implementation models that ensures that an individual or organization can easily shift between different cloud vendors without encountering implementation, integration, compatibility and interoperability issues.
Component	One of the parts that make up a system.
Computing platform	A computing platform is the environment in which a piece of software is executed. It may be the hardware or the operating system (OS), even a web browser and associated application programming interfaces, or other underlying software, as long as the program code is executed with it. Computing platforms have different abstraction levels, including a computer architecture, an OS, process containers, or runtime libraries. A computing platform is the stage on which computer programs can run.
Concept	An abstraction; a general idea inferred or derived from specific instances.
Cyber-Physical System	Digital system that semi-automatically interacts with its physical environment as integral part of its functionality. It integrates computation with physical processes, where system properties are determined by both cyber and physical parts.
Device	Physical entity embedded inside, or attached to, another physical entity in its vicinity, with capabilities to convey digital information from or to that physical entity.
Digital twin	A digital twin is a digital model of an actual physical system, which has a "live" connection (digital thread) with the physical system, so that it represents its actual status, and is used to derive a higher-level representation of the system's status and performance.

Edge Computing	Edge computing is a new architectural paradigm in which the resources of an edge server are placed at the edge of the Internet, in close proximity to cyber-physical systems, mobile devices, sensors and IoT endpoints.
Framework	A framework is an abstraction in which “engineering bricks” providing generic functionality can be selectively changed by system engineers, thus providing application-specific solutions. It provides a standard way to build and deploy CPS.
Industry 4.0	Industry 4.0 is a name given to the current trend of automation and data exchange in manufacturing technologies. It includes cyber-physical systems, the Internet of Things, cloud computing and cognitive computing. Industry 4.0 is commonly referred to as the fourth industrial revolution.
Mechanism	An established process by which something takes place or is brought about.
Method	A method consists of systematic steps for performing a task, in other words, it defines the “how” of each task.
Methodology	A collection of related formalisms, techniques, processes, methods, and tools. A methodology is essentially a “recipe” and can be thought of as the application of related processes, methods, and tools to a class of problems that all have something in common.
Middleware	Middleware is computer software that provides services to software applications beyond those available from the operating system. It can be described as “software glue”. Middleware makes it easier for software developers to implement communication and input/output, so they can focus on the specific purpose of their application.
Mission-critical system	A mission-critical system is a system that is essential to the survival of a business or organization. When a mission-critical system fails or is interrupted, business operations are significantly impacted.
Mixed-criticality system	A system containing computer hardware and software that can execute several applications of different criticality, such as safety-critical and non-safety-critical, or of different Safety Integrity Level (SIL). Different criticality applications are engineered to different levels of assurance, with high criticality applications being the costliest to design and verify.
Offloading (*aka* computation or task offloading)	Computation offloading is the transfer of resource intensive computational tasks to a separate processor, such as a hardware accelerator, or an external platform, such as a cluster, grid, or a cloud. Offloading computing to an external platform over a network can provide computing power and overcome hardware limitations of a device, such as limited computational power, storage, and energy.
Orchestration	Type of composition where one particular element is used by the composition to oversee and direct the other elements.

	Note: The element that directs an orchestration is not part of the orchestration.
Partitioning (*aka* application or code partitioning)	Divides the application code into several parts that will be executed on different platforms, I.e., mobile devices, cloudlets, or the cloud.
Platform	A collection of interoperable system "engineering bricks" that can be used to set up a system engineering environment in a company. A technology or engineering brick can be: a software tool/product, a software component to build a software tool/product, a system engineering methodology, an interface, a standard, or means for establishing interoperability that is needed for the efficient development of safety-critical embedded systems.
Process	A process is a logical sequence of tasks performed to achieve a particular objective. A process defines "what" is to be done, without specifying "how" each task is performed.
Reference Architecture	A Reference Architecture (RA) is an architectural design pattern that indicates how an abstract set of mechanisms and relationships realizes a predetermined set of requirements. It captures the essence of the architecture of a collection of systems. The main purpose of a Reference Architecture is to provide guidance for the development of architectures. One or more reference architectures may be derived from a common reference model, to address different purposes/usages to which the Reference Model may be targeted.
Reference Model	A reference model is an abstract framework for understanding significant relationships among the entities of some environment. It enables the development of specific reference or concrete architectures using consistent standards or specifications supporting that environment. A reference model consists of a minimal set of unifying concepts, axioms and relationships within a particular problem domain, and is independent of specific standards, technologies, implementations, or other concrete details. A reference model may be used as a basis for education and explaining standards to non- specialists.
Safety-critical CPS	A safety-critical CPS is a cyber-physical system where the failure or malfunction may result in one (or more) of the following outcomes: death or serious injury to people, loss or severe damage to equipment/property, environmental harm.
Safety-critical system	A system whose failure or malfunction may result in one (or more) of the following outcomes: death or serious injury to people, loss or severe damage to equipment/property, environmental harm.
Service	Services are the mechanism by which needs and capabilities are brought together.
Service migration	Dynamically moving (migrating) the service (or task) from one processing element to another at runtime.

Software component	An independent software unit that communicates with the surrounding system through explicitly specified interfaces.
Software framework	In computer programming, a software framework is an abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software. It provides a standard way to build and deploy applications and is a universal, reusable software environment that provides a particular functionality as part of a larger software platform to facilitate development of software applications, products and solutions. Software frameworks may include support programs, compilers, code libraries, tool sets, and application programming interfaces (APIs) that bring together all the different components to enable development of a project or system.
Solution	A means of solving a problem or dealing with a difficult situation.
System	A combination of interacting elements organized to achieve one or more stated purposes.
System component	A system architectural element.
Technique	Technical and managerial procedure that aids in the evaluation and improvement of the [system] development process.
Tool	A tool is an instrument that, when applied to a particular method, can enhance the efficiency of the task; provided it is applied properly and by somebody with proper skills and training.

Table 2: Abbreviations and definitions

Term	Definition
AI	Artificial Intelligence
B/L/O	Business, Legal, Operational aspects
BMS	Battery Management System
CI	Continuous Integration
CD	Continuous Delivery
CPA	Composite Performance Analysis
CPS	Cyber-Physical Systems
CT	Computed Tomography

DES	Discrete Event Simulation
DICOM	Digital Imaging and Communications in Medicine
DSL	Domain-Specific Language
IaaS	Infrastructure as a Service
IGT	Image-Guided Therapy
IoT	Internet of Things
MDE	Model-Driven Engineering
MRI	Magnetic Resonance Imaging
M&S	Modelling and Simulation
PaaS	Platform as a Service
PACS	Picture Archiving and Communications System
QoS	Quality of Service
SLA	Service Level Agreement
SLO	Service Level Objective
RA	Reference Architecture
RTAT	Report Turn-Around Time
SCDCPS	Safety-Critical Distributed Cyber-Physical System
T&V	Transition & Validation
WWTP	Wastewater Treatment Plant

2 Introduction

2.1 Role of this deliverable

This deliverable relates to Task 2.3 - "Trade-off analysis and transformation of classic solutions into distributed safety-critical CPS solutions" of Work Package 2 (WP2) - "Distributed solution architectures and platforms for safety critical CPS". The main objective of this Work Package is the design and creation of the TRANSACT reference architecture. The structure of the work revolves around three main tasks: developing the reference architecture (Task 2.1), creating methodologies for designing and evaluating distributed Cyber-Physical Systems (CPS) (Task 2.2), and transforming traditional CPS into distributed systems while considering associated trade-offs and general development processes (Task 2.3).

The activities in Task 2.3 are focussing on three key aspects:

- **Trade-off analyses for the distribution of functionality** to find sweet spots in the decision which functions are best performed in the cloud, edge, or on-device tier of the TRANSACT architecture.
- **Strategies supporting the transition**, with a focus on general software design principles supporting a transition towards a distributed solution while at the same time enabling the capability to guarantee certain system functionality.
- **Generic development processes** reaching from top-level system design down to hardware/software architecture design.

In this deliverable, resulting from Task 2.3, methodologies and techniques facilitating the shift from classical CPS to distributed CPS solutions are presented and discussed. This encompasses techniques for analysing and optimizing function allocation, as well as for design comparison. Drawing on the work of the TRANSACT partners in the individual use-cases, the contributions cover a wide range of topics from Model-Driven Engineering of CPS, to optimized function distribution using Discrete Event Simulation, to service models for the management of cloud applications. Consequently, the contributions address different aspects of the overall task objectives. As visualized in Table 3, the objectives are evenly covered by the work areas addressed by the partners.

Table 3: Overview of how the techniques and methodologies discussed in each partner's contribution are linked to the objectives of Task 2.3

	Trade-off analysis	Strategies	Development process
UOC	X		X
VIN	X	X	
AVL			X
PMS		X	X
PST	X	X	
PFLH	X		
ITI	X		X
KUM		X	X

2.2 Relationship to other TRANSACT deliverables

This document relates to the following TRANSACT deliverables:

- The TRANSACT reference architecture for SCDCPS is presented in D7 (D2.1) and D26 (D2.4). The reference architecture represents the basis for the investigations of transition techniques and methodologies presented in this report.
- The simulation and validation approaches presented in D27 (D2.2) are closely linked to the trade-off analyses described here.
- D6 (D1.2) reports about the technical requirements per use case and common methodology to support the transformation.
- The transition guide to facilitate safety-critical distributed CPS solutions (D23 (D1.3) and D36 (D1.5)) describes the transition methodology framework with respect to TRANSACT focus areas and cross-cutting aspects. The transition methods in many of these aspects are supported by analysis and simulation approaches detailed in this document.
- As system performance is a key element of the trade-off analysis described in this deliverable, the performance modelling and prediction technologies detailed in D33 (D3.5) can also be employed for the purpose of trade-off simulation and sweet-spot identification.

2.3 Modelling and Analysing DCPS

In order to transform CPS into distributed systems and evaluate the trade-offs in view of individual use-cases, the TRANSACT partners have utilized existing approaches and developed new concepts. These contributions, as detailed in this report, cover a wide range of tools and methodologies, spanning from systems engineering and model-driven engineering to the exploration of simulation and optimization algorithms. The development of such modelling and analysis methods for distributed cyber-physical systems is a rapidly evolving field. This section provides an overview of current literature and elucidates the context in which the TRANSACT contributions of this report are embedded.

The intrinsic complexity of distributed cyber-physical systems poses challenges for all areas of system design, optimization, and implementation (Mittal and Tolk 2020). To tackle these challenges, it is essential to have a good understanding of suitable modelling and simulation (M&S) techniques, how they can support in the transformation from system engineering to validation and trade-off analysis, and where there are potential pitfalls to be aware of (Falcone and Garro 2020).

Safety-critical aspects of the system would preferably be modelled using formal methods, mathematically rigorous and structured descriptions that provide mathematical proof for a specific system behaviour (Liu, Woodcock and Zhu 2013). Formal modelling can be very complex but offers the highest level of confidence and accuracy. A comprehensive literature review of formal methods can be found in (Masmoudi, et al. 2022). Even in the early design phase of requirements engineering, formal methods and languages can support the development process (Zahid, et al. 2022). Independent of the exact type of modelling, Model-Driven Engineering (MDE) approaches have become essential tools for the development of reliable CPS and are still a growing area of research (Mohamed, Challenger and Kardas 2020).

The behaviour of a system can be described and formalized using different modelling languages. An overview of modelling languages and frameworks applicable to CPS is provided in (Graja, et al. 2020) and (Zahid, et al. 2022). Typically, structured languages such as the Unified Modelling Language (UML) (OMG, Unified Modeling Language n.d.) serve as a basis for formalizing processes and interactions. For the purpose of

systems engineering, SysML has been developed as an extension of UML (OMG, What is SysML? n.d.). In order to account for specific requirements of distributed CPS, attempts have been made to adapt the UML formalism for this specific purpose (L. Zhang 2018). In other specific domains, variations of the business process modelling and notation (BPMN) framework have been found to be applicable, as in the case of hospital processes related to TRANSACT use case 4 (Pufahl, et al. 2022).

Once a structured model of the system is available, simulation methods help to understand the behaviour of the system under different conditions – a prerequisite for validation and trade-off analysis. Due to the complex nature of DCPS, several different simulation methods may be required. Some very prominent methods are explained in the following.

Aspects that require capturing the continuous physical dynamics of a (sub-)system can be modelled using continuous analytical methods, typically employing ordinary differential equations (ODEs). This may be the method of choice for describing sensors or actuators interacting with the physical world or any other system component that is exposed to dynamic physical processes.

Other aspects of the systems exhibit discrete states and state transitions, as is mostly the case for digital (cyber) part of the system or for any step-by-step process representations of the system behaviour. These aspects are often modelled using a Discrete-Event Simulation (DES) method (Fishman 2001). In discrete-event simulations, the behaviour of the system is determined by a series of events, occurrences that influence the state of the system at specific points in time. DES is particularly useful for modelling queueing and networking aspects. When a system is better described as a collection of components that interact with their environment in a specific manner, an Agent-Based Simulation (ABS) may be the method of choice. This method involves modelling and simulating the behaviour and interaction of autonomous parts (“agents”) within a system. ABS has been found particularly valuable for modelling distributed cyber-physical systems that are difficult to describe by simple linear process models. Although ABS had been hypothesized to replace DES in operations analysis already years ago (Sieber, et al. 2010), there are still many situations where DES is very well applicable, such as the process-driven medical image reconstruction analysis exemplified in Section 5.7. An overview of state-of-the-art simulation software, in particular for DES and ABS, can be found in (Paape, Van Eekelen and Reniers 2024). Additional simulation frameworks tailored to the needs of CPS design are constantly evolving (Tampouratzis, Mousouliotis and Papaefstathiou 2023).

Finally, since many systems comprise very diverse components, processes, and interactions, it is often necessary to combine simulation methods to a hybrid approach (Tolk, et al. 2018). Hybrid models can incorporate combinations of continuous and discrete methods. It is worth noting that such hybrid approaches, when performed in a rigorous and accurate manner, still pose many challenges and questions about how to create consistent models remain (Bliudze, et al. 2019).

When hypotheses about the system behaviour need to be tested, but numerical algorithms are infeasible, a statistical model checking (SMC) approach can be used, which is also based on the simulation models. In SMC, the violation or satisfaction of assumptions is inferred from statistical properties of a finite set of samples (Legay, Delahaye and Bensalem 2010). A review of statistical model checking approaches in the context of CPS can be found in (Pappagallo, Massini and Tronci 2020).

In the design process of a DCPS architecture and setup, it is often required to perform a trade-off analysis to balance different behavioural aspects of the system. The modelling and simulation techniques described above can be used to evaluate different implementations of the DCPS and perform a trade-off analysis by varying the system parameters, as elaborated in different parts of this document. When the parameter space is large, optimizing the implementation details with respect to a certain performance or behavioural balance requires calculating a large number of configurations. This is where a genetic algorithm can help to find an optimum solution for given boundary conditions and optimization targets (Alhijawi and Awajan 2023). An application of such an algorithm is described in Section 5.8.

2.4 Structure of this deliverable

The remainder of this document is organized as follows.

In Chapter 3, an overview of the distributed solution architecture is given, which has been developed as part of the TRANSACT project. It constitutes a universally applicable concept for transforming safety-critical CPS into distributed CPS solutions. TRANSACT aims to enable the hosting of critical applications and facilitate the allocation of their functions over the device-edge-cloud computing continuum. The infrastructure also seeks to enhance system capabilities by offloading non-critical functions from the device and deploying safety-critical functions to the edge tier, thereby improving reliability, performance, and innovation speed. The architecture also introduces core and value-added services to ensure safety, performance, security, and privacy.

In Chapter 4, T&V^2 is introduced as a transition and validation methodology, a holistic approach for the transition towards distributed solutions in the edge-cloud continuum. It considers technical, operational, legal, and business aspects, and extends beyond development to deployment and maintenance. The methodology involves stepwise refinement of requirements and exploration of the design space, aiming for early feasibility determination. It aims to explore the design space for a distributed solution and guide verification and validation towards meeting diverse requirements in a complex system.

