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

D7 (D2.1)

Reference architectures for distributed safety-critical distributed cyber-physical systems v1

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.	T2.1
Deliverable No.	D7
Deliverable No. in WP	D2.1
Deliverable Title	Reference architectures for distributed safety-critical distributed cyber-physical systems v1
Nature	Report
Dissemination Level	Public
Document Version	v1.0
Date	31/05/2022
Contact	Jordi Arjona
Organization	ITI
Phone	+34 963877069
E-Mail	jarjona@iti.es

Authors Table

Name	Company	E-Mail
Jordi Arjona	ITI	jarjona@iti.es
Javier Fernández-Bravo Peñuela	ITI	fjfernandez@iti.es
Miguel García Gordillo	ITI	miguelgarcia@iti.es
Joan Valls	ITI	jvalls@iti.es
Claus-Henning Friederichs	AVL	Claus-Henning.Friederichs@avl.com
Wolfram Ratzke	AVL	Wolfram.Ratzke@avl.com
Javier Ferrer	KUM	javier@kumori.systems
Thomas Strathmann	DLR	Thomas.Strathmann@dlr.de
Sebastian Vander Maelen	DLR	Sebastian.vander.maelen@dlr.de
Elisabeth Salomon	TUG	elisabeth.salomon@tugraz.at
Christian Prehofer	DENSO Automotive	c.prehofer@eu.denso.com
Abel Gómez	UOC	agomezlla@uoc.edu
Iván David Alfonso	UOC	ialfonsod@uoc.edu
Andreas Morgenstern	IESE	andreas.morgenstern@iese.fraunhofer.de
Ralph Weissnegger	CISC	r.weissnegger@cisc.at
Juhani Latvakoski	VTT	Juhani.Latvakoski@vtt.fi
Roel Hermans	PMS	roel.hermans@philips.com
Marco Jahn	ECL	marco.jahn@eclipse.foundation.org
Jeanneth Nodland	NVT	jeanneth.nodland@navtor.com

Reviewers Table

Version	Date	Reviewer
v0.5, v0.6	18/05/2022	Benny Akesson (TNO), Teun Hendriks (TNO)
v0.5, v0.6	18/05/2022	Gaurav Choudhary (DTU)
v0.5, v0.6	18/05/2022	Martijn Hendriks (TU/e)
v0.5, v0.6	18/05/2022	Jeroen Voeten (TU/e)

Change History

Version	Date	Reason for Change	Affected pages
v0.1	20/09/2021	TOC for State of the Art	All
v0.2	08/04/2022	State of the Art topics are complete	All
v0.3	29/04/2022	Homogenization and unification of bibliographical references	All
v0.4	02/05/2022	Document polished and added a list of open-source components	All
v0.5	06/05/2022	Document ready for review	All
v0.6	19/05/2022	Version after final review	All
v1.0	31/05/2022	Final version for submission	n/a

Table of Contents

List of Figures

List of Tables

1 Glossary

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 center 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 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 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 1: Terms and definitions

Term	Definition
AI	Artificial Intelligence
BDaaS	Big Data Analytics as a Service

CPS	Cyber-Physical System
IoT	Internet of Things
MAPE	Monitor Analyze Plan Execute
MDP	Markov Decision Process
ML	Machine Learning
PIM	Platform-Independent Metamodel
QoS	Quality of Service
RA	Reference Architecture
SLA	Service Level Agreement
SLI	Service Level Indicator
SLO	Service Level Objective
SoA	State of the Art

Table 2: Abbreviations and definitions

2 Introduction

The goal of TRANSACT's Task 2.1 is to design an initial reference architecture for distributed safety-critical cyber-physical systems, application programming interfaces, and developing a basic implementation. An early version of the reference architecture is provided and described in Chapter3. In order to advance this architecture, the first step is to revisit the State of the Art (SoA), enabling the identification of architectural patterns, with a focus on safety-critical solutions, real-time systems, failure detection and handling strategies, and microservice-based architectures intended for leveraging the device-edge-cloud compute continuum.

The analysis of the state of the art and current architectures is conducted in Chapter 4. In order to collaboratively choose the topics to be studied, a survey was conducted, where the participating partners could point out those topics that they judged the most relevant to TRANSACT's field and its architectural scope. These were later analysed and unified, turning out to the identification of nine topics of potential interest. Afterwards, the partners validated the topics having been selected and a mapping from topics to partners was made, indicating the partners responsible for preparing a state of SoA analysis for each topic, directed to its applicability to TRANSACT and the advance in its architectural design and specification. When possible, the study of each topic, carried out from an academic research point of view, is complemented by a subsection focused on industrial practices, contributed by industrial partners. The following topics were selected to be studied:

1. Secure communications in the compute continuum
2. Self-monitoring & Awareness
3. SLO enforcement in the compute continuum
4. Operational mode management
5. Software updates in safety-critical systems
6. Architectures for distributed safety-critical systems
7. Microservice-based architectures and configurations
8. Big Data infrastructure and machine learning components
9. Eclipse Open-Source IoT and Edge technologies

These topics are expected to provide the knowledge needed for designing the tiers and components depicted in the reference architecture, along with the system level requirements extracted from T1.2 and the requirements and concepts defined by T3.1 and T3.2. All topics are placed around the safety and security restrictions of TRANSACT, having into account the three-tiered compute continuum of its architecture.

Topics 1 and 2 are dedicated to low-level aspects of the architecture, such as communication channels and protocols and the monitoring of the components in execution, which also enables the system to be aware of surrounding circumstances. These support Topic 3, that needs of such monitoring and measurement of network latencies to enforce a proper quality of service. The objectives of Topic 3 will often be reached through switching the operational mode, this is, changing the way the system acts according to its runtime circumstances, including offloading functions across tiers, which is studied in Topic 4. A matter also related is studied in Topic 5, regarding how software updates are performed in systems of these characteristics, spanning multiple layers and usually involving a high volume of end-devices. Topics 6 and 7 focus, from a higher-level perspective, on architectures that are suitable for TRANSACT's needs and purposes, in the context of highly distributed systems that hold safety-critical requirements and restrictions. TRANSACT's architecture shall also need to support the deployment of high-level applications. Those applications usually

need of Big Data analytics and machine learning functionalities, that are addressed in Topic 8 in order to provide a common knowledge framework to all those applications which might require them. Last but not least, Topic 9 proposes some open-source technologies that, under the scope and licensing model of the Eclipse Foundation, are particularly relevant and beneficial for TRANSACT, due to their application in the edge and device areas.

In addition to these research topics, some partners have proposed a collection of open-source technologies and components, that are planned to be actively used for development or are particularly relevant to TRANSACT's scope. These are briefly presented in Chapter 5.

Finally, a summary that draws some relevant conclusions about the work carried out in this document, and how further tasks and deliverables will benefit and be influenced by it, is provided in Chapter 6.

2.1 Relationship to other TRANSACT documents

This document relates to the following TRANSACT deliverables:

- D26 (D2.1) Reference architectures for SCDCPS v2
- D27 (D2.2) System Test Platforms for Simulation based-design and assessment of SCDCPS
- D8 (D3.1) Selection of concepts for end-to-end safety and performance for distributed CPS solutions
- D9 (D3.2) Selection of concepts for end-to-end security and privacy for distributed CPS solutions
- D13 (D4.1) AI services integrated per Use Case v1
- D14 (D4.2) Strategies for continuous updating and independent releasing v1

The results obtained by this deliverable document D2.1 will be used to create the demonstrators in WP5, and it will also affect the solutions devised in WP3 and WP4. It is expected to align with the use cases and end user requirements specified by WP1 and should also be used as an input in the requirements definition of T3.1 and T3.2. The output attained by this T2.1, from a task-centered point of view, will be used in T2.2 and T2.3, providing an architecture for its analysis, and T4.1 and T4.2, providing an architecture for integration and module updating.

3 Initial reference architecture

The TRANSACT project has as aim to develop a universally applicable distributed solution architecture concept, framework and a transition methodology for the transformation of safety-critical CPS into distributed safety-critical CPS solutions. In the project proposal phase, an early reference architecture was proposed. In this section we present the current status as an initial three-tier reference architecture. This initial reference architecture will be further developed in the project.

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

A key element of the TRANSACT concept is that both these off-device tiers (edge and cloud) are multi-application, multi-device, possibly multi-tenant, scalable and interoperable, and not in the least, safe and secure. Applications can be flexibly deployed over this distributed architecture. Novel services and applications can share edge and cloud infrastructures, re-use domain services, and in general be faster developed, tested, and released independently from the safety-critical device itself. This newly distributed architecture concept will enable radically new business models by transforming system manufacturers into solution providers.

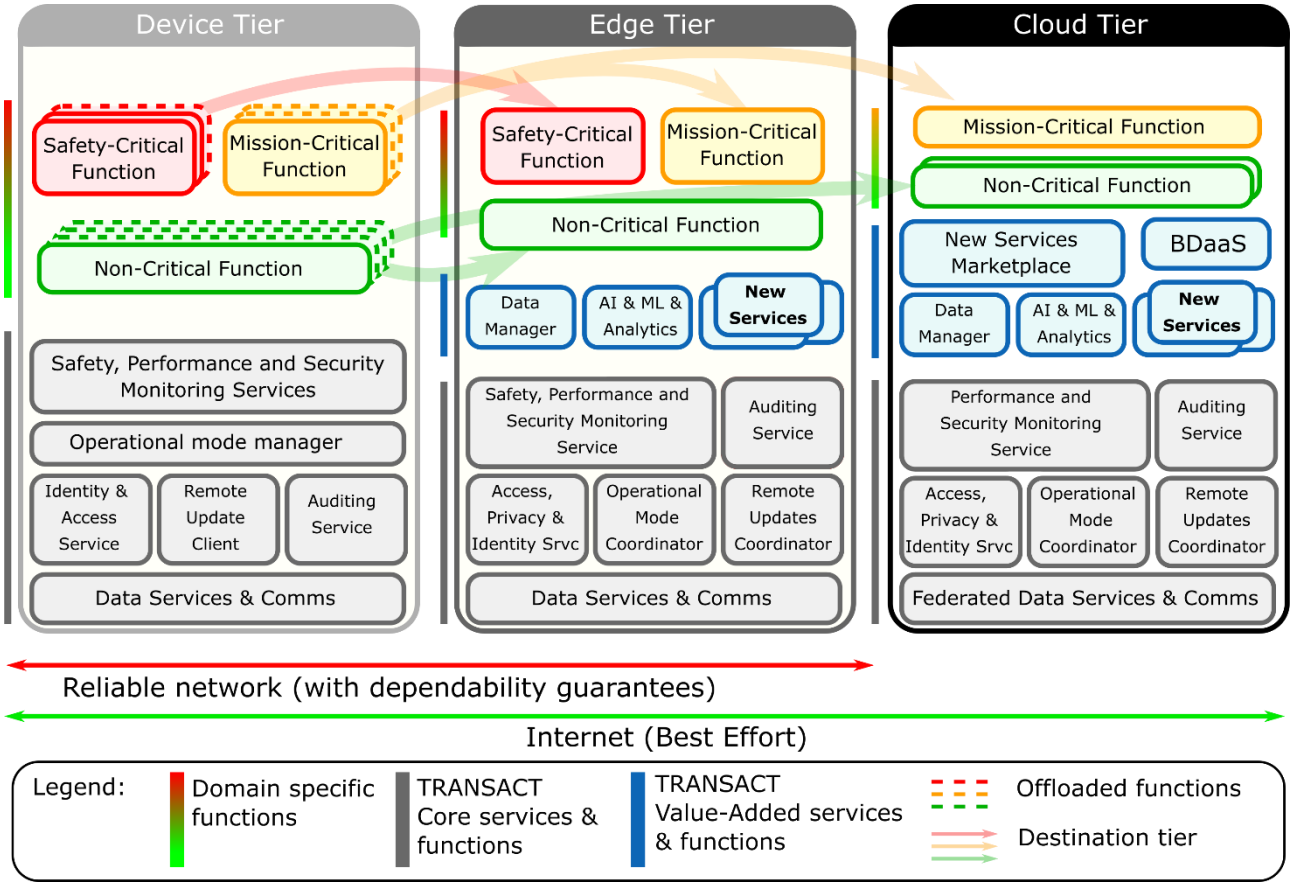


Figure 1: TRANSACT initial reference architecture

Each tier in the architecture provides a specific quality of service level especially with respect to performance aspects such as response times and data transfer guarantees, ranging from best effort to reliable and time-

deterministic data transfers. Safety-critical and mission-critical functions often have timing-related constraints, for example, safety-critical applications often pose hard real-time constraints (which lead to severe failures when missed), whereas the mission-critical functions may have soft real-time constraints (which may degrade the system's quality of service when missed, but do not necessarily lead to failures).