Chapter 5 details the contribution of the partners, covering the different topics of Task T2.3:

- In Section 0, Model-Driven Engineering (MDE) is presented as a technique for simulation-based design and analysis of distributed CPS. Within MDE, CPS Architecture Models define the infrastructure, application, and deployment aspects of a system. A Code Generator converts the abstract model into simulation deployment and configuration files, including a monitoring component. These files are then used as input for the System Simulation and Monitoring stage, during which the system is executed and monitored within a cloud test environment. This approach facilitates the evaluation of various cloud configurations and architectures, serving as a basis for trade-off analysis.
- Section 5.2 addresses the technical challenges of managing cloud applications in the context of CPS. It emphasizes the need for advanced tools and automation to facilitate the deployment, installation, configuration, and monitoring of these applications. The discussion underscores the importance of adopting sophisticated tools and a language that supports separation of concerns, readability, and technology agnostic descriptions. The Kumori Service Model (KSM) is introduced as a language that embodies these principles, providing an abstract and declarative framework to describe and execute cloud application services.
- Section 5.3 investigates the transformation of classic (local) traffic control systems towards distributed CPS and examines the potential implications and opportunities for advancements in traffic management. It offers an overview of the current state in traffic control and addresses important implementation aspects. It provides a trade-off analysis, focusing on safety, latency, and cost-related considerations. Finally, it describes how external factors such as the adoption strategy of road authorities practically affects the transition towards CPS.
- Section 5.4 discusses the transition process towards distributed CPS within the context of three different domains: Requirements, Behavioural, and Physical. It emphasizes a design and development process involving the physical decomposition of a system, followed by an assessment of its elements according to quality attributes. This is combined with a functional decomposition and an evaluation of the functional and physical domains with regard to these criteria. The process is exemplified in the context of Cloud-Featured Battery Management Systems, specifically in the

computation of battery cell resistance, effectively illustrating a mapping to the functional and physical domains.

- Section 5.5 focuses on the concept of "build once, run anywhere" for medical software, addressing the challenges in the highly regulated medical domain when it comes to the transition towards distributed CPS. The use of a microservice architecture with well-defined interfaces is discussed to manage the dependencies of complex applications and to facilitate the development of cloud-ready applications.
- In Section 5.6, the transition towards distributed CPS using an edge/cloud platform is examined in the context of medical image data for various healthcare applications, with Vesalius3D as a case study. It provides an analysis of the trade-offs between edge and cloud deployment, considering factors such as smooth interaction, scalability, application access, and costs. The discussion further emphasizes the shift towards cloud deployment to target new market segments and improve adaptability, scalability, and usability in support of enhanced treatment planning and patient care. Additionally, it highlights the importance of aligning technical aspects with business objectives for a successful transition.
- In Section 5.7, distributed CPS are investigated in the context of a safety-critical clinical environment. Discrete Event Simulation (DES) is introduced as a tool to guide the distribution of services in an Edge-Cloud-based clinical application platform, with the goal of enhancing resource utilization, scalability, and accessibility. Complex clinical workflows and processes are analysed using an individual event-based approach. An example based on clinical data is used to demonstrate the application of DES for evaluating the distribution of computational tasks related to medical image reconstruction across various tiers of infrastructure.
- Section 5.8 introduces the concept of Design Space Exploration with Genetic Algorithms for transition towards distributed CPS. In order to facilitate the design and analysis of distributed CPS, with conflicting design objectives and constraints, a multi-objective, non-linear constrained optimization tool, based on genetic algorithms, is presented. Based on Application, Platform and Deployment models, the design space optimization is able to provide multiple potential solutions in terms of a Pareto Set, providing a mapping of software functions to the components of the CPS platform.

3 Use Cases and the Reference Architecture

3.1 Reference Architecture (ITI & NUN)

The TRANSACT project has developed a universally applicable distributed solution architecture concept, framework and transition methodology for the transformation of safety-critical CPS into distributed safety-critical CPS solutions.

TRANSACT's reference architecture is based on a three-tier computing continuum that spans from the CPS device (first tier), through the edge (second tier), to the cloud (third tier). This architecture, displayed in Figure 1, brings together CPS end-devices at the edge of the network, with edge computing servers and cloud computing facilities, hosting multiple and heterogeneous mixed-criticality applications.

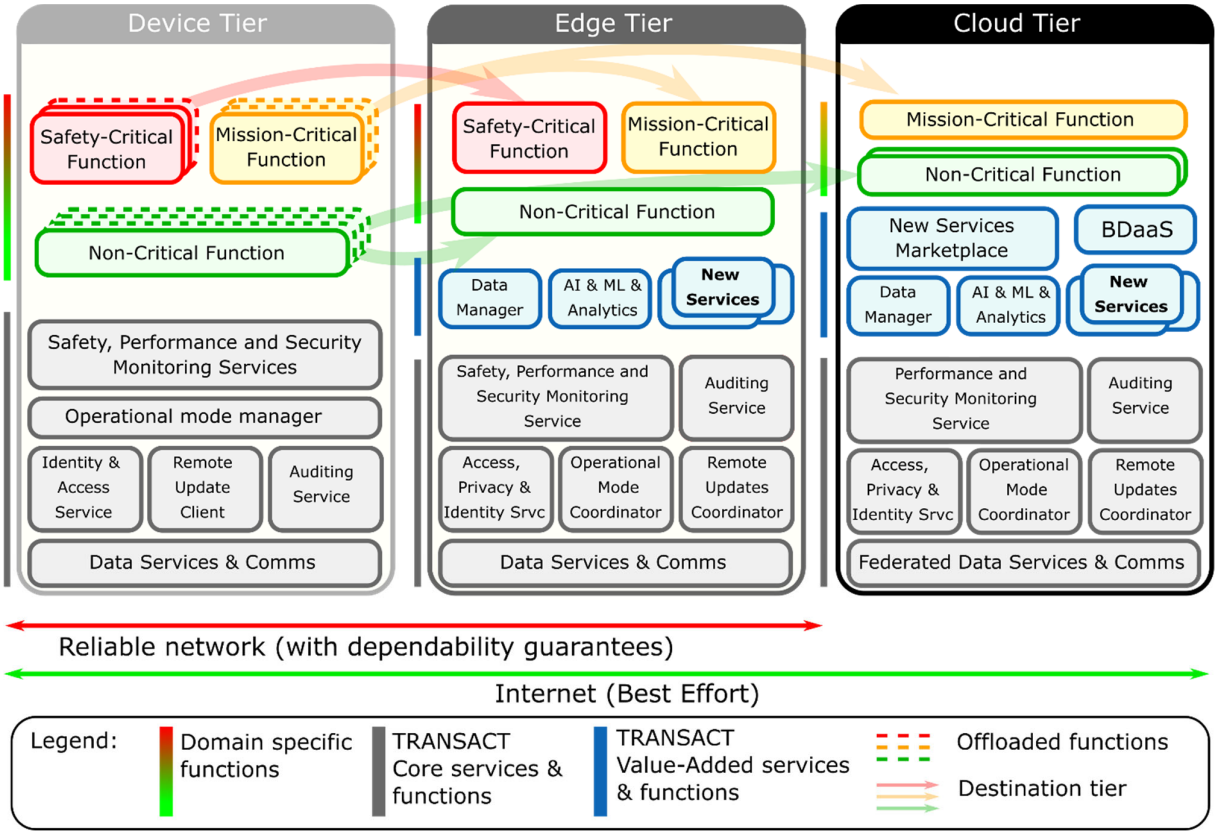

Figure 1: TRANSACT Reference Architecture

3.1.1 Three-tier computing continuum

A key element of TRANSACT is that both these off-device tiers (edge and cloud), in addition to being multi-application, multi-device and possibly multi-tenant, support scalable and interoperable, as well as safe and secure, solutions. Complex applications can be flexibly deployed over this distributed architecture. Novel services and applications can share edge and cloud infrastructures, reuse domain services, and in general be faster developed, tested, and released independently from the safety-critical device itself. This distributed

architecture concept is directed to enable new business models by transforming system manufacturers into solution providers.

The main system functionalities are distributed across the three tiers. The CPS end-devices are linked to the cloud computing facilities via the edge infrastructure, while the end-devices and the edge tier are connected via a reliable network, whereas the cloud tier is only accessible from the Internet. Each tier provides a specific quality of service level especially with respect to performance aspects (such as response times and data transfer guarantees) that are essential for the safety- and mission-critical functions. These, along with non-critical functions, may be offloaded from the device to other tiers.

3.1.2 TRANSACT services and functions

The safety- and mission-critical functions are key in the safety-critical CPS. TRANSACT aims to improve the existing CPSs by, firstly, stripping the device of the functions that are not safety- or mission-critical and can be executed remotely; and secondly, by offloading certain safety-critical functions to the edge tier. The functions to be offloaded are identified at design time and deployed in the required tiers. As a result, the device would only keep the basic safety-critical functions while offloading the remaining functions to the other tiers. Such an approach presents numerous advantages such as: improved reliability and performance of the device (as fewer services are running on it), improved efficiency of the offloaded functions due to usage of better hardware at the edge or cloud, and improved innovation speed of the distributed CPS solution as the new or upgraded functions can be deployed with greater ease at the edge and cloud.

The architecture introduces several Core services deployed across all tiers to ensure safety, performance, security and privacy of the new solution, especially when offloading functions from the device. They are available to every use case. Value-added services and functions, that enhance the system capabilities, are also defined. These functions can be introduced during the system design or after the system release (as part of the system updates). They refer to potential components that may be included depending on the use case.

In the TRANSACT project, this universally applicable distributed solution architecture has been elaborated and augmented with a framework and transition methodology for transforming safety-critical CPS into distributed safety-critical CPS solutions. The architectural solution can host a diverse set of mixed-criticality applications, which can be distributed over multiple resources, e.g., CPS devices, high-end computing resources hosting multiple applications of mixed-criticalities, or in the cloud. It also supports a variety of interesting trade-offs by allowing the allocation of mixed-criticality functions over the computing continuum of device-edge-cloud.

TRANSACT reference architecture is thoroughly described in deliverable documents D7 (D2.1) and D26 (D2.4).

3.2 Transition to the TRANSACT concept (ITI & NUN)

Business, product, and technical system requirements are mapped and assimilated into the reference architecture to ensure its effectiveness. However, the complete architectural vision covers not only the technologies matching the different components, but also the operation environment for the development, deployment and execution of the solution, including the edge and cloud underlying infrastructure. Therefore, a blueprint assessing an approach to the reference architecture is required, focused on how to realize the transition from a monolithic safety-critical CPS to a distributed solution that keeps ensuring safety, performance, security and privacy, as well as regulatory requirements compliance.

As deliverable D23 (D1.3) establishes, to make a successful transition and transformation from the on-device solution to the distributed solution using edge and cloud services, careful consideration and planning are

required. First, it is needed to thoroughly review and understand the current system architecture and the technology stack it is built on. Second, the cloud providers' capabilities and services need to be explored, to optimally match application workloads to the cloud provider environment and services, finding the best combination of technologies and services that is cost-effective. Third, the high-level blueprint of the new solution should be created covering all the product design stages, supporting development, deployment and operation of the solution, in addition to the required underlying infrastructure, as Figure 2 depicts.

These elements are analysed along the next paragraphs, aiming to point out considerations for the adoption of TRANSACT's approach and reference architecture, that should be kept in mind when tackling the transformation of a safety-critical CPS into a distributed solution. A more detailed study and description of these elements, that guide such a transition process, can be found in deliverable documents D23 (D1.3) and D36 (D1.5).

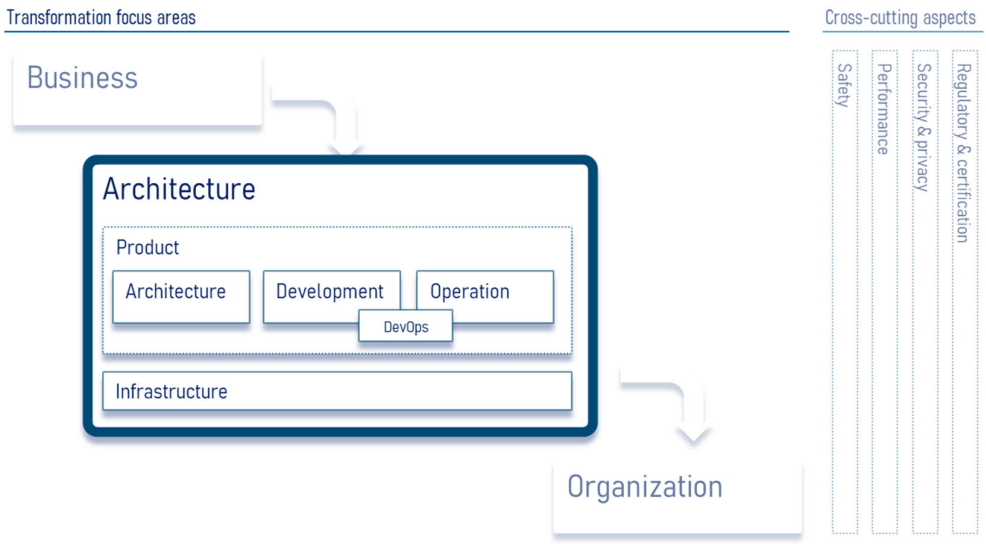

Figure 2: TRANSACT transition: Architecture Transformation Area elements

Taking into consideration the outline, analysis and itemization of the reference architecture presented in Section 3.1, the following aspects impacting the **architecture** should be considered while transforming an on-device CPS into a distributed solution:

- Architectural approaches for distributed safety-critical solutions
- Migration of safety-, mission, and non-critical functions to the new architecture
- System security
- System updates
- System monitoring and observability

Migrating safety-critical CPS to a distributed solution over the device-edge-cloud computing continuum also impacts **development**. The following aspects require particular attention when developing a product:

- CI/CD pipeline
- Testing approach
- Source code version management
- Knowledge and expertise on device-edge-cloud computing development tools and methodologies

Product **operation** is impacted by the transformation to a distributed solution since it is concerned with the deployment and monitoring of the delivered solution in the device-edge-cloud continuum. Therefore, the (currently device-focused) operations team needs to expand their capabilities to also cover edge and cloud concepts and technologies in order to ensure the most efficient deployment and support of the new edge/cloud solution. In general, the new system architecture requires the product operation team to focus on the following aspects:

- Deployment
- Maintenance and updates
- Runtime monitoring & logging
- Runtime optimization

Traditional solution creation approaches have a strict separation of roles like development, operation, quality engineering, and security, that may lead to inefficiencies and, consequently, delays of new product releases. **DevOps** is a set of methodologies, practices and technologies aiming to improve the speed and efficiency of the software development and product release processes, emphasizing the collaboration between the development and operations teams. In the new distributed CPS solution, DevOps will play a significant role in the product lifecycle, especially due to the complexity of the new deployment over the three tiers, more intricate product design and the new concerns in the areas of safety, performance, privacy and security. These non-functional properties must be made relevant during the CPS regular development and operational activities, aiming for a more effective and efficient realization of cross-functional requirements. Furthermore, considering the teams are well-formed, sized and proper team DevOps skills to confront the new ways of doing will be highly important for the migration success.

Finally, to support the new device-edge-cloud continuum solution architecture, the **infrastructure** for product development and deployment needs to be adapted as well. In general, the new solution architecture will not only depend on the infrastructure and software services that are owned by the organization, but also on the infrastructure of the cloud provider or the specific services and solutions of a certain cloud platform. The product development infrastructure, used by the development team, covers all the development environments and tooling used to build and test the solution's assets. The product deployment infrastructure, used by the operations team, covers the CD environment and tooling used to deploy and monitor the running solution over the device-edge-cloud continuum, including safety-critical and mission-critical functions. When an organization considers adopting the aforementioned DevOps model, then the development and deployment infrastructures should be tightly integrated for efficient and optimized product deployment, updates and operation.

4 Transition and Validation Methodology (T&V² Methodology)

In this section, we outline a general strategy for the development and transition of a monolithic device to a solution that is distributed over the edge cloud continuum. It draws heavily on the T&V methodology of T2.2, which guides the development of a distributed solution through accompanying verification and validation activities. The methodology presented below adapts the T&V methodology presented in D27 with a focus on a holistic transition methodology.

The T&V² Methodology[1] places a stronger focus on considering that the design decisions induce business constraints, legal obligations and operational changes (B/L/O). In addition, it considers that not only the development of the technical solution itself is necessary, but the solution must be deployed and maintained. The methodology hence specifies the requirements on the migration activities that are implied by the respective design decisions. Note that deployment and maintenance are both complex processes that are monitored and have to deal with faults. Data from these processes can be propagated back to the development phase to drive the specification of requirements. Figure 3 illustrates the development process of the transition process, where the traditional development activities of the technical solution are a major activity but this activity is accompanied by the concretization of requirements relating to the additional dimensions mentioned. Figure 3 also illustrates that information of the deployment and runtime phase is fed back triggering the further adaptations and the continuous evolution of the system.

The methodology described in this section constitutes a generic framework generally applicable to transition and validation processes. This high-level description of the relevant concepts is complemented by detailed examples of real-world implementation aspects in Section 5.

[1] T&V² abbreviates Transition & Verification, Validation. The superscript 2 serves to distinguish it from the T&V methodology of T2.2.

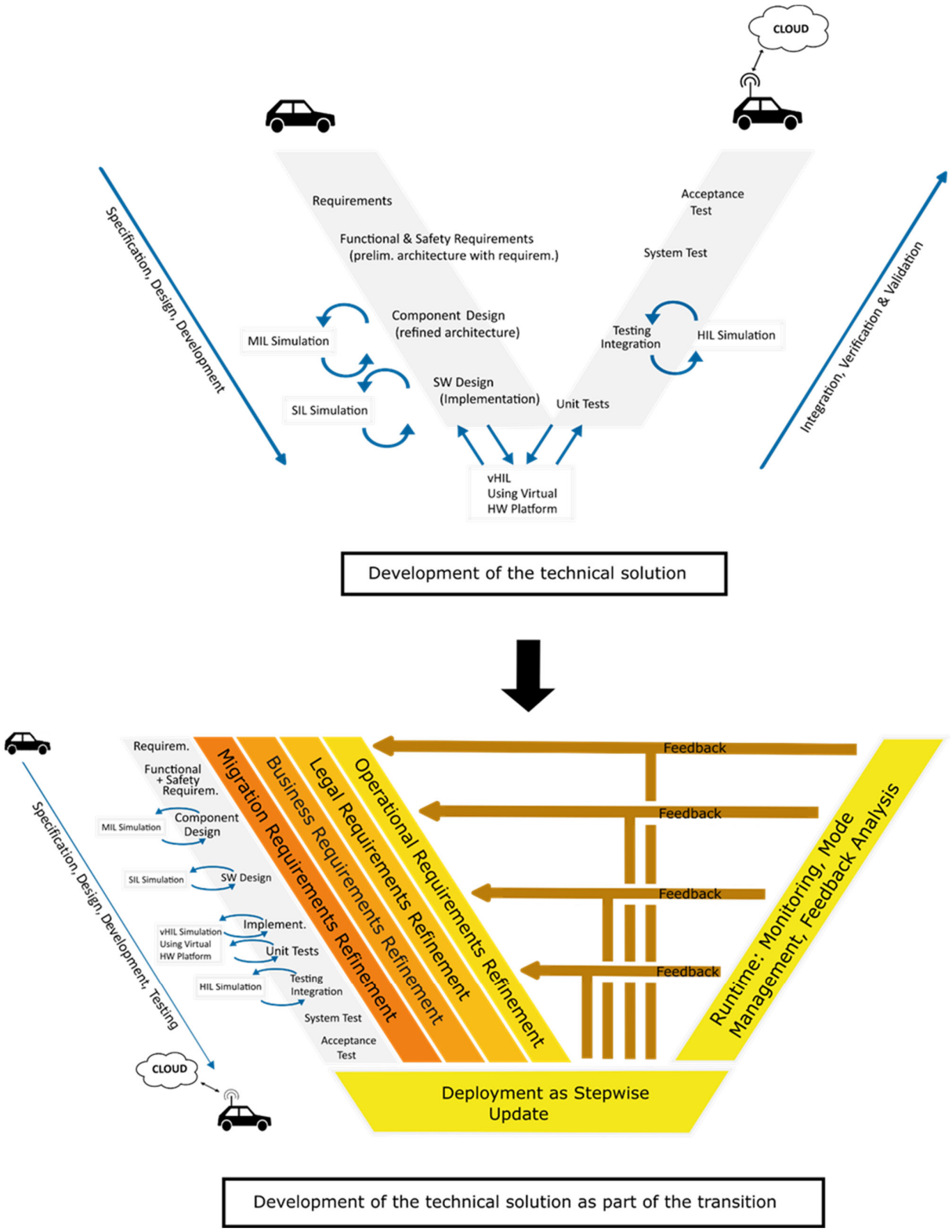

Figure 3: Changes towards a development of a technical solution

4.1 Motivation

In the following, we describe in a generic way a stepwise process developing a distributed solution along with constraints on a migration plan. The methodology starts with a given system that shall be enhanced by a distributed functionality in the edge-cloud continuum.

The methodology starts with what is expected from the overall distributed solution and ends with having defined the technical requirements for all components, as well as requirements for

- migration,
- running in normal & fail-safe modes and
- maintaining & evolving

the system. Also, requirements that result from the

- operational
- legal
- business

perspectives are specified. It defines increasingly more concrete requirements reflecting design decisions that have been taken (e.g. “What functionality is located where?”, “What services are used in the cloud?”, etc.). When it turns out that a requirement cannot be realized, the design variant will be dismissed and the alternatives will be explored. The method proceeds until the design of a technical solution has been concretized to directly implement it component-wise.

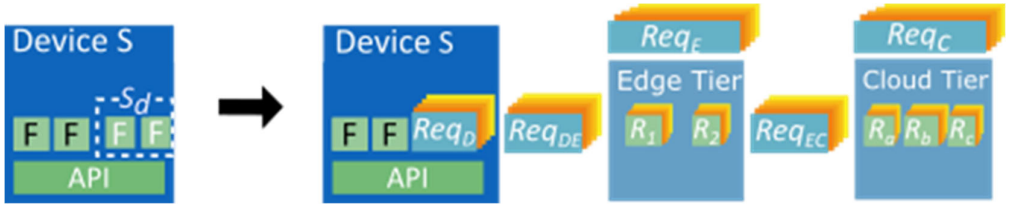

Figure 4: Requirements definition by the T&V[2] methodology leads to a distributed solution *S*. The requirements relate to technical aspects but also to business, legal and operational aspects as well as to the migration plan.

4.2 Relation to the TRANSACT transition methodology

The TRANSACT transition methodology (cf. D23 (D1.3) and D36 (D1.5)) considers the transition process holistically starting from the business perspective. In contrast, in T2.2 (and in D27(D2.2)) the technical system development process is considered to explain where in the development process the methods & tools for validation and assessment of a system (model/variant) are required. In this deliverable, both methods are combined in order to derive strategy for development for a distributed system and requirements on the migration plan.

4.3 Steps of the T&V Methodology

The T&V[2] methodology considers the case that an improved version *S'* of a given SCCPS *S* is to be developed. The focus is on the case where *S'* is derived from *S* by distributing a set of its functions S_d over the edge-cloud continuum. More precisely, we assume that S_d gets replaced by a set of functions S'_d that use the services of the TRANSACT architecture and together with them provides the improved functionality. The T&V[2]

methodology leaves open how the exact transition strategy is. For instance, S_d may be rehosted or S_d may get refactored and rehosted.

The basic argument for "*S'* is an improved version of *S* that satisfies the requirements *Req'"* is that it has been established that the distributed S'_d "adequately replaces" the initial S_d. The T&V methodology assumes that the requirements will be defined such that S'_d will "adequately replace" S_d, i.e. the concretized requirements refine the initial requirements. The T&V^2 methodology drives the definition of requirements and aims to establish whether the requirements can be realized as early as possible.

Figure 5 (cf. D27, Sect. 3.3) gives an overview of the T&V^2 methodology. In the following, we explain the methodology in more detail.

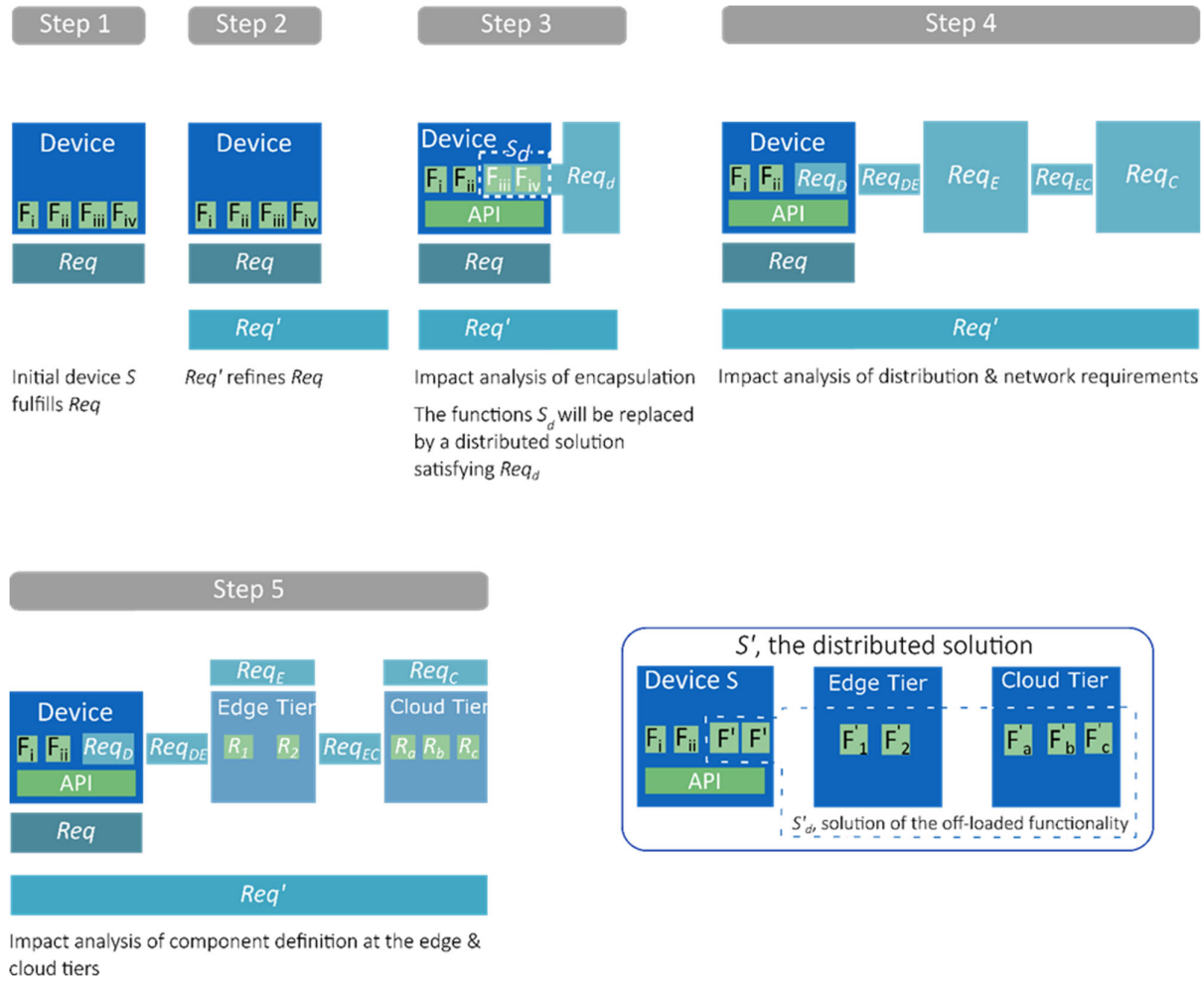

Figure 5: T&V Methodology

The methodology stepwise concretizes the distributed solution by concretizing requirements by decreasing the level of abstraction. Since we aim for a fast development process, we prefer to evaluate as early as possible whether a design variant is realizable or not.

Whether a variant is realizable depends on whether all requirements of the different dimensions can be satisfied. We focus here on the dimensions *operational, legal, business* and *technical*. Operational

requirements refer to how to run the system. What data has to be logged, what requirements are on the startup/shutdown controls or back up, etc. Legal requirements refer to constraints that have to be obliged by law and rules that result from it for subsystems. This relates to laws that concern the solution but also to obligations that the solution has to guarantee due to sales contracts. We moreover consider business policy rules here. There are certainly other dimensions that constitute an important focus, like ethics or sustainability, but TRANSACT did not focus on these.

As a running example, we consider the operational requirements "availability of staff", legal "data protection demands", business "maximum cost may not be exceeded".

At each concretization step, the engineer hence has to ask:

- What does the technical design decision to refine *Req* to *Req'* mean for the operational, legal, business requirements? Are they still satisfied?
- What (refined) requirements follow from this decision regarding migration, running in different modes and maintenance & system improvement?
- Are these requirements satisfiable?

In the following, we use decorated versions of *S* (*S'*, S'_d,…) and *Req* (*Req'*, Req'_d,…). We usually refer to implemented solutions of requirements (i.e. system or functions) as *S* (plus some decoration) and we refer to the specification of requirements by some decorated form of *Req*. In order to stress that the implemented solution has to satisfy the respective requirement, we try to use matching decorations, i.e. *Req'* is implemented by *S'* (and which is synonymous for "*S'* satisfies *Req*'" in the following).

4.4 Stepwise Refinement of Requirements & Exploration of the Design Space

For simplicity, we assume that *S* only consists of an end device and satisfies the requirements *Req*. We also assume that the system *S'* -to be developed- is developed in order to provide additional or improved functionality.

In **Step 1**, the process starts with a CPS end device *S* on which a collection of functions is implemented. The device is known to satisfy the requirements technical *Req* and the business requirements Req_B, legal requirements Req_L and the operational requirements Req_O, since they have been established during the design of *S*.

In **Step 2**, the new technical requirements *Req'* are specified as well as the relevant business, legal and operational requirements Req_B, Req_L, Req_O. As we are interested in service improvements, the requirements *Req'* refine the requirements *Req* of the initial device *S*. The legal requirements Req_L are all the laws and regulations that the to-be-designed solution has to satisfy. This relates to data protection laws but also company specific regulations where data may be processed and stored. But also employee protection laws might become relevant, if distributing a solution necessitates e.g. that technical staff has to be ready to fix off-site devices. This business requirements will often focus on the gained value. The requirements Req_B, Req_L, Req_O describe the business/legal/operational requirements that are relevant for the to be developed solution S' that satisfies the new requirements *Req'*.

In **Step 3**, the question "How can *S* be modified in order to satisfy the new requirements *Req'*?" is considered. The different design alternatives will be evaluated in terms of cost-benefit considerations, as well as the risks of changing established components. If the new requirements *Req'* cannot be fulfilled locally, a plausibility check will evaluate whether it is feasible to distribute certain services over the edge-cloud-continuum. Figure

6 gives an overview of Step 3 for the case that S_d gets distributed. In the following, we explain step 3 for offloading S_d in more detail.

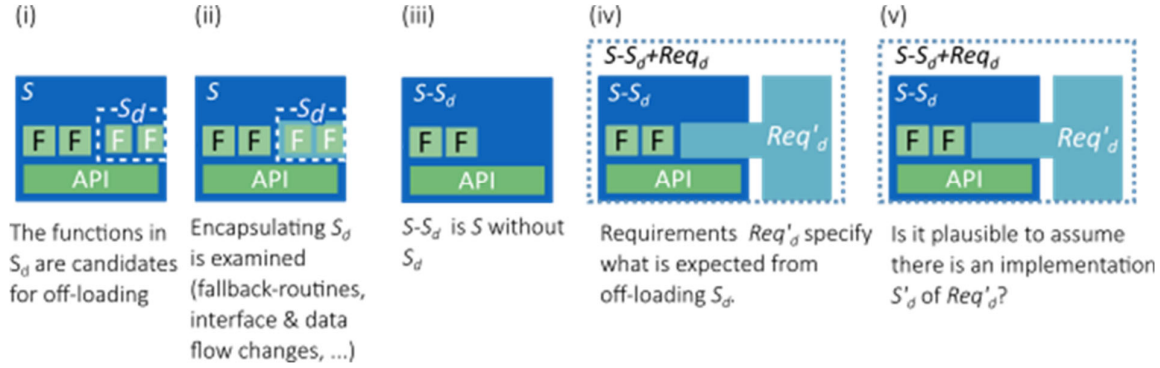

Figure 6: Off-loading the set of functions S_d

Figure 6 illustrates that a set of functions S_d is identified that are good candidates for offloading. S_d is used as a starting point to specify the requirements on the offloaded functionality, Req'_d. Req'_d (cf. Figure 6(iv)) specifies what the future off-device solution S'_d (cf. Figure 6, right most and Figure 5, step 3) must provide for the end-device, so that S', the final system, satisfies the strengthened requirements *Req'*.

Req'_d already implies an impact on the data flow. To implement the new improved functionality the required data will leave the device. The decision whether the functionality will be processed at the edge or the cloud tier has not been taken yet. At this step additional B/L/O requirements must be added that ensure that the distributed solution will be acceptable. The guiding questions here are, "What is the impact on B/L/O when the initial functionality will be replaced by Req'_d that is provided on the edge-cloud continuum?".

The device also has to be redesigned. Since offloading entails additional communication and synchronization for the end device to connect to the edge-cloud continuum, (R-a) the services S_d will have to be detached from their old infrastructure and (R-b) a preliminary interface between the end device with the TRANSACT device tier components and the to-be-developed solution S'_d is specified. As data will be send to the edge-cloud continuum data-preprocessing at device tier may be necessary for e.g. anonymization. (R-c) The new design also must provide fail-safe mechanisms to cope with the changed response times or outages due to network problems for the distributed functionality Req'_d.

Since at this point, the exact solution is not clear, a plausibility check must assert that appropriate fail-safe mechanisms can be implemented on the device. During the plausibility check the requirements Req'_d on the to be developed solution can be concretized. Regarding the design of the fail-safe mechanisms a guiding question is "What does the edge-cloud solution S'_d need to guarantee, so that the device tier is able to function correctly even in the worst case?" and "What kind of proof/certification do the involved edge-cloud components need to provide in order to be useable in a safety-critical system?".