The TRANSACT reference architecture defines firstly the safety- and mission-critical functions, and non-critical functions. These domain specific functions or components, that are depicted in red, yellow and green depending on their criticality, may be offloaded from the device to other tiers. Furthermore, the reference architecture defines the core TRANSACT components, available to every use case, which are depicted in grey. Finally, blue components refer to potential Value-Added functions that may be included depending on the use case.

The safety- and mission-critical functions are key in the safety-critical CPS. TRANSACT aims at improving the existing CPSs by, first, stripping the device of the functions that are not safety or mission-critical and can be executed remotely; and second, by offloading certain safety-critical functions to the edge tier. The functions to be offloaded are identified at the design time and deployed in the required tiers. As a result, the device would only keep the basic and 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 the device), improved efficiency of the offloaded functions due to usage of better hardware at the edge or cloud, improved innovation speed of the distributed CPS solution as the new or upgraded functions can be deployed easier at the edge and cloud.

However, when considering offloading functions from the device, it is critical to ensure CPS solution's end-to-end safety and security. Therefore, a number of dedicated core services are introduced to cooperatively realize that objective. The *safety, performance and security monitoring services* are responsible for monitoring, detecting, and preventing safety, security and performance failures. In addition, they measure the Service Level Indicators (SLIs), such as latency, throughput, accuracy or uptime.

The Service Level Objectives (SLOs), measured by the SLIs, are used by the *operational mode manager* (running on the device) and the *operational mode coordinator* (running at the edge/cloud tier) to decide at run-time the proper operational mode to be executed. E.g., function offloading from the device is associated with operational thresholds or ranges described in a SLO. If this SLO is compromised or out of range, the operational mode manager will ask the device to switch to an operational mode where the function is executed locally, to ensure the required performance.

Another area that the TRANSACT reference architecture addresses is updating the distributed solution. To achieve safe and predictable system updates the following core services have been identified: the *remote update client* (running on the device) and the *update coordinator* (running at the edge/cloud). Those services cooperate across tiers to perform remote automatic updates of the different device services in a secure and safe way. The updates ensure uniform software versions on the tiers and keeps the system services up-to-date with the latest functionality. In addition, the automatic updates allow rolling-out a new functionality or introduce new value-added services minimizing system downtime. Each update activity is coordinated with the operational mode coordinator service to keep the system in the safe state at any time.

The secure access to the system functionality is managed by the *access, privacy and identity services*. These services are responsible for granting/denying access to the system resources based on the policies defining who has what access (in which role) to which resources. Another core services contributing to the system security are the *auditing services*. These services collect information about accessing and using the system in order to help detecting the security policy violations when system is accessed by unauthorized users or in an unauthorized way. The security aspects are also addressed by the *(federated) data services and comms* services helping in efficient and secured data handling, both, in transit and at rest.

This initial TRANSACT reference architecture also defines the value-added services that enhance the system capabilities. Those functions can be introduced at the system design or after system release (as part of the system updates). Within these services are the *data manager services* that encompass the storage, protection and access to data in the cloud and edge tiers. They ensure that data is validated and fully accessible to services and applications when needed. Also, the *Big Data as a Service (BDaaS) component included in the cloud tier* enables the infrastructure to provide other services and applications with advanced analyses on large data sets.

Further value-added services are the *AI & ML & analytics services* giving insights in the collected data by extracting valuable information that helps improving the user activities. For example, in the healthcare domain those services can enhance the health specific algorithms assisting doctors in the diseases diagnostics and supporting them in making the clinical decisions; in automotive domain those services can enhance the routes predictability or leverage the risk analysis to assist in better automatic and human driver decisions. Next to the user task improvements, such services can help in optimizing an organization's operational performance and costs (for example by better equipment utilization in the healthcare or transportation domains).

The new value-added services can also be available via the *new services marketplace*. Such a marketplace service opens possibilities to provide new solutions (applications, services, algorithms, AI models, etc.) not only by the system builder but also by the 3rd party vendors.

In the TRANSACT project, this initial universally applicable distributed solution architecture concept will be further elaborated, and augmented with framework and a transition methodology for the transformation of safety-critical CPS into distributed safety-critical CPS solutions. The final reference architecture solution will be able to hosts a diverse set of mixed-criticality applications, which can be distributed over multiple resources, e.g., CPS devices, high-end edge computing resources hosting multiple apps of mixed-criticalities, or in the cloud. It also will support a variety of interesting trade-offs by allowing the allocation of mixed-criticality functions over the computing continuum of device-edge-cloud.

4 State of the Art

In this section, the state of the art is explored for a number of identified topics. As explained in Chapter 2, these topics are expected to provide the knowledge needed for designing the tiers and components depicted in the reference architecture, and so form input to the next stages of WP2.

4.1 Secure communications in the compute continuum

This section covers generic and domain specific secure communication patterns. A general introduction in security patterns is given in section 4.1.1. Subsequent sections cover specific patterns in different domains like the automotive domain. The industrial practises covers application of secure communication patterns in the automotive and medical domain.

4.1.1 Security Patterns

In general, the security patterns are a broad set of solutions that address specific security problems by controlling (stopping or mitigating) a set of specific threats through some security mechanism defined in a given context [1]. However, unlike, the design patterns, that are primarily used during the system design and development, the security patterns cover also deployment aspects and security-processes enhancing system development, deployment, and operation. In general, applying security patterns should help to proactively adopt the security measures already during system design and development and this way (by design) ensuring more secure system deployment and its operation.

By moving safety-critical CPS architectures away from the centralized, on-device solution toward distributed, cloud-based architectures , the security surface of the new solution increases significantly by making it more vulnerable for security attacks. In addition, since those systems perform safety operations, the security-holes may be subject to attacks, leading to hazardous situations or even user harm (e.g., in healthcare and automotive domains). Moreover, the data collected by the distributed systems (like driver information or health patient information) is highly sensitive and require special care not to be exposed due to security attacks or software vulnerabilities.

Due to the inherent complexity of CPS and safety nature, using the security pattern during its development or migration should strengthen not only the core security principles around user/patient data confidentiality, integrity, and availability, but also improve handling of user identity, services/data access control (by whom what services/data can be used), and accountability for the performed actions on the system. Figure 2 presents various security pattern types mapped to the proposed TRANSACT system architecture.

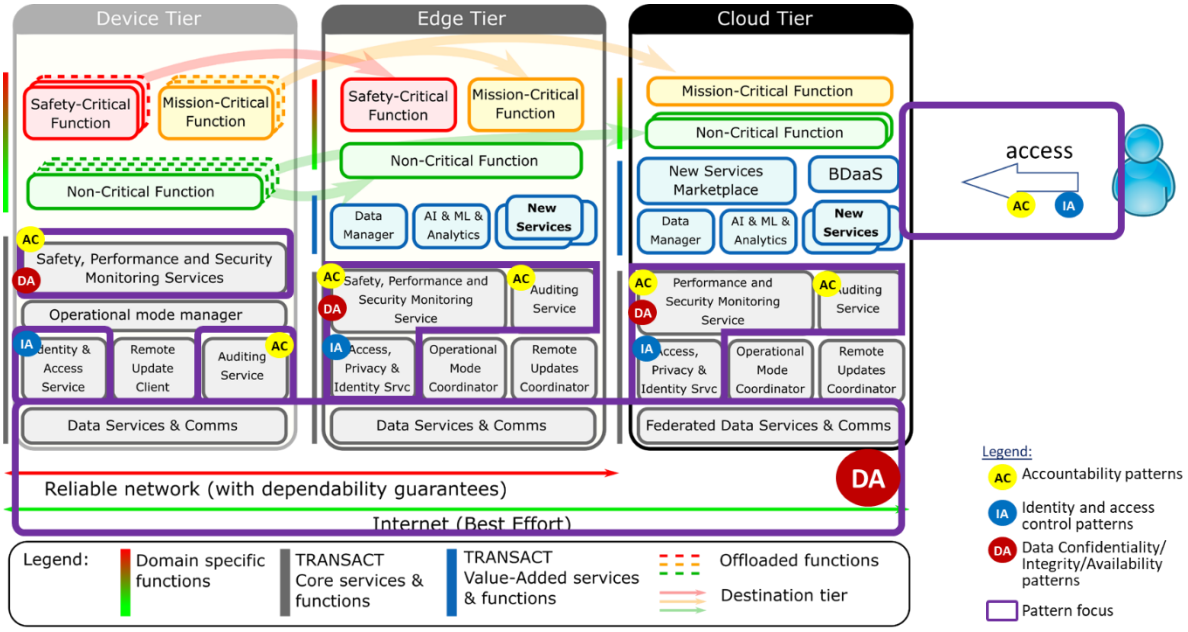


Figure 2: Security patterns types mapped to the preliminary TRANSACT reference architecture

The security patterns that can be considered for guarding the communication security of the migrated CPS system are grouped according to the following types: 1) data confidentiality/integrity/availability, 2) identity and access control, 3) accountability, and 4) security processes and framework patterns. The specific patterns are enumerated below with short description.

Remark: in the pattern description, the term '*subject*' is used to represent both users and devices.

In the patterns described below, the patterns originated either from the SECREDAS project report [2], indicated by the Design Pattern (DP) coding, or the book Security Patterns in Practise [1], no additional indication.

4.1.1.1 Data confidentiality/integrity/availability patterns

The three terms cover different aspect. Data confidentiality covers a state of trust where information is accessible only to those authorized to have access. Data integrity covers a state of data being complete and untampered. Data availability covers a state of data being accessible (in a timely manner). The latter two terms are applicable both for data at rest or in-transition.

- Abstract Virtual Private Network pattern: describes how to set up a secure channel between two endpoints using cryptographic tunnelling, with authentication at each endpoint. An endpoint is an interface exposed by a communicating unit (user site or network).
- Abstract intrusion detection system (IDS) pattern: allows monitoring of all traffic as it passes through a network, and its analysis, to detect possible attacks and trigger an appropriate response. More specialized patterns derived from it are: the *Signature-Based IDS* pattern and the *Behavior-Based IDS pattern*.
- Symmetric/asymmetric encryption patterns: protects message confidentiality by making a message unreadable to those that do not have access to the key.
- Secure messaging patterns: by default, the SSL/TLS protocols enable the secure services communication for any API style (REST, SOAP, etc.). To improve the security of the E2E communication, the WS-Security mechanisms can be employed. WS-Security describes how to embed existing security mechanisms, such as XML Encryption, XML Digital Signature and security

tokens, into SOAP messages to provide message confidentiality, integrity, authentication, and non-repudiation.
- DP06: Healthcare Data Exchange pattern: provides reliable, secure access to accurate healthcare information about the driver by considering the privacy aspects to be used for fitness-to-drive check. Despite focusing on automotive industry, this pattern is applicable in other domains as well.
- DP17: Secure key storage pattern.
- DP18: Separation of networks pattern.
- DP20: Secure remote communication pattern.
- DP27: Anonymization and Data Protection pattern.

4.1.1.2 Identity and access control patterns

Identity control ensures that the entity trying to access a resource can establish he/she is what he/she claim to be. Access control establishes what operations the established entity is allowed to perform with the resource.

- Identity Provider pattern: allows the centralization of the administration of subjects' identity information for a security domain.
- Identity Federation pattern: allows the formation of an identity federation based on several service providers.
- Authenticator pattern: allows verification that the subject intending to access the system is who or what it claims to be.
- Authorization pattern: describes who is authorized to access specific resources in a system, in an environment in which we have resources whose access needs to be controlled.
- Various detailed access control patterns:
  - Role-Based Access Control pattern: describes how to assign rights based on the functions or tasks of users in an environment in which control of access to computing resources is required .
  - Policy-Based Access Control pattern: describes how to decide whether a subject is authorized to access an object according to policies defined in a central policy repository.
  - DP30: Data Usage Control pattern: enables sensitive data protection according to the configured policies based on the corresponding context information.
- DP05: Digital Identity Management and Smart Profiling pattern
- DP03: Bastion Host pattern.
- DP04: Cryptographic Erasure pattern: Revoke cryptographic keys that decrypt distributed data from the appropriate entities.
- DP08: Decentralized Key Distribution pattern: Key distribution is important for establishing Public Key Infrastructures (PKI) and for authentication and authorization processes.
- DP10: Authentication and Authorization module pattern.
- DP29: Accountable Privacy-Preserving Authentication pattern: enables authentication which is privacy-preserving for network entities as long as they are well-behaving.
- DP32: Derived Identity pattern: enables to issue a derived identity leveraging multiple Service Providers (SP)

4.1.1.3 Accountability patterns

Accountability patterns ensure that information regarding actions on important resources is established in order to ensure proof that entitled entities performed these action or evidence to detect illegal actions. For example, the identity and access on electronic personal health information needs to be established.

- Security Logger and Auditor pattern: describes how to keep track of subjects' actions in order to determine who did what and when.
- Digital Signature with Hashing pattern: allows to prove that a message was originated from a source subject. It also provides message integrity, by indicating whether a message was altered during transmission.
- DP34: Privacy Preserving Path Detection patterns.

4.1.1.4 Security processes and framework patterns

To ensure the above secure (communication) patterns are in-place for a product, there needs to be both design for security and testing for security. The security processes ensure that these activities take place. Below are patterns typically used in the security testing.

- DP02: Automated Threat Detection and Vulnerability Management pattern: enables conduct vulnerability management and service enumeration in order to create automated tests.
- DP12: End-2-End Security pattern: This design pattern is a framework that controls access to sensitive functions, owing to the authentication scheme provided by a secure element .
- DP22: Security Test Framework pattern: describes an architecture for a security testing framework focused on testing and verifying the security of data communication
- DP31: Security Testing Process.

4.1.2 Model-based extraction and analysis of network security policies

Firewalls, designed to filter the traffic of a network with respect to a given number of access-control rules, are key elements in the enforcement of network security policies.

Although there exist approaches to derive firewall configurations from high level network policy specifications [3] [4], these configuration files are still mostly manually written, using low-level and, often, vendor-specific rule filtering languages. Moreover, the network topology, that may include several firewalls (potentially from different vendors), may impose the necessity of splitting the enforcement of the global security policy among several elements. Due to the complexity of the process, it is likely that organizations end up with differences between the implemented policy and the desired one. Moreover, security policies must often be updated to respond to new security requirements, which requires evolving the access-control rules included in the firewall configuration files.

Therefore, there is a clear need of an easy way to represent and understand the security policy actually enforced by a deployed network system. At the moment, this still requires a manual approach relying on, again, low-level and vendor-specific expertise. Given a network system consisting of several firewalls configured with hundreds of rules, the feasibility of this manual approach could be seriously questioned. While the security research community has provided a plethora of works dealing with the reasoning on security policies—succeeding at providing a good analysis and verification of the low-level firewall rules—they typically fail at obtaining a comprehensive solution. This is because these approaches do not typically provide a high-level, easy to understand and manage representation nor take, generally, networks composed by several heterogeneous firewalls into account. Moreover, the extraction step is often neglected and the solution presented over synthetic rules without providing the means to bridge the gap between them and the real configurations.

In order to effectively manage network security policies in an organization as a whole, new integrated solutions must have the following features: First, they must provide independence from the concrete underlying technology, so that the focus can be put into the security problem and not in implementation mechanisms like chains, routing tables, etc; second, they must provide a higher-level representation so that

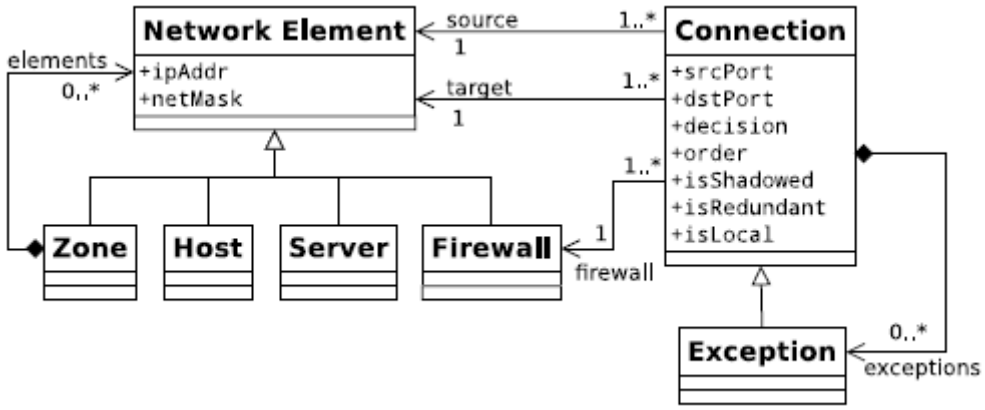


Figure 3. Network Access-Control Metamodel proposed by Martínez et al. (excerpt)

the policy becomes easier to understand, analyse and manipulate; and third, solutions must take into account the contribution of each policy enforcing element (firewall) to the global policy, as the partial picture given by isolated firewalls do not provide enough information to understand the network policy.

In [5], Martínez et al. propose a model-driven approach aimed at covering this gap. In their approach, the authors extract and abstract the information of each firewall configuration file to models conforming to a platform-independent metamodel (PIM) specially tailored to represent network-access control information in an efficient and concise way (see Figure 3). Then, after performing structural verification of the information in the individual models, they combine these models to obtain a centralised view of the security policy of the whole network. Finally, this global network access-control model is analysed and further processed to derive useful information. The application scenarios are:

- *Metrics and Advanced Queries*. Having the access-control information of a network represented as a model enables reuse of a plethora of well-known, off-the-shelf MDE tools. Editor generators, model transformation engines, etc., become automatically available. An immediate application would be the use of the well-known OCL query language to calculate interesting metrics on the model and perform some advanced queries on the rules represented in it to obtain a deeper knowledge on the security policies.
- *Forward Engineering*. A PIM model extracts and abstracts information from working networks. Nevertheless, the PIM is still rich enough to be able to be used as starting point for the regeneration of the configuration files if necessary (e.g. after modifications on the PIM to update the security policy of the network according to new requirements). In that sense, some existing forward engineering efforts [3] [4] that produce firewall configurations from high level representations can be reused.
- *Visualization of the Topology*. A PIM can also be used to derive the topology of the network, i.e., the arrangement of components and how the information flow through them. For this purpose, a model-driven visualization tool like Portolan[1] can be used by defining a model transformation from the PIM towards the Portolan Cartography model.

[1] https://github.com/atlanmod/portolan

- *Network PIM to XACML*. The above network access-control PIM is a specific representation tailored to the network domain, but a translation from a network-focused PIM towards a more generic access-control representation allows reusing tools and results that work on a more general access-control model, such a XACML. XACML [6] is an OASIS standard for the specification of access-control policies in XML and is ABAC [7] and RBAC [8] (two of the most successful access-control models) capable. Its flexibility to represent multiple policies for the same system and the fact of counting with a reference implementation, along with the increasing adoption by industry and academy, makes XACML a good choice for a generic access-control representation. Indeed, some works in the model-driven community already chose XACML as a target language as in [9] and [10].

Model-based approaches for the extraction and analysis of network security policies are not only applicable to stateless firewall configurations (i.e., security policies of first-generation firewalls, based on packet filtering at the lower layers of the OSI reference model). In [11], Garcia-Alfaro et al. apply a similar approach to analyse the deployed configurations of second and third generation firewalls peeking into the transport and upper layers. Specifically, they extend the Martinez et al. approach [5] to assist and guide in the correction of the complex misconfiguration cases, in which misconfigurations are driven by the omission of explicit rules in a policy. Their approach based on the specification of general automata that describe the different states that traffic packages can take throughout the filtering process. As in [5], Garcia-Alfaro et al.'s proposal is based on model-driven engineering, with the aim of getting rid of the low-level details of the concrete solution, and provide a solution for any other stateful filtering system with minimum effort. Their prototype provides an extension of MIRAGE [12], a firewall audit tool for the automatic detection and correction of stateless firewall configuration anomalies.

Other strategies for filtering network traffic and guaranteeing the integrity and confidentiality of information in multi-layered systems (including edge, fog, or cloud) are proposed in [13] and [14]. Prabavathy et al. [13] propose an Online Sequential Extreme Learning Machine (OSELM) algorithm to detect attacks in multi-layered IoT systems. This algorithm (deployed on the edge nodes) analyses the network traffic traveling from the device layer (sensors and actuators) to the edge and cloud layers. The incoming packet is classified as normal, or attack based on the training data. The learning algorithm can detect several types of attacks including Probe, R2L, U2R, and DoS with 97% accuracy. In [14], Bica et al. propose a security framework for multi-layer (device, edge, and cloud layers) IoT systems with emphasis on the health and social care monitoring. This framework is based on a decentralized architecture composed of a central management node deployed in the cloud. Communications between architecture components are secured using authentication mechanisms, such as digital certificates, asymmetric keys, or pre-shared symmetric keys. To mitigate DoS attacks, a packet filtering module (a low-cost Denial of Service mitigation) is deployed at the Edge layer. This module can cover all the networking stack layers, focusing on the Message Queuing Telemetry Transport (MQTT) and CoAP application layer protocols.

4.1.3 Reliable Communication in Low-Power Wireless Networks

Communication is the backbone for systems based on a device-edge-cloud architecture. Especially for distributed safety-critical systems it is therefore highly important to meet given communication requirements, i.e., to ensure high reliability and low latency. Reliable communication is also crucial in order to guarantee secure communication for safety-critical systems and particularly challenging in dynamic, low-power wireless networks. Environmental properties like interference, multipath fading, and temperature, can drastically impact the reliability of wireless communication. Depending on the used technology and the system requirements, different solutions can be applied to ensure dependability.

Bluetooth Low Energy (BLE) is among the most ubiquitous low-power wireless technologies in the IoT and increasingly used for time-critical applications that require interaction with the cloud. Therefore, it needs methods to guarantee end-to-end dependability requirements for cloud-based BLE applications. Spörk et al. [15] deals with sustaining a given end-to-end reliability and a given end-to-end latency for the communication between a BLE end-node and a server in the cloud by monitoring the end-to-end network delays and packet loss. An analytical model of the communication between a BLE end-node and the cloud is used to dynamically adapt the BLE connection parameters (i.e., number of retransmissions, connection interval, and slave latency) to sustain end-to-end dependability requirements while minimizing energy consumption. The communication in this work is based on IPv6-over-BLE (RFC7668). Compared to other works, this one does not only investigate how to sustain end-to-end requirements in the local subnet, but to maintain the communication across the entire network path.