The plausibility check should also include a preliminary migration plan that specifies requirements on how and when the device can adapted/exchanged. Guiding questions are (m1) "How can the device be transformed?", (m2) "Who can do it? (Employees or subcontractors)", (m3) "What equipment is needed?", (m4) "What are the costs of changing the system, doing the migration and running the changed end-device?".

After having collected the new requirements, another activity of the plausibility check is to answer whether a distributed solution for Req'_d may be found that satisfies all the B/L/O requirements. The cost-benefit considerations will have a dedicated focus on the safety, security and privacy risks associated with offloading S_d.

In Step 3 no concrete solutions for S'_d must be considered. Step 3 evaluates the influence of connecting an abstract functionality Req'_d to the end device and checks whether it is plausible to assume that later in the development process an implementation S'_d can be found that realizes Req'_d. Therefore, the approach examines the composition of

1. S-S_d, the end-device after S_d has been stripped off and which is equipped with the necessary TRANSACT components of the end-device, and
2. Req'_d which represents abstractly all possible distributed solutions S'_d.

If the realization of Req'_d is not plausible, the process steps back to determine a different set S_d of services to be offloaded.

In **Step 4**, functionality is assigned to the edge and to the cloud tier. The goal of Step 4 is an impact analysis of the networks on the distributed solutions for Req'_d. Any distributed solution of Req'_d must cope with the different service levels of the two network types of the TRANSACT architecture: the reliable network between device and edge tier, and the less reliable internet connection that is available between all tiers. Guiding questions for this decision are “What response times are required?” and “What reliability is needed?”, but also “Is the data distribution and storage in accordance with regulations?”.

The candidate solutions are specified as abstract as possible in terms of their requirements. To this end, their requirements are defined on

- the network connection between device and edge tier Req_{DE},
- the services that will be at the edge tier Req_E,
- the network connection between edge tier and the cloud tier Req_{EC} and
- the services that will be at the cloud tier Req_C.

Still no concrete implementation of the services on the edge tier (i.e. a solution of Req_E) or on the cloud tier (solution of Req_C) is required. By defining Req_E and Req_C it is specified what functionality will be realized where. This concretizes the data flow to some degree: What data is needed at the cloud to realize Req_C and what data is needed at the edge to realize Req_E.

Hence, in Step 4 also security and privacy aspects will be (re)evaluated and respective constraints on the technical solution and regarding legal requirements have to be specified. A guiding question moreover is “What certification/data protection guarantees are necessary for the to-be-outsourced data?”. Business requirements might be added to specify which services (and knowledge) should be developed inhouse and which can be outsourced. Business requirements should also state the value gained by outsourcing, the accepted costs for outsourcing, and required minimum contractual guarantees of the outsourced services.

Regarding the migration, maintenance and updating rough estimates will be used to assess whether the choice of Req_E and Req_C are plausible. Also, abstract failback mechanisms will be specified and the data transmission requirements and timing requirements will be specified.

After having collected the new requirements, a plausibility check has to answer whether the requirements are satisfiable.

For assessing whether it is plausible that an implementation of Req_{DE}, Req_{EC} can be found, the available network providers and the types of networks are considered. One focus in this step is on timing requirements and reliability. As a result of a thorough assessment, the initial versions of the fallback routines (designed at Step 3) may be adjusted and the according changes in the design must be propagated.

In **Step 5**, the edge requirement Req_E and the cloud requirement Req_C are further refined into component requirements. In Figure 5 we illustrated an example where Req_E is refined by the composition of

requirements Req_1 and Req_2, while Req_C gets refined by the composition of the requirements Req_a, Req_b and Req_c. These refinements concretize how the functionality will be implemented on the edge and cloud using the services available at the respective tier.

After specifying the requirements on the individual cloud and edge services, provider and the concrete services are chosen. Additional non-technical requirements that are important in this step are hence "Is sufficient customer support provided for the chosen cloud services?" "Is the communication between inhouse stakeholder and the provider easily possible?". The previously identified requirements regarding acceptable contractual guarantees/conditions/costs and regarding sufficient certification must be checked. Moreover, the data flow for monitoring and updating/deployment must be checked for the compilation of services. A migration plan that satisfies the requirements must be worked out with the involved service providers and agreed upon.

If no plausible implementation of the requirements can be found, the process steps back to Step 4 (or earlier steps). Stepping back to Step 4 allows a new assignment of functionality to the off-device tiers. Stepping back to Step 3 allows to off-load a different set of functions, stepping back to Step 2 allows to modify the targeted functionality *Req'* and stepping back to Step 1 would allow to exchange even the initial system.

This process of refining Req_E and Req_C to component requirements is often guided by the initial implementation of the functions F_{iii}, F_{iv} on the end device and by the added-value services on the respective tiers, that together can be used to realize the requirements. As in Step 4, the required service level has to be ensured, mode management exploiting fallback mechanisms will be used. This may lead to additional communications, so that stepping back to one of Step 4,3,2 or 1 may be necessary.

The Steps 1 to 5 together build a strategy to explore the design space for creating a distributed solution. After Step 5 has been successfully completed, the requirements for all components are specified along with requirements on B/L/O and constraints on the migration/updating. In the best case, the specified technical requirements allow to derive their implementations via a press button approach. The B/L/O requirements can evolve even after the technical implementation has been accomplished – for instance due to test results that induce change in the operational processes.

4.5 From Requirements to a Solution -- Verification and Validation

The above process describes how a monolithic pure on-device solution is (partly) distributed to cloud and edge and how requirements for the distributed solution can be derived based on the original solution's requirement and how the requirements can be (re-)established.

The process is not intended to specify a strict temporal order in which the respective requirements are further concretised. The order can be based on the need to analyse critical design decisions. Rather, the T&V[2] methodology outlines the key design steps that lead to a distributed solution and guides the exploration of the design space.

Since it must be shown that the final system fulfils the requirements and the complexity of the system is immense, the T&V[2] methodology can benefit from the use of contract theories. We can think of the requirements as contracts. Consider a contract theory that defines composition, quotient, and refinement of contracts. If we have shown that a contract C is refined by the composition of contracts C_1 and C_2, then the further evolution (refinement to final implementation) of each C_i can be done independently of the others. The contract theory guarantees that contract C is fulfilled. So, when using contracts, it is sufficient to prove the refinement relations concretizing the design. Thereby, it can be avoided to analyse the final implementation of S' as a monolithic system, instead the abstract contracts $C, C_1, \ldots$ are examined, and the C_i only refer to parts of the overall system. This allows a large reduction of the complexity. Although there is

a wide range of formal contract theories, not all aspects of a system are covered. Properties such as reactivity or security, for example, are such properties. This means that there is no general mathematical theory that defines how a monolithic requirement *Req* can be decomposed into more concrete requirements $Req_1, ..., Req_n$ so that the composition of solutions for $Req_1, ..., Req_n$ is guaranteed to be a solution for *Req*. Consequently, (i) structured, expert-based methods such as (Rakow 2021) Risk Storming must be applied to define the requirements during the design process, and (ii) holistic simulation approaches are important to verify that these requirements are met. It is not enough to test a single component, but the interaction of the solutions must be thoroughly tested, also for emergent behaviour.

Evidence is required to show that the nontechnical requirements are satisfied equally. As argued in (Rakow 2021), the different stakeholders should have various contracts that also relate to the involved infrastructure in order to make the update process predictable.

5 Transform safety-critical Cyber-Physical Systems

In this section, multiple examples of transformation, validation, and trade-off aspects for TRANSACT applications and use cases are presented. The diversity and complexity of the problems to be addressed indicate that abstract and high-level descriptions as the T&V[2] methodology presented in Section 4 can be helpful to cover the individual aspects in their entirety.

5.1 Model-driven Engineering for simulation-based design and trade-off analysis (UOC)

Model Driven Engineering (MDE) is an engineering approach that focuses on the use of models to design, analyse, and build systems. MDE seeks to reduce complexity in the construction stages of systems to speed up development and validation by using different strategies and solutions based on the use of models. For example, model transformation allows deriving concrete implementations from abstract models to automatically obtain solutions as source code or software artefacts. There are three main types of model transformations: model-to-model (m2m) transformations (e.g., to transform a high-level model describing the architecture of a CPS into a low-level model specifying the implementation details), text-to-model (t2m) transformations (e.g., to transform a machine description and characteristics to JSON[2] format), and model-to-text (m2t) transformations (e.g., to produce YAML[3] configurations from a deployment model). See also Figure 7.

To support the simulation-based design and analysis of the CPS following the TRANSACT architecture, this section proposes an approach based on model transformation for the execution of CPS simulations. Concretely, this approach enables the deployment of the system in a cloud testing environment following the CPS architecture to monitor the infrastructure. Figure 7 shows an overview of the approach and the main components involved.

- The *CPS Architecture Model* addresses the specification of the CPS architecture, including three main concepts: infrastructure, applications, and deployment. The CPS infrastructure involves the specification of sensors, actuators, edge nodes, cloud nodes, and messaging brokers (if the system implements asynchronous communication). The applications and services that address the system requirements are also specified in the architecture model. These applications correspond to the system functions that address the requirements (i.e., safety-critical, mission-critical, and non-critical functions). Finally, the deployment describes which nodes of the infrastructure will host the functions.
- *Code Generator* is the component that performs the model transformations to generate the simulation deployment and configuration files. This component takes the input model (CPS Architecture Model) and performs a series of M2T transformations to generate several YAML files. The files contain the code for deploying and configuring the functions using a cloud test environment. A monitoring system is also automatically deployed to monitor the infrastructure and quality attributes such as application and node availability.

[2] JavaScript Object Notation is a text format for data exchange.

[3] YAML is a data serialization format used in this study to represent Kubernetes objects.

- Finally, the last stage involves the *System Simulation and Monitoring*, where the generated code is executed to deploy the system (using the cloud test environment) and the monitoring system. A sensor emulator is also run to recreate data generation (using a historical data dataset) from the device tier sensors. The monitoring system provides information about the resource consumption (such as CPU, RAM, and Bandwidth consumption) and the availability of the functions and nodes. For example, it is possible to identify bottlenecks or overloaded nodes that may generate issues.

Using this generative approach, trade-off analysis can be easily implemented by performing small changes in the source CPS Architecture Model, e.g., it is possibly to analyse different deployment configurations (either in the edge or the cloud) or architectures by simply modifying a single parameter in the source models. By comparing the results when a specific configuration has changed, developers and operators can take decisions on the most suitable architecture, system configuration, or deployment configuration.

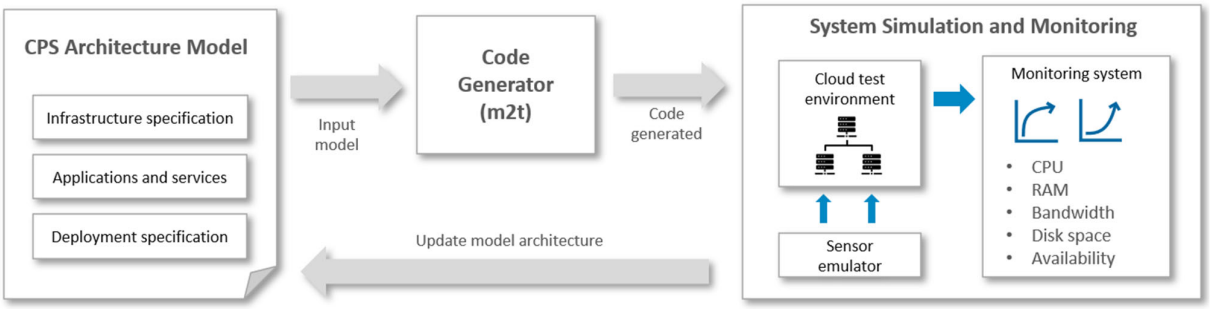

Figure 7: Model transformation approach to support simulation of the CPS architecture

In this approach, the architecture implementation is simulated in a Cloud test environment. Additionally, the generation of device layer sensor data is simulated and controlled by means of a *Sensor emulator* (a Python script that can read a historical sensor dataset and emulate the data generation).

In UC5, this approach was designed and focused on the specification of the WWTP CPS architecture, model transformation, and system simulation. The main components developed are as follows.

- A Domain-Specific Language (DSL) for the specification of the CPS architecture. This DSL was developed using MPS[4] providing a mix of editors and multiple notations such as tabular, graphical, and textual.
- A *Code Generator* built in MPS to perform model transformations and generate code. This component generates a group of YAML files for the deployment of the functions (using software containers) and the system monitoring in the cloud testing environment.
- A *Sensor Emulator* to emulate the device tier sensors. This component generates data (like real sensors of the WWTP) reading a historical dataset of five variables: total suspended solids (TSS), chemical oxygen demand (COD), electrical conductivity, pH, and temperature.

Finally, the modelled architecture can be deployed in a test cloud environment by provisioning virtual machines at a cloud service provider. The data generation frequency of the *Sensor Emulator* can be modified to evaluate the capabilities of the infrastructure and functions deployed. Architecture design decisions and trade-off analysis can be made based on the data captured by the monitoring system.

[4] Meta Programming System of JetBrains

To sum up, the *CPS Architecture Model* is transformed to support the simulation of the infrastructure and functions of the system. For instance, the TSR40 (the architecture should support the usage of message brokers) is an availability requirement that could be evaluated using this approach as follows. First, to specify the device-edge-cloud continuum architecture model including the functions and message broker(s); then, using the code produced by the Code Generator, deploy the system in the cloud testing environment; Finally, test and analyse the system by modifying the number of clients and the frequency of sending data to the broker(s). The monitoring system collects several resource consumption metrics and displays system information about the availability of nodes and applications (including messaging functions and brokers). If a fault is detected, the architectural model can be modified/upgraded to run the simulation again.

5.2 Design Language for Cloud Applications (KUM)

5.2.1 Overview

With the shift from traditional to distributed CPS, utilizing the three-tier computing continuum, cloud services providers encounter new challenges related to the specific requirements and complexities of deploying and managing safety-critical CPS in a distributed environment. In order to address these challenges, specialized infrastructure, services, but also operational practices must be developed to handle to the unique demands of distributed safety-critical CPS solutions.

Cloud applications are typically composed by several pieces or components deployed over several servers in a cluster. Those pieces collaborate to perform the cloud application tasks, usually related to user requirements.

Which components are part of a cloud application and which servers run each component can dynamically change over time. Both software and hardware can be updated or are scaled while the cloud application is running. Service providers and maintainers also need to know how their cloud applications are performing and which issues arise over time.

Installing, configuring, and monitoring the cluster's nodes and the cloud application's components manually is an almost impossible task and some sort of tools and automations are needed. As an example, shell scripts can be used to install, configure, update, and monitor both the software and the underlying infrastructure. Unfortunately, shell scripts can be difficult to maintain and usually require a deep knowledge of the system being installed. This is why the majority of cloud applications used in production environments employ more advanced tools, while shell scripts are delegated for some minor tasks.

Using a tool implies declaring to that tool what it should do using some sort of language. The choice of tools and languages determines how complex and error prone the management of the cloud applications and the underlying clusters will be.

5.2.2 Choosing the Language

The following three aspects can improve the tools' usability and maintainability:

- Separation of concerns: taking into account the roles of the different people involved in a cloud application life cycle.
- Readability and maintainability: the language should never add complexity to tasks.
- Technology agnostic description: a description should only be coupled to whatever it is describing and not to any other external technology.

Separation of Concerns

Several roles are involved in the management of any cloud application used in production, especially in applications of a certain size with demanding service quality requirements. Just as an example, components developers and application operators have different requirements. A good cloud application description language should allow each role to focus on what they know or need and forget about the rest.

As an example, a cloud application might require an administrator password for its configuration. The operator sets this password, and the developer knows how this password is provided to the authentication component (configuration file, environment variable). If both things (the password value and how it is exposed to the authentication component) are included in the same file, that file requires both roles, so that file has not a clear owner. If in one file the configuration parameter values of the cloud application are provided and in another file it is described how that configuration is exposed in the authentication component, each role can work independently and not get lost in aspects they don't fully understand and should not worry about.

Readability and Maintainability

As explained in the overview, shell scripts are hard to read and hard to maintain. A conceptually minor change might require a lot of time and be very error prone even for an expert. Furthermore, scripts tend to be patched and extended over time, which typically makes them less readable and less maintainable.

One of the problems with shell scripts is that they include sequences of operations and mix *what* should be done with *how* it should be done. In a purely descriptive language, the complexity is only related to *what* is being described, no matter how many changes have been made to that file.