In IEEE 802.15.4,low-power networks, the typical channel access mode is based on a contention-based MAC protocol. The two major factors affecting the reliability of wireless protocols using contention-based channel access (e.g., the IEEE 802.15.4 and IEEE 802.11 standard), are *contention* and *channel errors.* The former occurs in the case nodes access a channel simultaneously, which can lead to message loss caused by collisions or failure in the CSMA/CA algorithm. Channel errors, on the other hand, have an impact on the transmitted data and are independent from contention. For a star network using the beacon enabled channel access mode, Di Francesco et al. [16] propose to monitor both factors and adjust connection parameters accordingly in order to guarantee reliability. They designed the ADaptive Access Parameters Tuning (ADAPT) algorithm, a solution for monitoring and adapting the communication based on the IEEE 802.15.4 standard, with the purpose of being energy efficient and satisfy a given reliability constraint.

Schedule-based MAC protocols (e.g., slotted CSMA/CA and TSCH) meet delay and reliability constraints by efficiently assigning timeslots and frequency bands to a group of nodes, which makes them suitable for real-time operations [17]. A new trend in research is synchronous transmission in low-power wireless networks [18]. Compared to common low-power wireless protocols, synchronous transmissions do not avoid collisions (e.g., using contention-based channel access or scheduling), but let multiple nodes transmit the same information at the same time; this works because of the capture effect and constructive interference. Synchronous transmissions are applicable to off-the-shelf wireless protocols, like IEEE 802.15.4, BLE, and LoRa, and protocols exploiting the concept of synchronous transmission (e.g., Glossy [19] and BlueFlood [20]) were shown to be both highly reliable and efficient. Glossy [19], for example, was demonstrated to efficiently flood a 5-hop network within less than 3 ms and enables time-synchronization with the initiator for free (i.e., nodes are implicitly synchronized as packets propagate through the network). SoA protocols exploiting synchronous transmission can be applied in various CPS and IoT applications. However, further efforts are required to make them secure and standardized.

The Transmission Control Protocol (TCP) guarantees reliable communication, but is considered as too resource heavy and therefore ill-suited for low-power and lossy networks (LLNs) with limited processing, memory and energy resources. Kumar et al. [21] investigate the applicability of TCP for IEEE 802.15.4-based LLNs. It was shown that the reasons for poor TCP performance are small link-layer frames that increase TCP header overhead, hidden terminal effects over multiple wireless hops, and poor interaction between TCP and duty cycled-link. By resolving these issues, running full-scale TCP on low-power devices was shown to perform well in LLNs. TCP-enabled LLN devices would be natively interoperable with the rest of the Internet via TCP/IP: IoT protocols that assume a TCP-based transport layer and security tools for TCP/IP networks could be used, without requiring an application-layer gateway (using TCP would allow the application-layer gateway to be replaced with a network-layer router).

4.1.4 Hardware Security in Low-Power Wireless Networks

With the increasing amount of pervasive computing elements in our daily lives, the topic of end-to-end security has gained importance. When data is exchanged between the elements, it must be protected in order to provide authentication, data confidentiality and integrity, regardless if used in automotive, industry or consumer domain. One way to achieve secure data transfer is through the use of a Secure Element (SE). This section examines how to include a SE in a wireless communication between two devices. A three-component architecture is introduced for that purpose. It allows the modification of the general purpose storage of a SE by a mobile device over a wireless communication channel. The architecture is implemented using a SE on a Raspberry Pi that serves as a Bluetooth Low Energy (BLE) peripheral for data exchange with a mobile device. The goal is to provide a proof of concept as the foundation for future development of a secure BLE commissioning system.

Over the course of the last few years, the idea behind the Internet of Things (IoT), a network of connected smart devices, has increased in popularity [22]. With the growing number of pervasive devices and increasing number of the application fields of ubiquitous computing security has become an important topic.

In the IoT, four layers of security can be identified. The first one is the perception layer. It consists of sensors, actuators and the radio element. The second layer is the network layer which is responsible for handling the wireless communication. Third, there is the middleware layer. It processes information, interacts with databases for storage and provides services for all connected nodes. Fourth, the application layer builds the top-most security layer. Its contents depend on the IoT application code. For every layer, different security threats can be identified. An overview of the layers and their threats can be seen in Figure 4. Focusing on the application layer, sniffing attacks are a major security threat. A good way to counteract these attacks is to encrypt the data before it is transmitted. Combining this encryption with the encryption of the Bluetooth connection built into its connection protocol, a two-layer security architecture is created [23].

Generally speaking, three classes of security threats relevant to end-to-end security can be named: availability threats, integrity threats and confidentiality threats. The first class of threats makes a system unavailable and incapable of performing its tasks. This could be due to interference attacks on the radio module or a sleep deprivation torture attack of a low power hardware. The second class is integrity threats. Integrity is violated when unauthorized parties can modify information. The last class are confidentiality threats. Confidentiality in a system is threatened if unauthorized parties get access to protected information. For this type of attack, it is especially important to encrypt the transmitted data [24].

Perception layer	Network layer	Middleware layer	Application layer
Unauthorised access	Sybil attack	Unauthorised access	Code injection
Tag cloning	Sinkhole attack	DoS	DoS
Eavesdropping	Sleep deprivation	Insider attack	Spear-Phishing
Spoofing	DoS		Sniffing
RF jamming	Code injection		
	MitM		

a. DoS = Denial of Service, MitM = Man-in-the-Middle

Figure 4: The authors of [23] identify four layers of security for IoT devices and examine potential threats.

Based on the different types of threats various security requirements for pervasive systems can be identified. The first requirement is device authentication. In order to implement this, it is important to authenticate both ends of a connection. Another requirement is that the pervasive platform ensures that no malware can be deployed on the device. In addition, the communication is required to be confidential while at the same time it must provide message integrity. All of the listed security requirements have the need for keys. The computation and storage of keys must be secure and tamper resistant. This can be achieved through the use of an additional secure environment [25].

In the introduced architecture, the secure element is controlled over a wireless communication channel. The technology used for this is Bluetooth, in particular Bluetooth Low Energy (BLE). BLE is a new Bluetooth protocol optimized for low power consumption.

4.1.4.1 Concept for hardware security

The embedded controller (gateway) is the main actor of the secured data access layer. It acts as interface between sensor nodes and the cloud environment. It is responsible for collecting data from smart sensor nodes on one hand and forwarding it in a tamper-proof and privacy-preserving way on the other hand. A schematic overview is described by Figure 5. Furthermore, hardware and software requirements regarding the gateway, the corresponding cloud layer and its interfaces can be found in the following.

4.1.4.1.1 Hardware requirements

- Contactless wireless capabilities (BLE):

BLE is a short-range wireless technology. It greatly benefits IoT applications due to its power saving design, the coexistence of connectionless (broadcaster and observer roles) and connection-based (peripheral and central roles) data transfer procedures, its robustness against obstacles, and compatibility with a wide range of smart devices. It can be used for the actual transaction and redemption handling between local communicating devices, without the need of an active internet connection.

- Device life-cycle management and maintainability:

The gateway will allow secure updates in the field after the manufacturing phase. A trusted unique identity makes it possible to manage the device throughout its entire lifecycle. This makes it possible to address individual devices and support secure life-cycle functionality, such as device activation, secure provisioning,

secure firmware updates, device blacklisting/disablement, secure management of device functionality which in the end facilitates the deployment and management of connected embedded devices.

4.1.4.1.2 Software requirements

- Enhanced interoperability between heterogeneous service providers:

Definition of appropriate APIs for effectively interconnecting different IoT and cloud-based systems. In this context, cloud-service-access, as well as data exchange between different backends shall be enhanced in regard to speed and integration costs.

- Secure cloud interface:

The cloud part works as a certification authority and data management system. The idea behind this layer is to provide means for sharing confidential datasets in a cryptographically secure way. Different application layers may communicate to this layer via a RESTful interface.

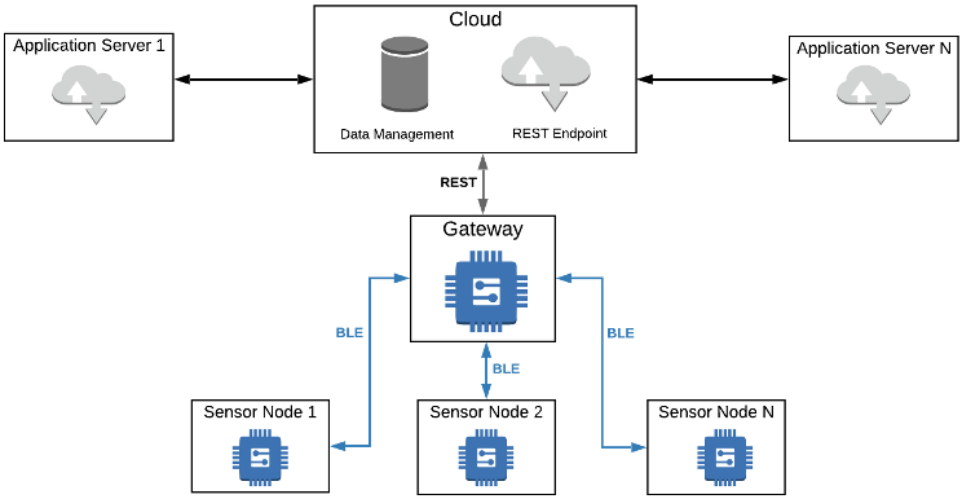


Figure 5 Schematic overview of gateway interfaces

4.1.5 Light-Weight Components Security Integration

4.1.5.1 Context

Modern electronic vehicle systems have to be secured against unauthorized access and manipulation via diagnostic ports and remote interfaces. Electronic control units can be switched into diagnostic mode to read protected error codes (DTCs) or other data (DIDs), or to set up configuration and calibration parameters. Malicious manipulation influences the behaviour of the component in operation mode, and therefore switching over to diagnostic modes has to be secured in a way that only authorized parties are able to read protected data or manipulate settings of the component.

Up to now, security measurements are used that can be operated by a standalone workshop tester without connection to the Internet. The UDS security access 0x27 provides a switch-over protection using a challenge-response-proceeding that can be handled by the software of the workshop tester. This leads to the fact that the secret to respond to the challenge is implemented in the software of the workshop tester and can be analyzed and distributed to others.

To improve security, the challenge-response-based access controls will be replaced by certificate-based solutions where centrally generated access tokens including certificates are used, which have to be checked

by the appropriate control unit. This typically requires a secure time base and asymmetric cryptography, which may overextend smaller electronic units that do not need these capabilities for their operation.

Future network architectures in the automotive environment will be based on HPCs (High Performance Controllers), which are connected to light-weight peripherical systems in a network performing safety-critical operations on electric-mechanical components.

4.1.5.2 Problem

Future network architectures in the automotive environment tend to concentrate the functionality in few powerful computation units. To operate the peripheric systems, a digital network is used today and in the future to keep the amount of electric wiring low. The intended concentration of calculation power to HPCs by keeping digital network communication leads to smaller electronic units in the peripheric systems. These units come as 'zone ECUs' to drive special functionalities in a specific functional corner of the vehicle, or as 'smart sensors' and 'smart actors' with integrated digital communication system. These lightweight units may provide a diagnostic mode or, in case of safety-relevant functionality, a tamper protection of its operational communication to adapt the unit to the system needs. To keep production costs low, the security mechanisms should be implemented in a way that requires limited technical capabilities inside the unit.

Figure 6 shows an HPC-oriented EE architecture in a vehicle, where one or more HPCs work together with lightweight peripheric units at their functional location.

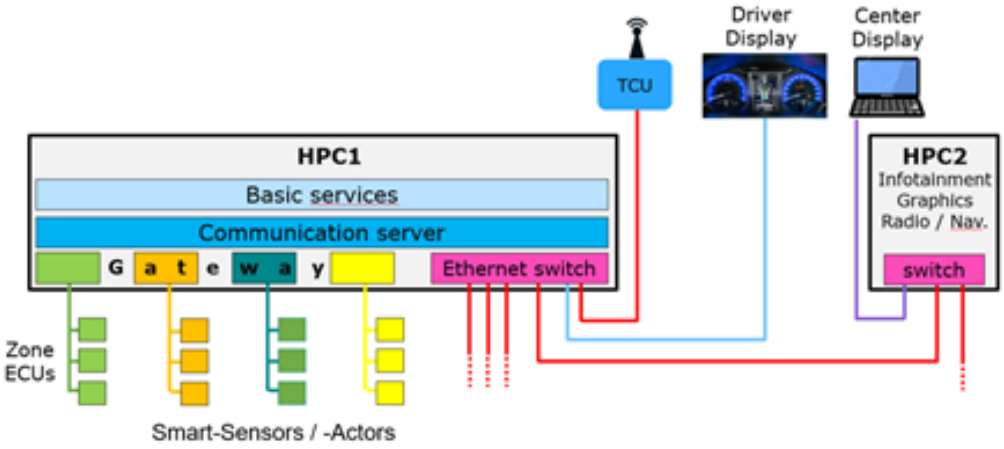


Figure 6 HPC based network architecture

The peripheric units have to be capable to drive the assigned hardware components, to react in a safe manner on system faults, and participate in the vehicle network communication in a secured way. So even these lightweight units need 'safety functions' and 'security elements' (e.g. cryptography) to ensure reliable controlling by the HPCs and safe operation without the risk to be attacked by malicious messages on the vehicle network. Therefore, they should be designed in a way that the required security functionality can be handled in their limited technical boundaries, e.g. real-time-based certificates cannot be processed properly in units without real-time synchronization. Certificate-based security systems may require higher technical effort than these units are designed for.

4.1.5.3 Feasibility and Constraints

Especially high automated driving functions demand high standards of safety and security that have to be met in (nearly) all components of a safety-critical vehicle system.

4.1.5.3.1 State of the art

Certificate-based systems are used for access control to diagnostic functions in the devices (ECUs) of the vehicle using individually granted certificates issued by the cloud facilities. The devices need the technical capabilities to work with certificates what constrains lightweight devices to be integrated into the security system. The communication protection in operation mode is set up in diagnostic mode, and this is why the operational safety depends on an efficient protection of diagnostic functions.

4.1.5.3.2 Proposed Proceeding

This leads to the need to find a scalable security solution that covers all sizes of system components to be ready for the requirements of future use. Since the decision for a security architecture is a far-reaching topic, this has to be planned very well with many future scenarios in sight. The security architectures for future vehicles have to be flexible to meet the requirements of future electronic component designs.

4.1.5.4 Solution

4.1.5.4.1 State of the Art

Certificate-based security solutions are smart to use in complex computer systems providing the necessary environment. On request of an ‘edge service’ located inside or outside the vehicle, one or more certificates are issued by the ‘cloud facilities’ and transported to the ‘devices’ to activate the requested functionalities. This ensures centrally controlled and monitored access control and function activation to provide ‘secure diagnostic’ and ‘function on demand’ in electronic vehicle systems. The protected diagnostic functions are used to set up the communication protection in operation mode. This is mainly the installation or negotiation of cryptographic keys for the ‘Secure Onboard Communication’.

4.1.5.4.2 Proposed Proceeding

Certificate-based security systems require the capability to check the certificates inside the devices. This requires a real-time synchronized clock to check the validity (elapse date) of the certificate, asymmetric cryptography to check the signature of the certificate chain and an OCSP (Online Certificate Status Protocol) request or stapling to ensure that the certificate has not been revoked.

To enhance security functionality to lightweight units, where a certificate-based solution would be too complex, a proceeding can be introduced into the project. This proceeding is designed to provide an adequate security for lightweight microcontroller systems by relocating functionality into a cloud structure and keep the technical effort inside the vehicle low (see Figure 7). The so-called ‘tickets’ in this proceeding carry an access profile and session key(s) from the cloud service into the ECU, which can be used in the following diagnostic operations to sign and encrypt data that shall be kept private.

The ‘cloud facilities’ is named as ‘server’ and the ‘edge service’ (managing unit or diagnostic client) is named as ‘agent’. The secret information, e.g., software or calibration data, comes from the ‘server’ and is signed and encrypted with a session key that is generated by the ‘server’. The session key is transported encrypted inside the ‘ticket,’ which is also generated by the server. Without a ticket, the agent is not capable to generate data that would be accepted by the device because it does not have the session key or the device key.

The proceeding operates with symmetric cryptography in two encryption layers, this gives the opportunity to operate in lightweight environments and tunnel secret information between a cloud service and an electronic unit without being visible in the managing unit (agent) that drives this operation. This includes a flexible access control and a key distribution to install cryptographic material for end-to-end secured data tunnels providing SoA within the limited technical boundaries of the peripheric system.

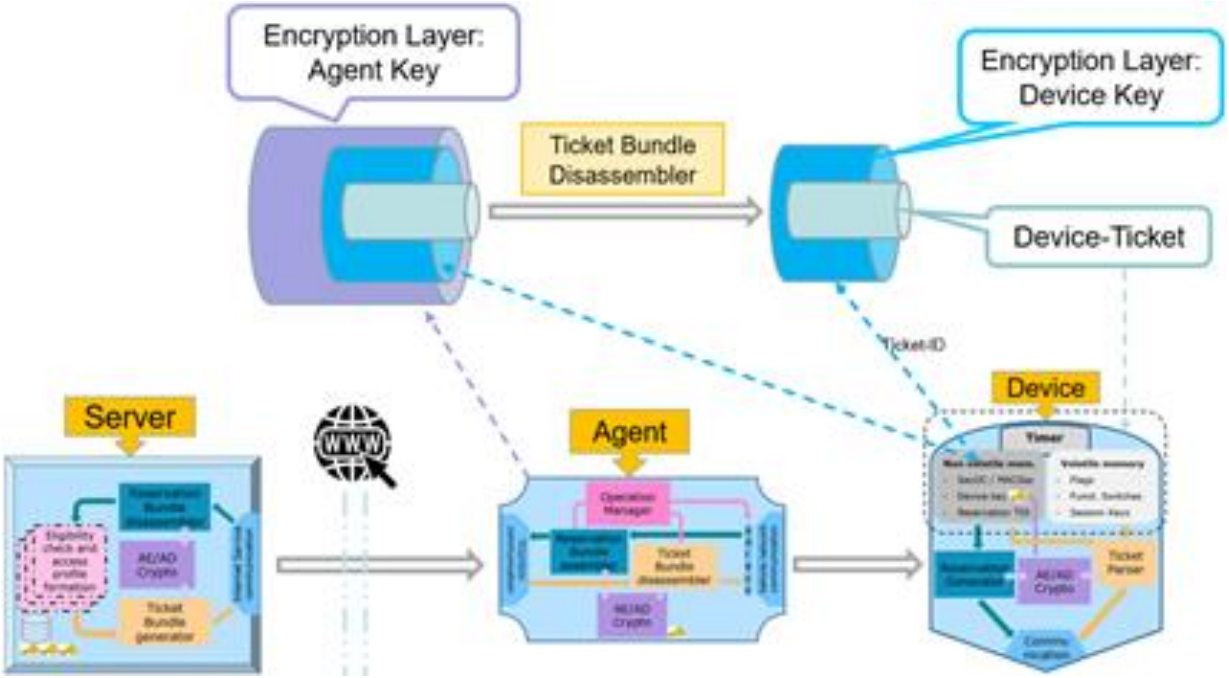


Figure 7: Encryption layer between the entities.

Set as an initial concept, this should be compared to existing security structures in the market to evaluate this for being integrated into the security architecture for the HPC-systems. Such lightweight security solutions abroad from existing IT pattern have to be surveyed for their viability because existing and upcoming UNECE regulations have to be met.

4.1.5.5 Examples

As an example for a lightweight electronic unit in vehicles, a **door lock with soft close** can be considered. This can be made as a mechatronic unit with its own controller connected to the vehicle network. This unit is safety relevant, because it should not open or close unexpectedly, and may have a security functionality to prevent from unauthorized opening by sending a message. In operation mode, the data has to be protected using cryptographic keys that have been installed in diagnostic mode, which has to be made in an authenticated and encrypted diagnostic communication to set up the cryptographic key structure.

A **steering actor** integrated into the steering gear with an electronic control unit can be easily identified as a safety relevant device because wrong steering movements will lead to serious accidents. In vehicles with centrally controlled driving functions, this unit will be made as a smart actor with network connection to the driving control unit and this connection has to be secured against malicious steering commands. The cryptographic key structure has to be set up in an authenticated and encrypted diagnostic communication.

A **BMS (battery management system)** in a BEV (battery electric vehicle) consists of several cell control units spread over the whole battery package to monitor the cell voltage, temperature, state-of-charge and state-of-health. These cell-ECUs have to be kept lightweight to fit into the battery pack. Dependent on packaging and housing these cell-ECUs or the main control unit only may have security functionality to prevent the system from manipulation. Unauthorized manipulation to increase battery capacity may cause battery failures and resulting in fires. The cryptographic key structure has to be set up in an authenticated and encrypted diagnostic communication. In addition to the secured onboard communication the cloud service may request operational data periodically to monitor the system, which needs to be transported in a

protected data channel to a service in the cloud, which can be established via a ticket according to the proposed proceeding.

4.1.5.6 Advantages

4.1.5.6.1 State of the Art

Certificate-based security solutions are smart to use if all components are capable to perform the necessary checks and cryptographic operations. Certificates can be issued using several signing services, which can be references to a joint certification instance and the amount of key material is kept low.

4.1.5.6.2 Proposed Proceeding

The proposed design pattern enhances the capabilities for secure communication to lightweight units with less calculation power that are not capable to fulfil complex security demands.

4.1.5.7 Disadvantages

4.1.5.7.1 State of the Art

In certificate-based architectures, the devices (ECUs) need the technical capabilities to proceed and check them based on real-time and certificate revocation lists that are not available in vehicle devices (ECUs).

4.1.5.7.2 Proposed Proceeding

The proposed design pattern requires a database to store individual keys of all electronic units in the field that should not be exposed to attacks, even if they are not installed in a car due to be spare parts Leaking of this database would expose the vehicles to the risk of attacks. Protecting the key database on the server side from leaking requires IT technology that has to be evaluated to find a secure solution.

4.1.5.8 Impact on Safety

Safety-critical systems have to be protected against unauthorized access to prevent the system from malicious control messages, for this safety-critical systems need security technology to ensure safe operation.

On the other hand, security requires additional calculation power that may limit data flows in the network. Malfunction in security mechanisms may lead to a loss of connection in operational data flow and this may causes dangerous malfunctions or serious accidents.