Technology Agnostic Descriptions

One of the traditional problems of some tools is that they are tightly coupled to the underlying technology. As an example, Helm is a widely used tool to ease the process of managing services in a Kubernetes cluster. A Helm chart is a recipe describing how a cloud application is deployed and configured. Helm charts are based on templates and, hence, are tightly coupled the underlying format of the final files (in this case, Kubernetes objects). If you create a Helm chart to deploy your application, this chart can only be used in a Kubernetes compatible cluster. To deploy it in a cluster managed by a different container orchestration system that doesn't use Kubernetes, objects and the templates are useless.

A good cloud application description must focus on just describing your cloud application components using an technology agnostic language. The tool reading that file is responsible for converting this to the underlying orchestration system low-level language.

5.2.3 The Kumori Service Model

Kumori Service Model (KSM) is the language that has been used in Use Case 5 and in the horizontal demonstrator. This model was successfully employed throughout the TRANSACT, while the Kumori Platform evolved during project in order to address the requirements of the different scenarios, while the underling The language is strongly based on the three pillars exposed in the previous sections:

- Separation of concerns.
- Readability and maintainability.
- Technology agnostic descriptions.

KSM is used to describe services in Kumori clusters but the description is technology agnostic. It is based on the following principles:

- Designed for describing services (cloud application).
- Declarative: it is purely descriptive and can be potentially converted to any target technology.
- Separates between service configuration, service description and component description.
- Uses a high level of abstraction.

The language is based on the declarative language CUE (CUE n.d.). It differentiates between:

- Artifact: describes a cloud application, including which configuration and resources (volumes, CPU, RAM, ...) it requires.
- Service Application: configuration values and resources used for a concrete execution of an artifact in a given cluster.

It also identifies two different types of artifacts:

- Component: an atomic runnable artifact usually associated to a docker image. In other words, a microservice.
- Service: a distributed artifact composed by several interconnected roles. Each role is an artifact.

Component manifests (Kumori n.d.) are written by components developers. Service manifests are written by service architects. Service application manifests are written by operators.

Artifacts

Every artifact manifest declares at least the following elements:

```
#Artifact: {
  ref: name:  "frontend"                                          <1>
  description: {

    srv: {                                                        <2>
      server: {
        restapi: { protocol: "http", port: 8080 }
      }
      client: {
        database: { protocol: "http" }
      }
    }

    config: {
      parameter: {}                                               <3>
      resource: {}                                                <4>
    }

  }
}
```

<1> The artifact name

<2> The artifact interconnection points (in which ports it listens and which external elements it requires connecting to)

<3> Which configuration parameters it requires

<4> Which resources (volumes, ...) it requires

A component is a specialization of an artifact backed by a docker image:

```
#Artifact: {
  ref: name:  "frontend"
```

```
  description: {

    srv: {
      server: {
        restapi: { protocol: "http", port: 8080 }
      }
    }

    config: {
      parameter: {}
      resource: {}
    }

  probe: {...}                                                    <1>

    code: {

      frontend: {
        image: tag: "kumoripublic/examples-hello-world-frontend:v1.0.6"     <2>

        mapping: {                                                <3>
          filesystem: {...}
          env: {...}
        }
      }
    }
  }
}
```

<1> How the component health can be checked.

<2> The image containing the component binary.

<3> How the configuration and resources are exposed in the docker container (environment variables, files, path).

```
A service is another specialization of an artifact.
 #Artifact: {
  ref: name:  "frontend"
  description: {

    srv: {
      server: {
        restapi: { protocol: "http", port: 8080 }
      }
    }

    config: {
      parameter: {}
      resource: {}
    }
    role: {                                                 <1>
      helloworld: {
        artifact: helloworld.#Artifact
        config: {...}
      }
    }

    connect: {                                              <2>
      cinbound: {
        as: "lb"
        from: self: "restapi"
        to: helloworld: "restapi": _
      }
    }
  }
}
```

<1> Declares which artifacts are part of this service and their roles. It also declares how the service configuration is provided to that artifact.

<2> Declares how the roles are interconnected.

Note that the descriptions do not depend on where they are going to be deployed and are purely descriptive. Kumori tools know how to run services in a Kumori cluster using the information provided in a deployment manifest and the underlying artifacts manifest. However, the same information can be used to deploy that service on any other cluster managed by any orchestrator by using the appropriate tool.

Service Application

A service application is the execution of an artifact in a cluster. It contains the values assigned to each artifact declared parameter and the resources assigned to each resource dependency declared in the artifact:

```
#Deployment: {
  name: "helloworlddep"          <1>
  artifact: s.#Artifact          <2>
  config: {
    parameter: {                 <3>
      myparam: "myvalue"
    }
    resource: {                  <4>
      myvolume: "volume1"
    }
    scale: detail: {             <5>
      frontend: hsize: 10
    }
    resilience: 0
  }
}
```

<1> The name of the new service

<2> Which artifact is being executed

<3> Assigns values to parameters

<4> Assigns resources to the new service

<5> Initial scale of the artifact elements

Note that the configuration provided to a given service execution is completely decoupled from the service description.

5.3 Traffic control (VIN)

5.3.1 Use case introduction

Traffic control systems are used to control the traffic flow at intersections, where traffic from different directions can be conflicting. A classical case is an intersection with traffic control lights that turn green one by one of are controlled via inductive loops in the road.

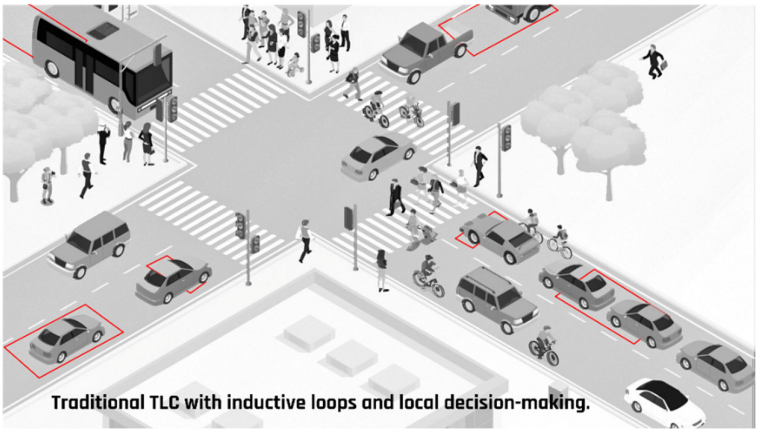

Figure 8. Classical intersection where traffic is controlled with traffic lights.

A recent development in the Smart Mobility domain is the adoption of CPS. Sensor signals, but also geo-data from connected vehicles are sent to the cloud where a traffic control application controls the traffic lights.

A standardization is ongoing for a Cooperative Intelligent Transportation Systems (C-ITS) Day-1 use cases as defined by the European Union: traffic light optimization, traffic signal priority and Green Light Optimal Speed Advice (GLOSA). These use cases have led to new value chains for traffic light control.

- Service providers of in-vehicle applications are now able to inform their customers about the actual and predicted signal states (time-to-green, time-to-red) and translate this into a Green Light Optimized Speed Advisory (GLOSA), creating extra value for their services.
- Logistic companies, public transportation and other modalities are able to get a higher priority at signalized intersections, reducing delays and fuel consumption.
- Better optimized traffic lights for traffic flows have a positive impact on the liveability in urban areas. The traffic throughput can be increased, while at the same time emissions (CO2, NOx, etc) are reduced.

The overall CPS consists of multiple layers: Traffic Light Controller (TLC) Facilities, ITS Applications, Cloud Services and Information Services. The communication between the different components of the CPS makes use of the latest international ETSI standards (CAM, SRM, SSM, SPaT, MAP, TLC-FI and RIS-FI) for sharing information between the different systems and services. These ETSI standards are globally recognized, making cross-border cooperation between systems and vehicles possible (very relevant for international transport!). The way in which these standards are incorporated in the Talking Traffic architecture is adopted by C-Roads. C-Roads is an international organization of road authorities representing the most European countries, both also Israel and Australia. This means that the developed architecture, products, and services can be implemented with minimal effort in these countries.

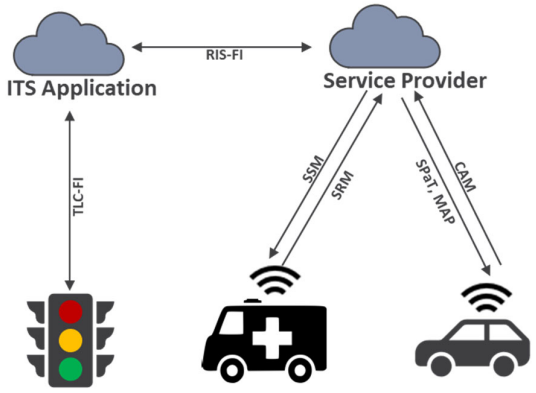

Figure 9 CPS architecture (simplified)

In the CPS, the ITS Application is the service responsible for scheduling green phases for the controlled intersection. Both traditional data (loop detectors) and information from connected vehicles are used as input to schedule green phases in advance, allowing them to be broadcasted to arriving vehicles for the GLOSA use case.

The higher the penetration rate of connected vehicles is, the better the three use cases can be executed. Currently the quality of the GLOSA service is low as the penetration of these connected vehicles via in-car or smartphone systems is less than 20%. Please note that this only applies to vehicles, for pedestrians and cyclists there is limited to no data available.

To improve the penetration grade, ViNotion developed a visual sensor that can measure the status of all vehicles from the side of the road. On a larger distance from the traffic light, the vehicle position and timestamp are registered, including the speed and the type of vehicle.

5.3.2 Trade-offs for the CPS

The transition from a conventional traffic control systems toward a CPS has many benefits:

- Having more information about the approaching traffic toward an intersection allow a more optimized green-schedule to improve the traffic flow.
- Controlling multiple interactions for a whole area, from a centralized controller in the cloud, allows overall traffic control to optime traffic flows on a macroscopic level.
- Gathering information from multiple sources such as inductive road loops, floating car data, weather conditions, pollution conditions, etc. are more ease to collect via IoT with a centralized application in the cloud.
- New sensors such as cameras with AI for object classification allow the application to include cyclist and pedestrians.
- Any policy can be implemented or dynamically changed. For example, if fine dust or CO2 levels are too high, the traffic controller can prioritize trucks because stopping and accelerating truck produce more emission.
- A cloud application with standardized interfaces to connect with the physical world allows any supplier to provide the traffic application software.

- Maintenance, improvements, and enhancements of the traffic application is easy to deploy with a cloud application.

The main trade-offs between a local traffic control system and a CPS-based traffic control system are the following:

- Safety: conflicting traffic-light schedules must not cause dangerous situations. Using a CPS-based system adds additional risks on availability because the Internet and wireless communication are shared resources that are under control by 3rd parties. These risks can be mitigated as follows:
  - The physical on-site traffic control hardware should comprise a local control application as fallback if availability of the cloud-based application is jeopardized.
  - The performance of a CPS from a supplier should be exposed to a strict certification process to proof that the availability and fallback implementations are well-covered.
- Latency: The control loop must be fast. Information from the sensors that is sent to the cloud-based application via a broker over the Internet and the feedback signals to control the traffic lights should enable green/red lights in phase with the traffic. To mitigate this risk, the following measures can be taken.
  - The maximum end-to-end latency is standardized. Late messages are ignored.
  - If the messages are ignored the quality of the traffic control is reduced. The supplier of the system will be held responsible. Non-compliance is punished by the broker and means the supplier cannot deliver its service to the customer.
- Costs: Processing and communication should be optimized.
  - As explained above, a cloud-based application has many benefits. Another advantage is that resources in the cloud can be shared with other applications at times that the traffic control application is not using the resources. However, AI-based video analysis is performed 24/7 with a fixed amount of compute resources. It appears that hosting this task in the cloud is relatively expensive as opposed to computing on the edge.
  - As mentioned above, local video analysis prevents the need for expensive cloud resources 24/7. Moreover, video data is instantly converted to traffic data which does not contain personal information. Hence, the privacy of people is protected.
  - The traffic data comprises a relatively small amount of data communication as opposed to video data. Consequently, edge processing limits the amount of data is significantly reduces the data bandwidth for communication to the cloud. This is another cost saving.

5.3.3 Transition toward CPS

A CPS-based traffic control application and an AI-based camera as sensor to collect data about all traffic participants near an intersection allows advanced traffic management as mentioned above. Figure 10 shows how all traffic participants are observed by road slide cameras that contain an edge device for video analysis. The data is transmitted to a cloud-based traffic control application.

To implement such a system, additional constraints on latency and availability are introduced. Typically, the local traffic-control equipment also hosts a relatively simple traffic controller, next to the connectivity towards the cloud application. This local controller implements a fallback control for cases where the latency and availability constraints are not met.

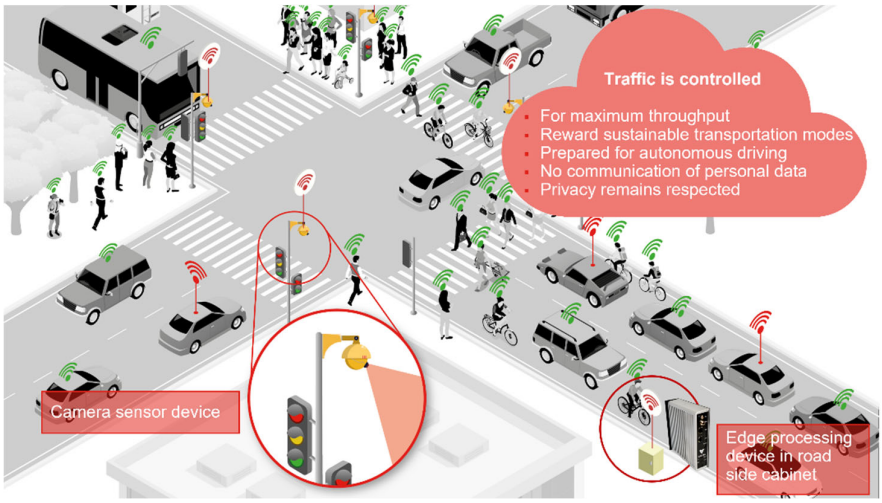

Figure 10. An Intersection CPS-based traffic control with traffic lights.

In addition, the local controller allows the road authority to upgrade traffic control systems to be cloud ready. Also, newly built intersections with traffic control equipment can use the cloud-ready equipment. If only inductive loops for traffic detection are used and other data sources are not exploited, the connection a cloud-based traffic control application is limited. Once the road authority decides to implement special policies, a cloud-application provider can easily connect to the traffic control equipment and take over the control.

Because the additional price for cloud-ready traffic control equipment is limited, most road authorities choose this option to be future ready. They can easily start with a traditional traffic control and easily switch to a cloud-based control.

The standard allows a road authority to choose between different suppliers of traffic control applications to stimulate a competitive ecosystem of suppliers. As such, ViNotion provides the edge-based traffic camera to collect traffic data which is completely independent of the traffic control application that exploit the traffic data.

5.4 Cloud-Featured Battery Management System (AVL)

The design and development process (see left side of the “V” in Figure 3) can be considered from three different points of view (Martin Glinz 2024), also named domains (see

Figure 11):

- Requirements Domain
- Behavioural Domain (a.k.a. functional or logical)
- Physical Domain

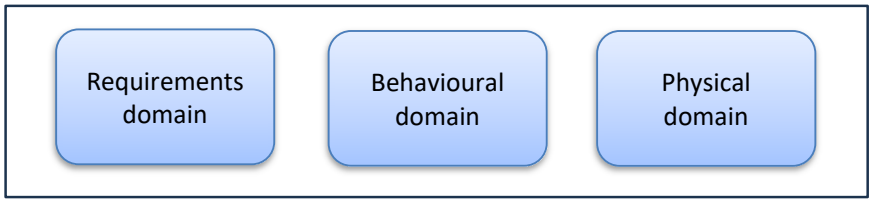

Figure 11: The three domains in systems engineering.

All three domains need to be aligned with each other.

According to the International Requirement Engineering Board (IREB), requirements are categorized in two groups: functional requirements and quality attributes (IREB n.d.). While the functional requirements cover the behavioural domain, the quality attributes strongly define the overall software architecture and are mainly reflected in the physical domain. In this project, however, the target architecture is already defined. Hence, the task is then rather to decompose the existing (or new) function and assign it to the corresponding tier and physical architecture element.

The following description provides a guideline for this decomposition and mapping process in the context of the automotive use case. Table 4 lists the most relevant quality attributes.

Table 4: Quality attributes relevant for software architecture design decisions.