4.1.6 Secure IoT communications

There are several specifications, industrial forums and even standards targeting industrially relevant cyber-physical IoT systems. Separate specifications have been developed for each sector/vertical domain, such as e.g., home/buildings, manufacturing/industry automation, vehicular/transportation, healthcare, energy, cities, wearables. The TRANSACT project use cases are also focused on different sectors, like healthcare (UC4), industry automation (UC5), automotive (UC3, UC1), traffic in cities (UC1), traffic on sea/ships (UC2). This means that we need to consider the requirements arising from different scenarios and specifications, and to look for horizontal solutions for the focused critical CPS systems. There are initiatives aiming at cross-domain, horizontal type of IoT platforms [26] [27] [28] [29] [30] [31] [32]. In addition, several standardization bodies like IEEE, OMG, W3C, OpenFog, AllSeen alliance, NGMN and ISO/IEC JTC1 WG41 have been working in the area. There are also other related actions ongoing, such as securing IoT products with blockchain [33], and comparisons and related studies of IoT Platforms. For example, Guth *et al.* compare OpenMTC, Fiware, SiteWhere, AWSIoT and provide IoT specification with IoT integration middleware, Gateway and Devices as basic building blocks [34]. Burg *et al.* review wireless communications and security technologies for cyber-

physical systems and conclude that security is an essential challenge for wireless cyber-physical systems operating in horizontal way across multiple domains [35].

Essential solutions in the referred horizontal IoT specifications are related to the edge system and IoT platform. The edge system usually comprises identifiable physical entities, which can be connected to IoT infrastructures and platforms either directly or via some sort of gateway [36] [37] [38] [39]. An IoT platform is typically an integrated physical/virtual entity system capable of controlling, monitoring, information processing and application execution. There are also typically different kinds of tiers defined according to the accessibility of the entities, platforms and enterprise systems. The information models are used to define the properties of IoT information content. In addition, several studies have investigated how virtualization capabilities of IoT systems can be deployed at the edges of the network. However, there are a number of challenges that are not addressed by existing IoT specifications. For example, one essential challenge concerns the collaboration between consumer, non-critical and critical CPS endpoints, which is not properly supported by any of the existing specifications. This challenge is also visible in the UC1 remote operation of autonomous vehicles for navigating in urban environment.

This is especially challenging when speaking about the information and service level, which easily mix the domain and potentially horizontal generic services when handling services related to mixed criticality information streams. The idea of using graphs of operators for stream processing queries has historically been explored in systems like Borealis [40] and STREAM [41], and more recently in frameworks like Millwheel [42], Storm [43], Heron [44], Spark Streaming [45] and Samza [46]. Most of these frameworks focus on data-parallel processing of partitionable streams and strong fault-tolerance guarantees within a single data centre. In contrast, for example Nokia World Wide Streams (WWS) platform [47] [48] focus on distributed and edge computing across wide-area cloud-integrated networks, modelled as a federated set of geographically widespread interconnected compute nodes, stream sources and sinks. While other stream processing platforms address edge locality and distribution within a confined machine cluster, like Quarks [49] and System S [50] [51], WWS is versatile and address wide-area cyber-physical systems. Further, 'Big Data' processing frameworks like Hadoop [52] and Dryad [53] [54] also model queries as data flows. Systems like Hive [55] and Sawzall [56] work with a query language and query plan optimization on top of the MapReduce paradigm [57]. However, they do not consider the wide-area setting, or, like CHive [58], do consider it exclusively from a data analytics perspective, which does not support the open-ended sink distribution, as e.g. needed in industrial CPS scenarios.

The dynamic changes in IoT systems, such as appearing/disappearing of physical entities involving OEMs and related SPs, heterogeneous sensors and actuators and other devices, have proved to be challenging especially from a security point of view. Traditionally, critical industrial IoT devices are often used in physically protected and isolated environments; however, today there is the need to enhance the operation also with many other information providing sources and devices (e.g. traffic signs, lights, cameras and even ordinary people with their computing assets and vehicles in UC1). SoA IoT specifications lack proper solutions for solving this challenge. Proper identification, authentication and authorization capabilities seem to be missing for such dynamic IoT environments, which prevents establishment of trust relationships. Therefore, uncertainty exists in information ownership and validity, and remote management of the physical assets including reconfiguration and reprogramming on the fly. Therefore, it is essential to increase the level of trust in communications between physical cyber-physical resources owned by different stakeholders, for example like the M2MGrids project[2] has done.

The challenges of communication with CPS entities arise from accessibility, trustworthiness, heterogeneity, mobility, and ownership of physical entities [59]. The interaction with physical resources (devices) over the communication networks shall concern especially the ownership, communications, and

[2] https://itea4.org/project/m2mgrids.html (see e.g. documents part of the page)

trust aspects. For example, the communication spaces concept provides a virtual home for people and their resources [60]. The respective idea has also been applied, for example, in the concept of virtual home environment for supporting roaming ecosystems in the mobile context for the mobile device of a user [61], and for sensor devices [62]. CPS systems usually require capability to deliver information from one source to one (1–1) or multiple destinations (1–N) according to the needs of the destinations, which refers to application of publish-subscribe paradigm [63]. For example, Extensible Messaging and Presence Protocol (XMPP) [64] [65] [66], message queue based telemetry transport (MQTT) [67] [68], data distribution service (DDS), advanced message queuing protocol (AMQP), and simple text oriented messaging protocol (STOMP)) or broker-less (e.g., zero message queue protocol (ZeroMQ)) has such capabilities. Naming and identification of owners are provided e.g. by electronic email systems (SMTP, RFC 5321 and 5322), Post Office Protocol (POP), and Internet Message Access Protocol (IMAP)), session initiation protocol (SIP/SIMPLE) [69] and XMPP, which support also management of relationships between owners. Presence management of mobile CPS resources could be supported by e.g. SIP/SIMPLE and XMPP, and e.g. the smart instant messaging (SIM) system provide solutions for presence management, user centric configuration and adaptive grouping [70]. The CPS information content heterogeneity requires multiple data sharing methods ranging from constrained systems (e.g., constrained application protocol (CoAP) for supporting REST-like applications), messaging with known topic names (e.g., MQTT), multimedia content (e.g., SIP) and domain specific content (e.g., energy domain). The capabilities to control who can use the resources could be supported by XMPP “Buddy list” service; however, there is lack of means for defining more specific rules and conditions for the information that can be shared considering e.g. aspects of privacy. Based on the analysis, none of the existing technologies is capable to fulfil all the required aspects. However, for example VTT CPS hub applies the specific features from XMPP and MQTT, combine with the trust solutions to operate with dynamic resources owned by different stakeholders and has evaluated the solutions in traffic accident case [71].

To date, trust, security and privacy are among the primary concerns that limit the widespread adoption of IoT. Roman et al. [72] analyse the features and security challenges such as interoperability and management of access rights with respect to various IoT architectures. This study shows the need to integrate centralized and distributed architectural approaches to provide the foundation of full-fledged IoT. However, this integration has an impact on the efficacy of authorization mechanisms. Ouaddah et al. [73] perform a detailed analysis of existing access control solutions for IoT with respect to domain specific IoT requirements. Alonso et al. [74] identify a number of requirements (related to application-scoped, client-independent, flexible, delegated and configurable) that need to be addressed in order to devise more effective access control mechanisms tailored to IoT ecosystems. A dynamic, scalable, and IoT-ready model based on OAuth 2.0 protocol that allows the complete delegation of authorization, so that an access control mechanism as a service is enabled. The challenges of data sharing in multi-cloud environments are analyzed and an extensible access control markup language (XACML) [6] based architecture, called SAFAX, has been provided [75]. SAFAX’s architecture with prototype realization allows users to deploy their access control policies in a standard format, in a single location, and augment policy evaluation with information from user selectable external trust services. The policy-based access control as a service could enable owners to define and manage their access control policies to protect their resources and support privacy related issues for multiple stakeholders.

4.1.7 Industrial practices

4.1.7.1 AVL (Automotive Domain)

In the field of secure communication, we have used a standalone workshop tester without connection to the internet. The UDS security access 0x27 provides a switch-over protection using a challenge-response-proceeding that can be handled by the software of the workshop tester.

4.1.7.2 PMS: Infrastructure security in the HealthSuite Digital Platform (Medical Domain)

HealthSuite Digital Platform (HSDP) provides an infrastructure solution for medical environments, like hospitals, to migrate from on-premises computing to cloud. For this, it offers a HealthSuite Cloud environment (for cloud services and client application hosting) and a HealthSuite Edge (to connect the on-premises devices to the cloud), which both contribute to a secure solution.

At the highest level, HSDP security should be such that it is compliant to security standards relevant for the medical domain, such as HIPAA [76] and HITRUST [77] . From a development perspective, several controls are in place to safeguard security, including security and privacy by design assessment, static and dynamic code analysis, open-source security compliance, and penetration testing. Also, the quality and regulatory processes include specific focus on information security policy ranging from human resource information security, to operations, to access control, to supplier security policies.

The subsections below focus on computation, networking and low-level communication aspects.

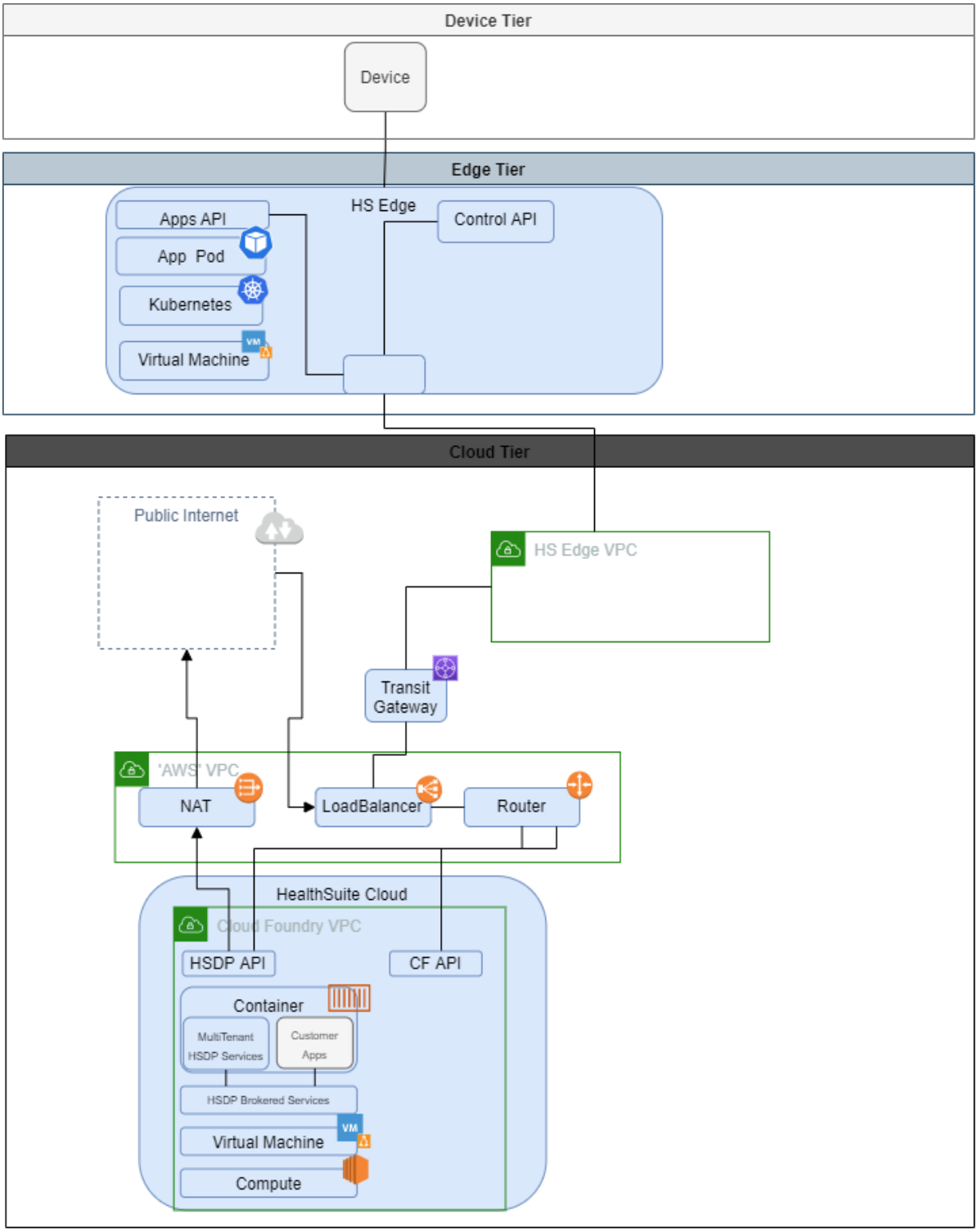


Figure 8: Architectural overview of HSDP Cloud and Edge. Blue indicates 'deployed/owned by Philips or its partners', Grey indicates 'deployed/owned by Philips customers/3rd party', the bounding boxes map it to the Transact Architectural Tiers.