Quality Attributes	**Description**
Adaptability	Describes the ability of a software system to be modified or extended easily to meet changing requirements or environments.
Extensibility	Describes the ease with which a software system can be enhanced or expanded through the addition of new features or functionalities.
Interchangeability	Refers to the ability of components or modules within a software system to be replaced or substituted with alternative implementations without affecting overall functionality.
Interoperability	Describes the ability of a software system to interact and operate seamlessly with other systems or components, regardless of differences in platforms, languages, or protocols.
Maintainability	Indicates the ease with which a software system can be repaired, modified, or updated over time, typically measured in terms of code readability, documentation, and architecture.
Performance	Refers to the speed and efficiency with which a software system executes its functions and processes data.
Persistency	Persistency in software refers to the ability of a system to retain data and state across different sessions or instances of execution. It involves storing data in a way that survives system restarts, shutdowns, or failures. This ensures that valuable

	information is preserved and can be accessed later, providing continuity and consistency to the user experience.
Reliability	Ensures that a software system consistently performs its intended functions correctly and predictably under various conditions, without unexpected failures or errors.
Safety	Ensures that the software system operates reliably to minimize risks of harm or damage to users, equipment, or the environment.
Scalability	Indicates the ability of a software system to handle increasing workload or growing user base without compromising performance.
Security	Ensures that the software system protects data and resources from unauthorized access, modification, or destruction.
Standards Compliance	Refers to the adherence of a software system to established industry or regulatory standards, ensuring compatibility, interoperability, and quality assurance.
Testability	Describes the ease with which a software system can be tested to verify its correctness, functionality, and performance, typically facilitated by modular design and comprehensive test suites.
Timeliness	Timeliness in software refers to the ability of a system to respond or produce results within a specified timeframe or within an acceptable period. It's often associated with real-time or near-real-time systems where actions or responses must occur within strict deadlines to maintain usability, effectiveness, or compliance with requirements.

The approach comprises the following steps:

1. Physical decomposition of the system.
2. Assessment of the physical elements according to quality attributes.
3. Functional decomposition.
4. Mapping of functional and physical domain w.r.t quality attribute.

For the sake of simplicity, the mapping with the Requirements Domain is skipped. However, the functional requirements are considered implicitly in the context of the functional analysis and decomposition.

Example: Computation of DC-Resistance of the Battery Cells.

Use Case: An algorithm shall compute the individual DC cell resistance of the battery pack.

1. **Physical Decomposition of the System:**

   Device Tier: Battery Management System (BMS; embedded system)

   Edge Tier: LTE Gateway

   Cloud: Cloud System

2. **Assessment of the physical elements according to quality attributes.**

   Assessment scores are shown in Table 5.

3. **Functional Decomposition:**
    a. Read cell voltages and current.
    b. Store data persistently.
    c. Compute cell resistance.
4. **Mapping of functional and physical domain w.r.t quality attribute.**

    By comparing the needs of the functions and the assessment of the quality attributes per tier, one can identify the following best overlaps:

    - *Read* shall be allocated to the BMS.
    - *Store* shall be allocated to the Cloud.
    - *Compute* shall be allocated to the Cloud.

Table 5: Assessment of quality attributes per physical component and function: Assessment score ranges from 1 (very good) to 3 (poor); need from the functional view ranges from 1 (very important) to 3 (minor relevance).

	Physical Domain			Functional Domain		
Quality Attribute	BMS	Gateway	Cloud	Read	Store	Compute
Adaptability	3	2	1	3	2	1
Extensibility	3	2	1	3	2	1
Interchangeability	3	2	1	3	3	3
Interoperability	2	2	1	3	3	3
Maintainability	3	2	1	3	2	1
Performance	3	2	1	3	1	1
Persistency	3	2	1	3	1	1
Reliability	1	2	2	1	2	2
Safety	1	2	3	3	3	3
Scalability	3	3	1	3	1	1
Security	3	1	1	3	1	1
Standards	1	3	3	1	3	3
Testability	3	2	1	3	1	1
Timeliness	1	2	3	1	2	2

Summary

The demonstrated approach shows how functional and physical domain can be mapped. It is noteworthy that there may also be other design decisions besides of the quality attributes, e.g. the clustering of consecutive functions.

5.5 Build once, run anywhere for medical software (PMS)

5.5.1 Problem description

The medical domain is highly regulated. This regulation and the fact that healthcare providers tend to be risk-averse make the introduction of new compute platforms like edge and cloud slow. A design that is completely tailored to work on an edge or cloud platform would require adoption of these new platforms. Due to the slow adoption and the hard requirement for the new platform, this could reduce the rollout pace which in turn reduces revenue. Often development is pushed into a mode where one solution can be offered to all customers, which could result in either innovation slowdown or additional costs due to multiple different implementations.

5.5.2 More than just containers

The slogan “build once, run anywhere” is derived from the slogan "Write once, run anywhere” created by Sun Microsystems in 1995 for their cross-platform Java language. Recently this phrase has been expanded to include containers, meaning that developers can package their entire application and all its dependencies in a container and run it anywhere.

This is a nice theory, but large (medical) applications tend to consist of a lot of components and have many external dependencies (e.g., databases, devices). So, by just wrapping all the software in a single container we don’t solve those problems. This is why for new product developments, we strive for a microservice architecture with well-defined interfaces. These new architectures are than be mapped onto different platform configurations to verify which options are viable and whether the interaction patterns can work given the constraints the different platforms have.

5.5.3 Decouple development from cloud adoption

Due to this new architecture, including mapping on the different platforms, we are now able to start development of cloud native/ready applications without requiring our customers to onboard to the cloud. This makes our products future-ready, where initially local deployments (only using the device tier) will be the default, but eventually more and more functionality can move to the other tiers. 45his can happen without major rework of the components, hence “build once, run anywhere”.

It is critical for new innovations that development teams get familiar with the new architecture and way of working. This ensures that the teams can focus on the new (medical) innovations and don’t have to worry about where their applications are running.

5.5.4 Demonstrator

New features, which usually require more advanced computation capabilities, will be introduced on edge or cloud tiers. In use case 4, a demonstrator is built showing a cloud-based reconstruction application. In this demonstrator, the components have been decoupled, allowing for fast updates and innovations of the used algorithms. An added feature in this demonstrator is a web-based viewer hosted in the cloud. This allows both local and remote viewing of the acquired data in a 3D viewer in the browser.

5.6 Cloud-based solutions for medical image exploration (PST)

5.6.1 Use Case Introduction

In patient diagnosis and treatment planning, image acquisition devices such as CT and MR devices create image data of the patient anatomy. For efficient treatment planning this data should be easily available for analysis by health care professionals in a diverse set of applications. Currently, however, most processing is done on the image acquisition device, which limits the number and types of applications that can easily use the data. By introducing an edge/cloud platform that includes the image acquisition device, easy data access for other applications can be extended, without changing the properties of the image acquisition device.

Vesalius3D is a medical visualization software application used in healthcare for interpreting medical imaging data, such as CT scans, MRI images, and ultrasounds. It is thus an example of a third-party application that would benefit from such an edge/cloud platform. Vesalius3D users can create insightful patient specific anatomy visualizations medical from medical image data. These 3D visualizations emphasize the regions of interest of the patient specific anatomical structure and exploration with intuitive navigation tools can yield important insights in the patient's condition. Traditionally, only the radiologists and surgeons benefit from these insights. A cloud deployment of Vesalius3D can open exploration of the 3D visualizations to a larger group of the health care chain. The visualizations and insights can then be shared along a larger range of stakeholders in the patient's treatment path, ranging from the radiologists and surgeons to e.g. physiotherapists, general practitioners, and of course the patients themselves. Through the increased shared understanding of the patient's condition, the treatment path can then be better optimized.

Ensuring smooth interaction with Vesalius3D is one of the key challenges. It is crucial for users to effortlessly navigate the 3D visualizations to the correct viewpoint of their region of interest to gain insights of the patient's condition. However, Vesalius3D's volumetric rendering demands substantial GPU rendering power, which is often unavailable in conventional workstations and acquisition devices. Smooth interaction can be achieved by leveraging the edge/cloud platform to deploy Vesalius3D on workstations with powerful GPUs on the edge or the cloud.

5.6.2 Trade-offs between visualization on edge or cloud

Edge and cloud deployment of Vesalius3D have different benefits and challenges. In the edge deployment (Figure 12), powerful non-standard workstations are deployed in the hospital environment. Healthcare professionals perform their analyses with Vesalius3D on these workstations. If the workstations need to be shared with multiple people in the department, dedicated physical work locations also need to be allocated. For the cloud deployment (Figure 13), Vesalius3D is embedded within the Philips IGT Cloud Platform. Any workstation with access to the cloud platform can then work with Vesalius3D.

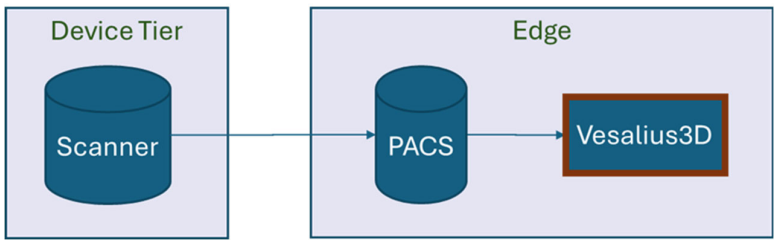

Figure 12: Platform deployment of Vesalius3D on the edge.

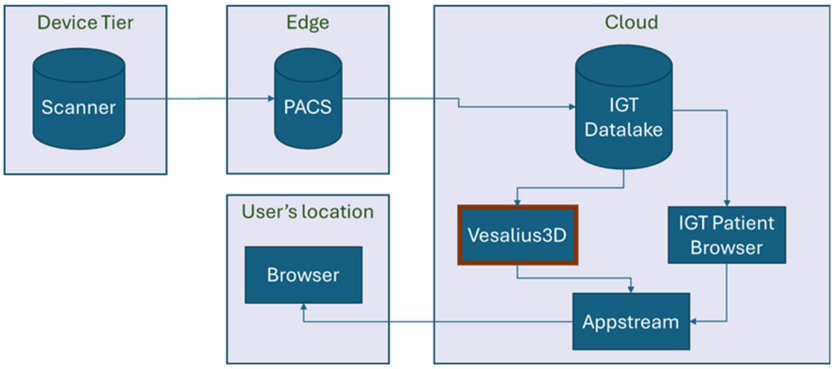

Figure 13: Platform deployment of Vesalius3D in the Philips IGT Cloud Platform

The main trade-offs to consider between these solutions are:

- Smoothness of Interaction
- Development Process
- Scalability
- Application access
- Costs

Smoothness of Interaction

The smoothness of the interaction is determined by two factors:

- The GPU of the workstation.
- The time latency between user input and the response.

To ensure smooth interaction, Vesalius3D's responses to user interaction should appear smooth in motion. Given that Vesalius3D's visualizations utilize volumetric rendering, refreshing and updating these in real-time puts strain on the GPU of the workstation. Thus, achieving smoothness in motion requires Vesalius3D workstations to have advanced GPUs capable of processing volumetric renderings efficiently. For both the edge and cloud deployment, smoothness can be achieved by selecting workstations with appropriate GPUs for Vesalius3D. In edge deployment, this may increase the upfront investment costs, while cloud deployment will incur higher variable usage costs.

In edge deployment latency between the user input and the responses is typically minimal due to the direct connection to the workstation. The latency in cloud deployment is, however, depends largely on the deployment's physical location and internet connection quality. To optimize smoothness of interaction, it's preferable to deploy the cloud infrastructure close to the customer's location.

Scalability

Scalability for Vesalius3D is the ability to modify the number of users that can work with Vesalius3D. Having better scalability can allow different product offerings, such as temporary additional work seats for a limited amount of time.

Edge deployment is not very scalable. Every user claims a workstation for the whole Vesalius3D session. Thus, scaling up the number of users that can work with Vesalius3D at the same time would involve acquiring additional workstations and deploying them. This is costly and can easily take days to weeks. Scaling the

number of users down can be more easily achieved by reducing the number of Vesalius3D licenses. In the case of dedicated shared workstations, the workstations also need to be repurposed or removed to free the office space.

In contrast, cloud deployment is very scalable. Following an onboarding process, in which the hospital IT departments define user access to the cloud platform, users can initiate and terminate sessions as needed. This flexibility allows Vesalius3D to be offered as a pay-as-you-go service. This approach opens opportunities for Vesalius3D to reach new customer segments at the very end of the health care chain, consisting of those that may not frequently work with medical images, but can benefit from the insights of their 3D visualizations, such as the physiotherapist or the general practitioner.

Application Access

Vesalius3D excels in providing clear visualizations of patient-specific anatomical structures, making it well-suited for treatment planning discussions and patient education. For these scenarios easy application access is essential.

In the edge deployment scenario, Vesalius3D is limited to use on the deployed workstations, restricting any discussions about patient cases to the physical location of these workstations. This limitation narrows down the available locations for workstation placement, potentially affecting accessibility and collaboration.

The cloud deployment removes this constraint, allowing Vesalius3D to be accessed from any workstation connected to the cloud platform. This enhances flexibility and accessibility, facilitating discussions at various locations within the healthcare facility or remotely. Additionally, reports and visualizations can be more easily shared with the patient and the other healthcare practitioners involved with their care. The subsequent enhanced understanding and recall by both patients and the healthcare practitioners can contribute to more efficient and productive meetings about the patient's condition and ultimately to improved treatment outcomes.

Costs

Cloud deployment incurs different fixed and variable costs compared to edge deployment of Vesalius3D. Edge deployment requires an upfront investment in powerful workstations and additional physical workspace. Afterwards, the fixed costs consist of licensing and software maintenance. Variable costs can occur due to customized workstation maintenance and repairs, both of which will have to be carried out by the hospital's IT department.

Cloud deployment does not require special workstations, thus an upfront investment for an existing hospital is most likely not required. The fixed costs will cover licensing, software maintenance and the maintenance of the cloud platform by the cloud platform provider. Variable costs consist of the costs incurred by using the cloud infrastructure. These variable cloud costs can greatly exceed expectations if care is not taken to ensure that the customer's use of Vesalius3D stays reasonably close to the initial estimates.

The direct costs of cloud deployments will likely be higher than the direct costs of edge deployment for most customers. However, some of these costs can be justified by cost reductions elsewhere. Maintenance and update procedures will for example no longer involve the IT department. Since special workstations are not required on site the customer can be more efficient with physical office space. And through increased understanding of their condition, meetings with patients can be more efficient.

5.6.3 Transition toward CPS

To tap into new market segments at the end of the healthcare chain, Vesalius3D should adopt cloud deployment. This approach offers scalability, smooth interaction, and the flexibility to use Vesalius3D anywhere. These features do not only enhance Vesalius3D for the current users, but also provide the means

to develop products tailored to users that would only consult the patient specific anatomy visualizations sporadically.

A practical transition could be based on the Use Case 4 demonstrator involving Vesalius3D. The demonstrator is based on the architecture displayed in Figure 13 and embeds Vesalius3D in the Philips IGT Cloud Platform. A lightweight communication layer was implemented around Vesalius3D to enable data transfers from the IGT data lake. This facilitated a swift migration, allowing for early testing with a nearly feature-complete Vesalius3D.

The next step in the transition involves an analysis of the business case. The technical considerations must be aligned with business and product objectives to check whether a viable business model can be found.

5.7 Discrete Event Simulation for Edge-Cloud-Based Clinical Application Platform (PFLH)

5.7.1 Use Case Introduction

Use case 4 explores how the TRANSACT methodology can be applied to improve workflow and interoperability in and between hospitals. In this context, Discrete Event Simulation (DES) has been investigated as a method to simulate clinical workflows and to guide the transition towards an Edge-Cloud-based clinical application platform.

Diagnostic imaging systems are safety-critical applications that require careful consideration. In the past, image guided therapy (IGT), computed tomography (CT) and magnetic resonance imaging (MRI) systems, have been developed and designed as self-contained systems. To guarantee optimal responsiveness even in times of peak loads, tightly integrated compute resources have been employed to control data acquisition, image formation and processing. While certain functionalities, such as those related to interventional imaging will continue to be integrated into the device, many of the aforementioned systems offer potential for virtualization and deployment on the edge or in the cloud. This will improve resource utilization, enhance scalability, flexibility, and accessibility, paving the way for the development of future innovative solutions.

In the context of other clinical systems, such as Picture Archiving and Communication Systems (PACS) and diagnostic workstations, there is already an ongoing trend toward cloud-based solutions for storing and processing medical image data.

However, when it comes to workflow and performance aspects, it is crucial to consider not only individual systems and their deployment across the TRANSACT tiers but also their interrelations. In this context DES is a powerful methodology that can be used to explore performance trade-offs and to identify optimal deployment strategies. It can help to identify sweet spots for deployment and ensure that the necessary workflow and performance requirements are met.

5.7.2 Transition Towards CPS

When considering diagnostic imaging, the clinical workflow is complex and involves multiple stakeholders, including patients, technicians, interventionalists, radiologists, as well as various technical systems, such as the imaging equipment and diagnostic workstations and image archives.

Virtualizing software components and functionality in the diagnostic workflow using the TRANSACT reference architecture offers a broad range of benefits. By removing the dependency on (often underutilized) local resources and enabling easy access to edge and cloud resources, resource utilization will become more efficient. The adoption of distributed CPS allows for the integration of new functionalities beyond the current

resource constraints, while ensuring easier maintenance through updates and facilitating the integration of 3rd party solutions. Additionally, the adoption of cloud storage can improve the availability of data, thereby leading to better data exchange, streamlined workflows.

The distribution of computational tasks across the three tiers of local, edge, and cloud infrastructure requires a trade-off of several competing aspects, including speed, system reliability and availability, transmission bandwidth, cost. As far as safety-critical systems are concerned, such a trade-off will usually need to be evaluated for individual events, rather than on an average basis. For example, in a clinical environment employing a distributed architecture for diagnosis and treatment systems, it must be ensured that the health conditions of each individual patient are considered in the trade-off of speed and reliability of data analysis. Optimizing just the average result turn-around time may have severe consequences for individual emergency cases.

Methods used for analysing and optimizing these trade-offs within distributed architectures must therefore work based on individual cases instead of or in addition to statistical descriptions. DES is a method that is particularly suited for the simulation of complex processes on an individual event basis.

Clinical processes in radiology and image guided therapy (IGT) are complex and interwoven. Due to the interdependency of clinical resources, delays in one procedure step can have an impact on the whole remaining schedule and even on the quality of care of other patients. Workflows may need to be adapted and reordered on the fly, for example to prioritize emergency cases. At the same time, the need for computational power increases, as more complex image reconstructions and evaluations are integrated in diagnosis and decision making. The interdependencies within the clinical processes and the different requirements for result turn-around times need to be considered when distributing computational load across on-premise, edge, and cloud infrastructure.

A thorough understanding of the clinical processes and methods to simulate and predict the impact of computational load distribution are therefore essential for a successful transformation from today's tightly coupled integrated on-premise devices towards flexible DCPS architectures. At the same time, it is important to have already reliable predictions for scaling the architecture to cover future computationally intensive technologies, such as advanced medical image processing.

5.7.3 Clinical Process Modelling and Discrete Event Simulation

The task of modelling and simulating clinical processes is challenging. One challenge is that the required information about real-world processes is not easily accessible. Hospital IT environments can be quite inhomogeneous, making it difficult to extract time stamps of operational events. While some operational information is only available within individual medical devices (such as image reconstruction events performed by a medical imaging system), others are stored in hospital-wide IT systems (such as time stamps of patient arrival or report submission in a hospital information system). Some prior descriptive analysis studies have relied on log data extracted from individual imaging systems to evaluating system utilization (Gunn 2017), and statistical analysis of diagnostic imaging workflow steps (Frydrychowicz 2021), others have used less granular data from radiology information systems (Zhang, Narra and Kansagra 2020) or picture archiving systems (Talati, Krishnan and Filice 2019). The combination of clinical data sources has enabled better prediction of some workflow details, for example radiology slot time prediction (Wang, Nikkhou Aski, et al. 2024), (Avey, et al. 2019) or complexity prediction of radiologic procedures (Wang, Uhlemann, et al. 2024), but these approaches are not easily scalable because of the difficult data access.

Another challenge lies in the complexity and the interdependencies of clinical processes. The descriptive analyses and workflow step predictions presented in the above-mentioned references are all based on statistical averages, but do not consider the dynamics and interdependencies of each individual workflow.

For the purpose of DCPS transition analysis, a dynamic simulation of the processes and resources is required. This can be achieved by a DES approach, in combination with a process and resource model that incorporates knowledge of workflow step durations found by statistical analysis of real data as described above. The process model itself can be designed with modelling frameworks such as the Business Process Modelling and Notation (BPMN) framework, although care must be taken to adapt the modelling language to the specific characteristics of clinical processes (Pufahl, et al. 2022).

DES evaluates the propagation of one or several "tokens" through a pre-defined process, while using random distributions to mimic decision points and delays in the model. In the clinical process example, a token could represent a patient undergoing a journey through a clinical procedure, or a piece of information being passed through different procedure steps. DES is an established modelling method in the healthcare domain and its usage has been observed to increase particularly in health and system operations during the past decade (X. Zhang 2018). This includes applications in the operational planning of interventional radiology (Tellis, et al. 2021).

5.7.4 Example Implementation of DES for CPS Transition Analysis

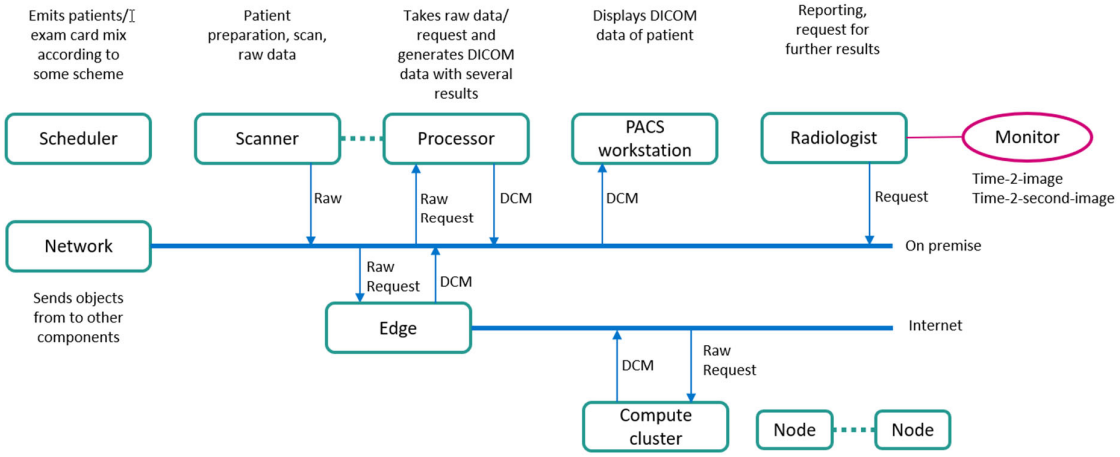

Figure 14: Components and data flow of the simulation pipeline for spectral CT reconstruction. The simulation covers two reconstructions paths: One using on-remise resources and the other using cloud-based resources.

The simulation study for spectral CT reconstruction serves as an example of an application of DES for guiding the transition to distributed systems in the clinical domain (see also D33/D3.5). The simulation has been specifically designed for a "sweet spot analysis", i.e. for determining how to distribute demanding computational tasks for medical image reconstruction across the three tiers in such a way that the requirements for report turn-around times (RTAT) for each patient can be met.

Figure 14 provides an overview of the simulation pipeline used in this study. Patients are scheduled for a scan based on the simulation profile. The raw data is then sent down one of two paths, either using on-premises resources or cloud-based resources. In the on-premises option, the data generated by the scanner is sent to a processing unit hosted by the hospital's IT department. The resulting DICOM image is transmitted to a PACS, so that they can be read and reported by a radiologist. In the cloud option, the raw data is sent via an edge device to a compute cluster where the reconstruction is performed. All aspects of the simulation are configurable, including computing resource availability (with dynamic scaling and load balancing of cloud

resources), network bandwidth, and patient profiles. The simulation also supports trauma patients requiring urgent imaging. The simulation records timestamps for each patient/exam to monitor and analyse resource utilization and network performance.

The results of such a simulation for an exemplary 8-hour hospital shift are shown in Figure 15: The upper plot represents a situation where image reconstruction tasks are performed only on insufficient on-premise computational resources. The data points off the blue diagonal line indicate cases that do not meet the RTAT requirements. Since several emergency cases needed to be prioritized, many other imaging reports had to be delayed. The lower plot shows how the same patient cases under the assumption that some less-critical reconstruction tasks could be off-loaded to a cloud service, while the critical emergency cases were still reconstructed with high priority on on-premise resources. Here, the balance of compute resources led to a situation where all reports could be delivered within the prescribed time window.

This study highlights the effectiveness of DES for analysing and designing distributed CPS in complex and safety-critical clinical environments. DES facilitates the modelling of site-specific characteristics, implemented workflows, and technical boundary conditions—such as available bandwidth and required computing resources. This capability enables the evaluation of various deployment scenarios on a use-case basis and supports the selection of optimal deployment strategies.

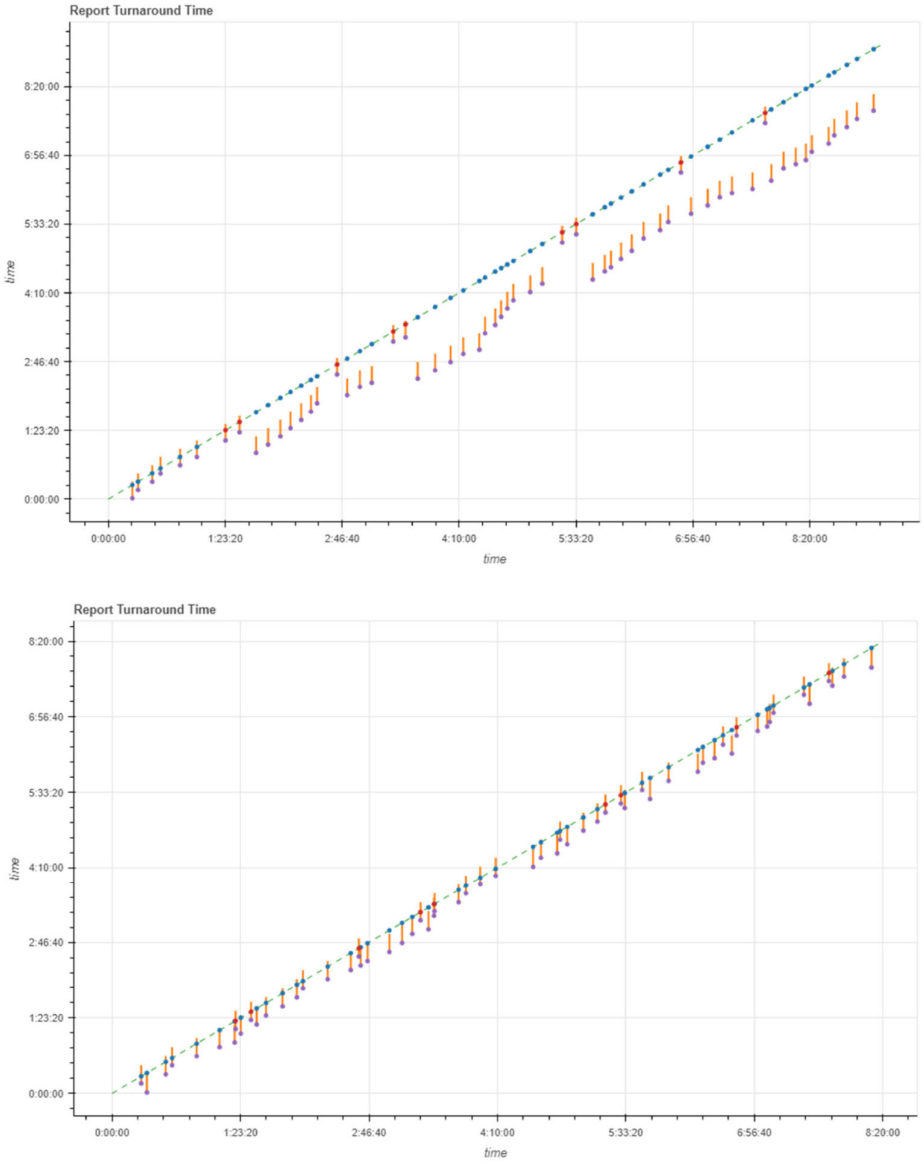

Figure 15: Simulation results of reporting times of spectral CT examinations (see also D33/D3.5). Start of the image acquisition and availability of the data in the PACS are depicted by dots. To meet the RTTA constraints, the data must be available in a 20min time windows, indicated by the orange bars. Upper: On-premise processing with insufficient compute power; Lower: Combined on-premise and cloud computation with a prioritization based on clinical urgency.

5.8 Design Space Exploration with Genetic Algorithms (ITI)

One of the major challenges when designing and maintaining a complex distributed CPS is that there are often several conflicting design objectives and constraints. For example, we might require the system to have a small response time, but this would require a high-performance processing node or perhaps several parallel processing nodes, which in turn requires more cost and power consumption. Thus, there is a great need for design and analysis tools which can assist the engineer in exploring and evaluating different system configurations at design time (and possibly at run-time as well).

There are important applications where the operating environment is dynamic. The TRANSACT architecture permits the offloading of functions from the device to the edge or cloud tiers when Service Level Objectives (SLOs) thresholds are not met. For that, it employs services to monitor and measure the Service Level Indicators (SLI) and the use of an Operational Mode Manager and Coordinator to decide at run-time the proper operational mode to be executed. In such a case the optimal CPS configuration needs to change to reflect the different operating conditions.

To this end ITI has investigated and developed some novel techniques to assist in the design and performance analysis of these types of CPS. In particular, ITI has created a multi-objective, non-linear constrained optimisation tool which uses a genetic algorithm to explore the assignment of functions (or tasks) to processing nodes with the goal of achieving certain performance objectives which satisfy defined design and operating constraints. A block diagram of the optimiser architecture is shown in Figure 16.

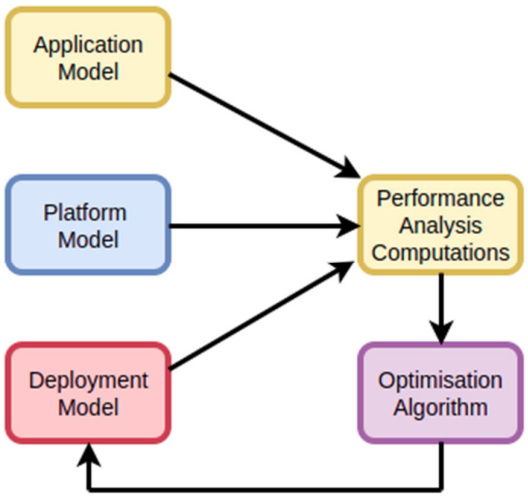

Figure 16: Architecture of the Design Space Explorer

The design space exploration architecture consists of several components:

Application Model	This is a global model of all the software that is running on the CPS. It is represented as a graph data structure. Nodes represent each function, while edges represent the calls between functions. Each node has a set of properties, for example the worst-case execution time of the task. (An example application graph is shown in Figure 18.)
Platform Model	This is a global model of all the hardware in the system. Again, we represent this as a graph network. Nodes represent the processors and devices, while the edges model the communications links and buses that transfer data between the processors and devices. Properties of the nodes might include information such as CPU type and speed, the task scheduling algorithm in use, power consumption etc. Properties of the graph's edges might include communication bandwidth and latency, for example. A typical platform model graph is shown in Figure 16.
Deployment Model	The deployment model defines the processing nodes assigned to each task. It is this artefact that is to be searched for by the optimisation algorithm.
Performance Analysis Computations	The Performance analysis computations define the targets of the optimisation. We can identify several non-functional quantities to me optimised: e.g. response times, power consumption, data throughput, financial cost, and so forth.
Optimisation Algorithm	The goal of the optimisation algorithm is to search for a suitable deployment model that optimises the performance analysis objectives. In most cases there will be two or more competing objectives: for example, data throughput vs financial cost. In addition, the optimisation algorithm must ensure that the resulting deployment model satisfies certain defined constraints (e.g. deadlines must be met).

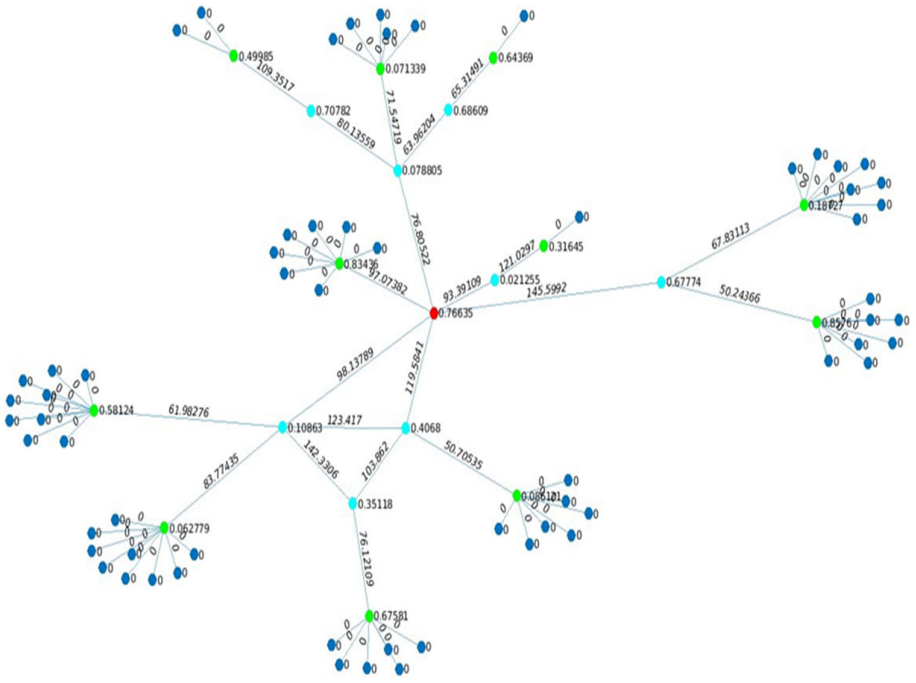

Figure 17: Typical CPS Platform Model (dark blue: edge devices, green: gateways, cyan: fog processors, red: cloud servers). Properties are assigned to each processing node and communications link. The numbers show in this graph represent the latencies of the communications links and resources available at each node.