4.1.7.2.1 HealthSuite Cloud

HealthSuite Cloud is an open, cloud-based healthcare platform that supports data collection across the spectrum from clinical to consumer lifestyle, allowing for integration and analysis of data from distributed sources worldwide. HSDP is built on top of Open Source Cloud Foundry and is designed to operate as a multi-tenant environment. The HealtSuite Cloud components are shown at the bottom 'Cloud Tier' in Figure 8.

Secure communication in this platform does not only depend on the encryption of data in transit, but also to a large extent on the virtual network architecture, the applied access policies, and its capabilities to ensure resource isolation to prevent breaches. Below we discuss the design choices made in these areas.

HSDP builds on top of AWS. AWS adopts a shared responsibility model [78], where AWS covers the "security of the Cloud" and the customers are responsible for "security in the cloud". Hence, HealtSuite Cloud is therefore one side of the AWS shared responsibility model. From a least-permissive perspective, we adopt a security model where clients/ops/admins only get a minimal number of roles/privileges, required to comply with additional security measures for access / login (e.g., MultiFactor authentication) in sensitive areas, and strict pre-conditions for elevated permissions in sensitive areas. There is also a strict separation between the development, operations, and product environments to avoid breached 'from the inside'.

In the product environment, the following designs are applied.

Build up on AWS: AWS services are typically publicly accessible (e.g., S3). HSDP adopts following to increase security in AWS services:

- HSDP provides a service layer on top of public AWS services (S3…), including an additional Identity Access Management (AIM) layer.
- HSPD leverages Virtual Private Cloud (VPC) to separate networks logically into network segments. The AWS services are in private subnets, with limited roles for the few public facing services (e.g. Elastic Load Balancers).
- Administrative access to EC2 instances is only allowed via Secure Shell (SSH) gateways.
- Applications/services run on a private network, which is shielded from ingress internet traffic.

The HSDP environment routes ingress traffic such that this never directly connects to the public internet. All ingress traffic traverses to load balancers, which does SSL termination and forwards it to the Private Network. All egress traffic traverses a Network Address Table (NAT) gateway.

The applications/services hosted on HSDP are enforced resource isolation by the following measures:

- Applications/services are hosted in containers leveraging Linux namespace primitives for isolation. There is no hosting on server/virtual machine/NGinx/…
- Privileged containers cannot be created; execution is in rootless containers with non-privileged accounts.
- There is no direct communication between deployed containers; all communication goes via routers.
- Ports assigned during deployment of a container are ephemeral with limited protocol allowed (e.g. TCP)
- Deployment of services/application is only allowed via a non-public docker image repository.

The applications/services hosted on HSDP need to use secret information that would compromise security if publicly available. To secure this secret information, the applications/services are enforced to use a Vault for storing information classified as secret (pushed into the Vault during deployment).

To increase scalability and decrease cost often a multitenancy pattern is used. This allows a single deployment to serve multiple customers. To secure communication, measures must be taken to ensure isolation and avoid one customer accessing another customers data. HSDP offers MultiTenancy and has measures to avoid information leakage from one tenant to another by

- MultiTenant deployments for stateless services, supported by IAM org model and per-tenant-APIs
- Single Tenant deployment for stateful services (e.g., database)

HSDP offers brokered services for basic infrastructure elements, like databases, caches, and messaging.

- There is access control such that these services are only accessible by the Services/App it was deployed for.

4.1.7.2.2 HealthSuite Edge

HealthSuite Edge is designed to provide a safe and easy to maintain virtual private network and lightweight application hosting environment that can be centrally managed from the HealthSuite Cloud. HealthSuite Edge is shown in the Edge Tier of Figure 8. One of the prime goals is to implement a single secure connection point to the above mentioned HealthSuite Cloud, but also other Philips/Partner networks.

Security in communication is realized by:

- Separation of data plane and control plane by role-based access control. The data plane is responsible for networking aspects like address allocation, external and internal overlay networks and access to foreign networks. The control plane is responsible for device management e.g. device provisioning, firmware updates, and configuration of the control plane
- Reducing attack surfaces by avoiding the exposure of generic deployment APIs like Kubernetes APIs.
- Applications run in isolated non-privileged pods (containers) on non-privileged virtual machines for maximum isolation of the device from potential hostile applications.
- VPN Tunnelling and communication uses chacha20/poly1305 for encryption as standardized by RFC8439 [79].

4.2 Self-monitoring & Awareness

Distributed systems are usually highly complex, not only because of their main characteristic of being located in multiple machines, but also because of the different heterogeneous systems they contain, connected by several network topologies. These systems interact with each other and with the human and external environment, making it difficult to predict and maintain. To help in this task, they should be given a greater capacity to adapt their own behaviour at runtime, with special emphasis on observing the internal and external context.

The TRANSACT reference architecture includes the safety, performance and security monitoring services, which are not only responsible for monitoring the current status of the system, but also for detecting and preventing safety, security and performance failures (see Section 3). These concepts of self-observation and failure detection are encompassed by the topic of self-awareness, which is explained in this section.

4.2.1 The concept of self-awareness in computing systems

The complexity of distributed architectures has increased the need to improve this type of system with the capability to obtain and represent knowledge about itself and its experiences, providing them with greater

adaptability [80]. This knowledge, as known as self-awareness, depends on the ability of the system to measure both the internal and external context and the analysis that it can make of the observed data. Private self-awareness is the ability to know the internal phenomena, such as the response time of one of its activities or the throughput in an internal communication channel; otherwise, public self-awareness is defined as the knowledge of the external environment, e.g. the location of a robotic device relative to its human operator.

The two main characteristics of a self-aware computing system are: (i) the ability to observe/capture the necessary information, both internal and external, to complete a pre-existing model of the state of the system or to model it continuously; (ii) use these models to reason about this knowledge and act accordingly [81].

A common approach to design these systems is the MAPE-K loop (Monitor-Analyze-Plan-Execute on a Knowledge base) [82]. The K component of knowledge stores the state of the system, the Monitor (M) registers the data, the Analyzer (A) processes this data together with the model, the Plan (P) defines the workflow to be carried out and the Execution (E) manages the actuators to perform the planned actions (see Figure 10).

When modelling self-adaptive systems, the following must be taken into account: (i) the objectives of the system, (ii) the changes to which it must adapt, (iii) the mechanisms available to adapt, (iv) and the impact after adaptation [83]. In the case of self-aware computer systems, this adaptation does not need to be perform directly by the system, it being possible that the action is informative towards the outside [81].

Both the MAKE-K loop and the objectives of the system have been previously described in section 4.1. This is due to the strong relationship between self-awareness and SLAs, on the one hand between monitoring phase and the indicators described by the SLIs and on the other between data analysis phase and compliance with the SLOs.

4.2.2 The vision of Self-aware Cyber-Physical Systems

Cyber-physical systems include an additional difficulty that is to integrate into environments with physical processes. New challenges arise from this difficulty, such as allowing human interaction, flexibility to respond to changes in the internal state (available resources, local environment, objectives...) or dynamically adapting to the use of resources in the cloud [84].

Increasing the autonomy of these systems is necessary to solve new challenges, providing them with capabilities that allow them to adapt to different situations. The concept of self-awareness can help solve these and other challenges, keeping in mind that it is not a monolithic property, but rather made up of several capabilities at runtime. These properties include self-monitoring, self-modelling, learning, self-analysis, self-adaptation or self-report [84]. Each system will include the runtime capabilities necessary to achieve its goals.

In recent years, different projects have emerged that include self-awareness capabilities. The CERBERO project [85] is developing a design environment composed by modelling, deployment and verification tools for adaptive CPS. The authors focus on its efficient support for run-time self-adaptation. CERBERO provides three different types of adaptability: i) functionally-oriented (changes in working-point or in algorithms during execution), ii) non-functionally-oriented (meet with requirements as performance or available energy), and iii) repair-oriented (providing self-healing and self-repair mechanisms).

An agent-based framework for modelling of self-aware applications is presented with RoSA [86]. The authors present a decentralised architecture for defining these applications, which are decomposed into agents, defined by functionalities. Each agent in RoSA operates in an ODA loop (Observe-Decide-Act), receiving inputs, manipulating data, and sending outputs (see Figure 9).

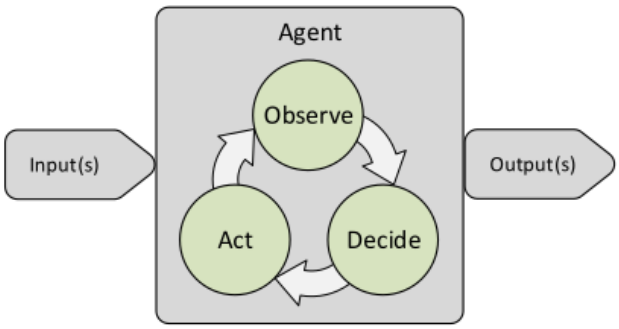


Figure 9: Observe-Decide-Act loop (from [86])

The SECREDAS project focuses on making future (safety-critical) autonomous driving secure from external malicious interference or hacking that would put car passengers or other road users in danger [87]. This project proposes the concept of security supervisor. This centralized supervisor must adapt to a heterogeneous distributed environment, due to the nature of a security scenario.

Focusing on the TRANSACT solution, self-awareness is not only related to compliance with the SLA, as described in section 4.2.1, but also to the input of the different services in charge of managing the distributed system. The operational mode management (see section 4.4.1) expects to receive the mode change requests, that can be provided as a decision to adapt the system to unexpected events. On the other hand, self-awareness can provide the knowledge to help prevent security issues (see section 4.1.1), e.g. monitor and analyse network traffic to detect possible attacks.

4.2.3 Self-monitoring in safety-critical CPS

Designing the architecture of a self-aware CPS comprises the design of its different capabilities (such as self-modelling or self-adaptation). One of the most important to consider is self-monitoring, which is also included in the first phase of the MAPE-K loop (see Figure 10). A correct definition of the data that will be necessary to know the status of the system is of great importance. To do this, knowing the taxonomy of monitoring concepts can help to design the correct solution [88]. In addition, it is important to know the different monitoring tools available on the market, which provide solutions for online verification (also known as runtime monitoring).

In [89], the authors present a concept for verification of timing aspects in distributed systems. The methodology is based on consistent handling of real-time properties in the design process based on the MULTIC methodology [90] [91]. The refinement of the specification along the system decomposition can be verified with a virtual integration test (VIT) based on the MULTIC tooling [92]. The general idea of the online verification approach is the generation of timing monitors from the specification of the system. Event recorders and timing monitors are directly generated from timing specifications. The accuracy of the concept is evaluated in a prototype and applied to a distributed system architecture.