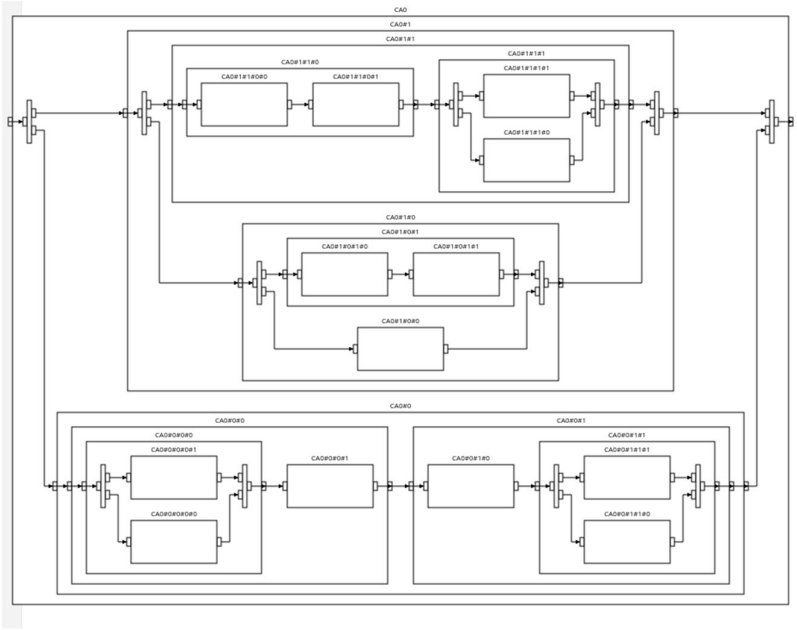

Figure 18: Typical Application Model Graph. This figure shows a set of software tasks and how these tasks are initiated by events resulting from the completion of preceding tasks.

Within such a system there arises the problem of finding the best processing nodes to locate the functions. A function might be located in the edge tier so that communication latency from devices is minimised, but this can be at the expense of time required to complete the software task because edge devices are most likely not as powerful as the CPUs or GPUs used in the cloud tier.

To this end we developed a high-level model for such a CPS, composed of two parts. Firstly, there is a model for the hardware platform of the system (platform model – Figure 17). This is effectively a graph data structure where the nodes are the processing elements, and the edges are the communications links. We then assign properties to each node and edge of the graph. For the processing nodes the properties might include items such as, for example, CPU & memory resources, clock speed, power consumption, and financial cost of use. In the case of the communications links the properties would include latency and data throughput rates.

The second modelling component, the application model (Figure 18) defines how the software functions are allocated to the processing nodes and how these software functions communicate with each other (i.e. the call graph). This software model also contains properties such as time required to execute the task, the deadline for completing the task, the resources required by the task (e.g. memory) and so forth.

Given these system properties we then developed a set of algorithms for computing non-functional performance metrics for the global system architecture. These metrics include latency between issuing a task request and receiving a response, overall power consumption, and financial cost. We also developed an improved version of the compositional performance analysis (CPA) algorithm to compute these metrics. Some possible performance objectives are shown in Table 6.

The CPA algorithm is used to compute the response times of each task. This algorithm first examines each processing node and uses the particular scheduling algorithm assigned to that node along with a specification of the triggering events for the functions running on that node to compute the expected response times of each task. These response times are propagated to subsequent tasks running on other nodes to define the triggering events for those services. A top-level iterative loop repeats this process until convergence is achieved.

Table 6: Performance objectives to be optimised

Performance Metric	Objective
Latency	Minimise
Data Throughput	Maximise
Financial Cost	Minimise
Power Consumption	Minimise

We also identified a set of constraints defined by the application requirements. Such constraints might include a specification that certain software functions are only allowed to execute on certain processing nodes. In addition, there are usually timing constraints such as certain functions must complete their tasks within a certain time frame (hard deadline).

The goal of the design space optimisation is to find a suitable allocation of software functions to processing nodes such that the performance objectives are optimised, and the constraints are satisfied. Thus, effectively we have a multi-objective, constrained optimisation problem. To solve this, we developed a genetic algorithm

optimiser to allocate the software services to processing nodes. This deployment can be captured by a matrix $A = [a_{i,j}]$, where $a_{i,j}$ is the number of instances of software functioa_i that are running on processing node j . It is the goal of the optimisation algorithm to find this matrix.

Since the problem has multiple competing objectives (e.g. financial cost *vs.* response time) the output of the optimiser is a *set* of potential solutions. These can be depicted by the so-called *Pareto Set.* An example of such a pareto set is shown in Figure 19 (Objective 1 is latency and Objective 2 is financial cost). This plot shows several potential solutions to the optimisation problem. For example, the point to the extreme left denotes a system configuration which has a cheap financial cost but has a large communications latency whereas the point to the extreme right has the smallest latency but the largest financial cost. It is subsequently left up to the system designer (or perhaps an AI system) to decide on the best solution according to requirements.

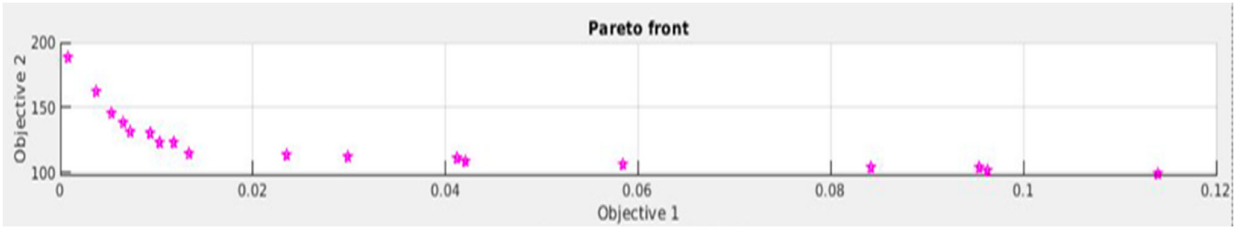

Figure 19: Pareto Front After Optimisation

6 Conclusions

This deliverable highlights key contributions of Task 2.3 within the TRANSACT project, providing insight into the transition from conventional Cyber-Physical Systems (CPS) into distributed CPS (DCPS) solutions, with a special focus on trade-off analysis and generic development principles.

The introductory section offered an overview of Work Package 2 (WP2) and its associated tasks and described the role of this document and its relationship to other deliverables. It also included a discussion of generic methods for modelling and analysing DCPS, supplemented by an overview of the distributed solution architecture developed within the scope of the TRANSACT project.

As a contribution of this deliverable, the Transition and Validation Methodology (T&V^2) was introduced - a theoretical framework for the migration of monolithic systems to distributed solutions over the edge-cloud continuum. T&V^2 is based on the T&V methodology, focusing on a holistic view that considers technical aspects, operational changes, legal obligations, and business constraints. The T&V^2 methodology emphasizes the deployment and maintenance of the solution, specifying requirements for migration activities, and considers data from these processes to drive the specification of requirements. The methodology follows a stepwise process, starting with given systems and ultimately defines technical requirements, migration plans, and requirements from operational, legal, and business perspectives.

Thereafter, driven by the individual use-cases, different aspects of the task objectives (cf. Table 3, p. 13) were explored by the TRANSACT partners. These address challenges across all stages of the transition, from system design and optimization to implementation and operation, which is reflected in a multifaceted set of contributions. To provide a comprehensive view, not only technical aspects were considered, but also legal and business challenges were taken into account.

These contributions have not only been instrumental in the achievement of the TRANSACT project's goals but have also offered valuable insights into the complexities of the transition towards DCPS as a whole. They have enriched the current understanding in the field of DCPS and laid the groundwork for further advancements.

7 References

Alhijawi, Bushra, and Arafat Awajan. 2023. "Genetic algorithms: theory, genetic operators, solutions, and applications." *Evolutionary Intelligence.* doi:10.1007/s12065-023-00822-6.

Avey, Gregory D., Daryn S. Belden, Ryan D. Zea, and John-Paul J. Yu. 2019. "Factors Predicting the Time-Length Variability of Identically Protocoled MRI exams." *J Magn Reson Imaging* 49 (7): e265-e270.

Bliudze, Simon, Sébastien Furic, Joseph Sifakis, and Antoine Viel. 2019. "Rigorous design of cyber-physical systems - Linking physicality and computation." *Software & Systems Modeling* 18: 1613–1636. doi:10.1007/s10270-017-0642-5.

n.d. *CUE.* https://cuelang.org/.

Falcone, Alberto, and Alfredo Garro. 2020. "Pitfalls and Remedies in Modeling and Simulation of Cyber Physical Systems." *2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT).* Prague: IEEE.

Fishman, George S. 2001. *Discrete-Event Simulation.* New York: Springer. doi:10.1007/978-1-4757-3552-9.

Frydrychowicz, A. 2021. "Automatic, log file-based process analysis of a clinical 1.5T MR scanner: a proof-of-concept study." *RoFo : Fortschritte auf dem Gebiete der Rontgenstrahlen und der Nuklearmedizin* 919-927.

Graja, Imen, Slim Kallel, Nawal Guermouche, Saoussen Cheikhrouhou, and Ahmed Hadj Kacem. 2020. "Acomprehensive survey on modeling of cyber-physical systems." *Concurrency Computat Pract Exper* 32: e4850. doi:10.1002/cpe.4850.

Gunn, M.L. 2017. "Improving MRI scanner utilization using modality log files." *Journal of the American college of radiology* 783-786.

IREB. n.d. *CPRE Foundation Level - Handbook.* Accessed March 19, 2024. https://www.ireb.org/de/downloads/tag:foundation-level#top.

n.d. *Kumori.* https://docs.kumori.systems/kpaas/1.0.0/index.html.

Legay, A., B. Delahaye, and S. Bensalem. 2010. *Statistical Model Checking: An Overview.* Vol. 6418, in *Runtime Verification. RV 2010. Lecture Notes in Computer Science*, edited by Howard editor="Barringer, Ylies Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon Pace, Grigore Roşu, Oleg Sokolsky and Nikolai Tillmann. Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-16612-9_11.

Liu, Zhiming, Jim Woodcock, and Huibiao Zhu, . 2013. *Theories of Programming and Formal Methods.* Heidelberg: Springer. doi:10.1007/978-3-642-39698-4.

Martin Glinz, Hans van Loenhoud, Stefan Staal, Stan Bühne. 2024. *CPRE Foundation Level - Handbook.* IREB.

Masmoudi, Chedhli, Pascale Marange, Eric Bonjour, Eric Levrat, and Alain Kerbrat. 2022. "Adopting formal methods on requirements verification and validation for cyber-physical systems: A systematic literature review." *IFAC-PapersOnLine* 55 (10): 3274-3279. doi:10.1016/j.ifacol.2022.10.131.

Mittal, Saurabh, and Andreas Tolk. 2020. "The Complexity in Application of Modeling and Simulation for Cyber Physical Systems Engineering." In *Complexity Challenges in Cyber Physical Systems: Using Modeling and Simulation (M&S) to Support Intelligence, Adaptation and Autonomy*, 3-25. John Wiley & Sons, Inc. doi:10.1002/9781119552482.ch1.

Mohamed, Mustafa Abshir, Moharram Challenger, and Geylani Kardas. 2020. "Applications of model-driven engineering in cyber-physical systems: A systematic mapping study." *Journal of Computer Languages* 59: 100972. doi:10.1016/j.cola.2020.100972.

OMG. n.d. *Unified Modeling Language.* https://www.omg.org/spec/UML/.

—. n.d. *What is SysML?* https://www.omgsysml.org/what-is-sysml.htm.

Paape, N., J. A.W.M. Van Eekelen, and M. A. Reniers. 2024. "Review of simulation software for cyber-physical production systems with intelligent distributed production control." *International Journal of Computer Integrated Manufacturing* 37 (5): 589-611. doi:10.1080/0951192X.2023.2228270.

Pappagallo, Angela, Annalisa Massini, and Enrico Tronci. 2020. "Monte Carlo Based Statistical Model Checking of Cyber-Physical Systems: A Review." *Information* 11 (588). doi:10.3390/info11120588.

Pufahl, Luise, Francesca Zerbato, Barbara Weber, and Ingo Weber. 2022. "BPMN in healthcare: Challenges and best practices." *Information Systems* 107: 102013. doi:10.1016/j.is.2022.102013.

Rakow, Astrid, Kröger, Janis. 2021. "Roles and Responsibilities for a Predictable Update Process - A Postion Paper." *Verification and Evaluation of Computer and Communication Systems: 15th International Conference, VECoS 2021.* Virtual Event: Springer. 17-26.

Sieber, P O, C M Macal, J Garnett, D Buxton, and M Pidd. 2010. "Discrete-event simulation is dead, long live agent-based simulation!" *Journal of Simulation* 4: 204-210.

Talati, Ish A., Pranay Krishnan, and Ross W. Filice. 2019. "Developing Deeper Radiology Exam Insight to Optimize MRI Workflow and Patient Experience." *J Digit Imaging* 32: 865-869.

Tampouratzis, Nikolaos, Panagiotis Mousouliotis, and Ioannis Papaefstathiou. 2023. "A Novel Integrated Simulation Framework for Cyber-Physical Systems Modelling." *IEEE Transactions on Parallel and Distributed Systems* 34 (10): 2684.

Tellis, Ranjith, Olga Starobinets, Michael Prokle, Usha Nandini Raghavan, Christopher Hall, Tammana Chugh, Ekin Koker, Siva Chaitanya Chaduvula, Christopher Wald, and Sebastian Flacke. 2021. "Identifying Areas for Operational Improvement and Growth in IR Workflow Using Workflow Modeling, Simulation, and Optimization Techniques." *Journal of Digital Imaging* 34: 75-84.

Tolk, Andreas, Fernando Barros, Andrea D'Ambrogio, Akshay Rajhans, Pieter J Mosterman, Sachin S Shetty, Mamadou K Traoré, Hans Vangheluwe, and Levent Yilmaz. 2018. "Hybrid simulation for cyber physical systems: a panel on where are we going regarding complexity, intelligence, and adaptability of CPS using simulation." *MSCIAAS '18: Proceedings of the Symposium on Modeling and Simulation of Complexity in Intelligent, Adaptive and Autonomous Systems.* Baltimore: Society for Computer Simulation International. 1-19.

Wang, Xinyu, Falk Uhlemann, Jörn Borgert, Siva Chaitanya Chaduvula, Ranjith Tellis, Alex Frydrychowicz, Jörg Barkhausen, and Thomas Amthor. 2024. "Analysis and predictability of technologists' perception of MR exam complexity." *Radiography* 30: 151-158. doi:10.1016/j.radi.2023.10.015.

Wang, Xinyu, Sahar Nikkhou Aski, Falk Uhlemann, VIkas Gupta, and Thomas Amthor. 2024. "Predicting slot lengths of MRI exams to decrease observed discrepancies between planning and execution." *Current Problems in Diagnostic Radiology.* doi:10.1067/j.cpradiol.2024.01.013.

Zahid, Farzana, Awais Tanveer, Matthiew M.Y. Kuo, and Roopak Sinha. 2022. "A systematic mapping of semi-formal and formalmethods in requirements engineering of industrial Cyber-Physical systems." *Journal of Intelligent Manufacturing* 33: 1603–1638. doi:10.1007/s10845-021-01753-8.

Zhang, Lichen. 2018. "Modeling Methods for Cloud Based Cyber Physical Systems." *IEEE SmartWorld; Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable.* Guangzhou, China: IEEE.

Zhang, Ray, Vamsi R. Narra, and Akash P. Kansagra. 2020. "Large-Scale Assessment of Scan-Time Variability and Multiple-Procedure Efficiency for Cross-Sectional Neuroradiological Exams in Clinical Practice." *J Digit Imaging* 33: 143-150.

Zhang, Xiange. 2018. "Application of discrete event simulation in health care: a systematic review." *BMC Health Services Research* 18: 687. doi:10.1186/s12913-018-3456-4.