Another runtime monitoring approach has been implemented as part of the art2kitekt tool [93]. The authors describe a runtime monitoring tool, used to measure and to visualise the temporal characteristics of a real-time application. The results of this characterization are used to analyse the temporal behaviour of the system.

In [94] [95], the authors present the concept of an uncertainty wrapper for data driven models like neural networks. The uncertainty wrapper is used to quantify the likelihood that the model's output is wrong, i.e.,

its uncertainty. In [96], they discuss how this quantitative uncertainty assessment could be used for safeguarding safety constraints dynamically during operation.

There are several challenges to consider when designing the monitoring architecture in TRANSACT. One of them is the heterogeneity of the monitors distributed in the device-edge-cloud continuum, which must be transparent to the components in charge of processing and analysing the collected data. Another challenge to take into account is the synchronization of the different clock sources that allows comparing the timestamp from different components.

4.3 SLO enforcement in the compute continuum

Application workload scheduling in hybrid Cloud-Edge-Device infrastructures has been extensively researched over the last years. The recent trend of containerizing application workloads, both in the cloud and on the edge, has further fuelled the need for more advanced scheduling solutions in these hybrid infrastructures. Unfortunately, most of the current approaches are not fully sensitive to the edge properties and also lack adequate support for Service Level Objective (SLO) awareness. The management of SLOs in this compute continuum is a complicated task due to the great heterogeneity and variety of computing infrastructures and networks and their dynamically changing operational runtime conditions [97]. This section deals with the currently existing solutions for managing SLOs and, more specifically, SoA research for enforcing them in the compute continuum.

4.3.1 Service level terminology

In order to manage a service correctly, is essential to understand which behaviours really matter for that service and how to measure and evaluate those behaviours. Three key concepts that describe basic properties of metrics that matter, what values should that metrics have, and how to react if the expected service cannot be provided are: service level indicators (SLIs), objectives (SLOs) and agreements (SLAs) [98].

An SLI is a **service level indicator**: a carefully defined quantitative measure of some aspect of the level of service that is provided. User-facing serving systems generally care about availability, latency and throughput. Storage systems often emphasize latency, availability and durability. Big data systems, such as data processing pipelines, tend to care about throughput and end-to-end-latency. In addition, all systems should care about correctness. Some of these indicators are not directly observable and must be computed from the combination of other indicators. As [99] appoints, security is another non-functional property that SLIs could also include.

An SLO is a **service level objective**: a target value or range of values for a service level that is measured by an SLI. A natural structure for SLOs is **SLI ≤ target**, or **lower bound ≤ SLI ≤ upper bound**. Choosing and publishing SLOs to users sets expectations about how a service will perform, which is called the **quality of service** (QoS), since potential users often want to know what they can expect from a service in order to understand whether it is appropriate for their use case. Thus, choosing an appropriate SLO is complex, since it is highly dependent on the system's nature and purpose, the execution runtime, the expected workload and the underlying infrastructure, among other factors. As [98] refers, working from desired objectives backward to specific indicators usually works better than choosing indicators and then coming up with targets.

Finally, an SLA is a **service level agreement**: an explicit or implicit contract with users that includes consequences of meeting (or missing) the SLOs they contain. When applied to public cloud providers, the

consequences are usually financial, but consequences of other types could also be stipulated, depending on the system's nature and criticality.

4.3.2 Enforcing service level objectives

Going deeper into SLOs, whose enforcement is the matter of this section, service level objectives should specify how they are measured and the conditions under which they are valid, possibly defining multiple objectives for a single indicator. In case of dealing with heterogeneous workloads, it may be appropriate to define separate objectives for each class of workload. Continuous meeting of service levels in heterogeneous or not directly controlled infrastructure, such as cloud and grid computing environments, can be ensured by **measuring and dynamically adapting service performance**.

An approach for the negotiation of SLAs of business processes and the corresponding QoS enforcement at the level of infrastructure in cloud and grid environments is presented in [100]. They are based on service-oriented architectures, focused on the interaction of autonomous and interoperable services that offer reusable business functionality via standardized interfaces, which has evolved into which nowadays is called a microservices architecture. By comparing the aggregated quality of service level provided for each individual service and the corresponding SLOs, they decide how to apply service replication, improving service levels regarding response time, transaction rate, throughput and availability. This ultimately results in the **automatic provision of service compositions** that best meet the requirements of business processes, that could be powered by service composition middleware for pervasive computing environments [101]. Of course, monitoring during runtime is crucial, so that the actual service levels can be compared to the expected ones.

As [100] hints at, it is not sufficient just to specify SLOs and apply monitoring during runtime in order to check if SLAs are either being met or violated. In case of deviations from the target values specified, **adequate countermeasures must have been selected along with SLOs**, so that they can be applied to restore the expected QoS. By following this research line, [102] proposes their Web Service Requirements and Reactions Policy Language (WS-Re2Policy), which can be used to specify monitoring requirements and countermeasures to SLA violations simultaneously. The resulting policy, that comprises the specification of requirements and reactions, can be deployed to different monitoring units for distributed SLA monitoring and enforcement. Of additional interest is the approach followed by Oracle in their Web Services Manager [103], which integrates centralized monitoring and policy enforcement with distributed information gathering of basic QoS parameters (e.g., response time), exceptions and security aspects. Therefore, an architecture containing non-intrusive (i.e., gateways) and intrusive (i.e., agents running the same application server as the web service) was developed, which allows both basic reporting of monitoring results as well as automatic selection and activation of policies based on given measurements.

Automatically reacting to an SLA violation with a dynamic reconfiguration of a service or component's replication factor [104] is called **elasticity**. To efficiently adjust the elasticity of a deployed cloud application upon a workload, based on its SLOs, a **Monitor Analyze Plan Execute (MAPE) loop** [105] is usually implemented. It comprises four phases: the *monitoring* of the system and workload metrics, the *analysis* of the metrics to evaluate whether the defined goals are met, the *planning* of actions to correct a violated SLO, and the *execution* of the planned actions (see Figure 10). The MAPE loop is also explained in detail in deliverable document D3.1.

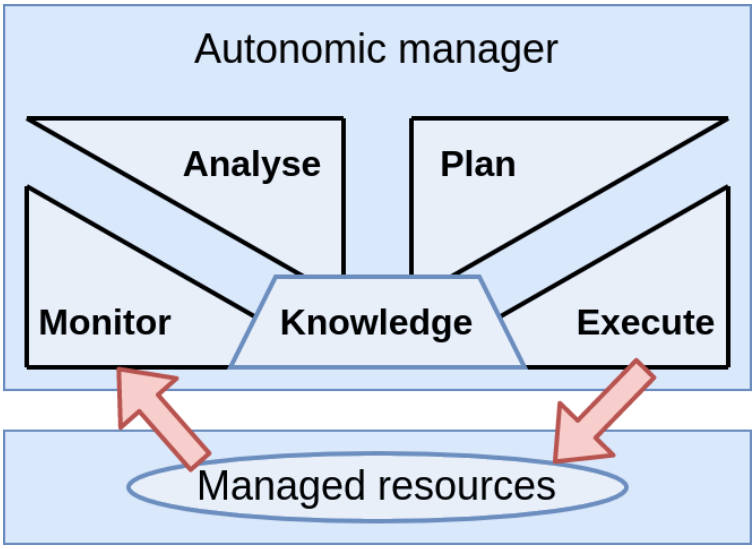


Figure 10: MAPE-K autonomic loop

4.3.3 Transition from the cloud to the cloud-edge-device compute continuum

Performing operations over large quantities of data that continuously move from field sensor devices to a cloud computing infrastructure, that should be processed and responded upon in a timely manner, can lead to low QoS that can be due to connectivity performance issues between the field devices and the cloud infrastructures. As a result, more distributed computing paradigms that are in close proximity to data sources have emerged as a means to address such non-functional requirements. Hybrid cloud-edge-device architectures have been widely adopted over the last years to address the requirements of computationally, memory and network intensive tasks [106]. They have proven useful to perform analytics on incoming data from a massive deployment of ubiquitous devices, support mission-critical tasks and assist end devices in performing heavy or real-time processing. In a service-oriented (or microservices) architecture, deploying services in **the cloud-edge continuum benefits from both the low access latency of the edge nodes and the high computational ability of the cloud layer** [107].

The introduction of edge computing brings the advantages of locality and proximity to cloud computing, minimizing latency and increasing bandwidth between the device and the nearest computing node. This extends cloud computing by providing several advantageous features, such as the next four [108]:

1. Highly responsive services, due to physical proximity between the device and the edge.
2. Scalability via edge analytics. Raw data flows from to the device to the edge via a high bandwidth local network, gets analyzed and only a much smaller quantity of extracted information and metadata is transmitted to the cloud.
3. Privacy-policy enforcement. By serving as the first point of contact in the infrastructure for device sensor data, the edge layer can enforce the privacy policies of its owner before releasing data to the cloud.
4. Cloud outages masking. If a cloud service becomes unavailable, a fallback service on the edge can temporarily mask the failure.

However, since one of cloud computing's driving forces is the lower management cost of centralized (yet remote) infrastructure, **the dispersion inherent in edge computing raises the complexity of management** considerably. Thus, developing innovative technical solutions to reduce this complexity is a research priority for edge computing [108].

4.3.4 SLOs in the compute continuum

The recent trend of containerizing application workloads, both in the cloud and on the edge, has also fuelled the need for more advanced application workload scheduling solutions. As these architectures become more complex and more dependent on third-party services, often deployed across multiple computing infrastructures from various providers, SLOs enforcement becomes increasingly important [97]. However, the **management of SLOs in this compute continuum is a complicated task** due to the great heterogeneity and variety of computing infrastructures and networks and their dynamically changing operational runtime conditions, which influences the resulting QoS [106]. Unfortunately, most of the current approaches are not fully sensitive to the edge properties and also lack adequate support for SLO awareness. This is due to the fact that traditional SLO enforcement and violation mitigation mechanisms, such as elastic scaling, are not as easily attainable for application workloads running in edge environments as they are for more traditional cloud-native workloads [97].

Next generation SLOs must go beyond the definition of the specific and measurable capacity guarantees of a workload, providing information about a workload's performance that turns out to be easier to understand, communicate and generally more relevant. Furthermore, there is a need to clearly map workload performance requirements to the resource capacity guarantees. In this line, SLOC [109] introduces a framework to enable SLO-native management of elastic cloud resources, shifting from general, business logic agnostic, low-level SLAs to intent-based, SLO-first, performance-driven and orchestration-aware elasticity models. In addition, [110] proposes a middleware that provides an orchestrator-independent SLO controller for implementing complex metrics, periodically evaluating SLOs and triggering elasticity strategies, while decoupling SLOs from the elasticity strategies to increase flexibility. Finally, [111] implements a service-based SLA management platform that generates comprehensive SLA violation's explanations in order to aid the user to renegotiate SLAs.

With the compute continuum setting, it is necessary to find new and dynamic means for the management of SLOs in view of achieving high QoS operation of the microservices deployed into containers [106]. Therefore, for edge-native workloads, it is important to **address SLOs as early as possible in these workloads' orchestration and management lifecycle**. Typically, this means considering SLO constraints and requirements already in the CI/CD pipeline, i.e., during the deployment phase [97]. A possibility for meeting SLOs in the compute continuum is by federating several computing resources and providing an orchestrator and a load balancer capable of automatically redeploying containers from one computing resource to another, in a dynamic and decentralized way, in view of maintaining high QoS. From this perspective, [106] proposes a Markov Decision Process (MDP) method for **automated decision-making based on the probability of achieving high QoS**. This method is used to build an automaton (probabilistic model), whose states represent all available deployment options and whose transitions contain probabilities for maintaining an idle state or redeploying a software component from one deployment option to another, which happens upon the violation of a QoS metric. In this regard, the deliverable document D3.1 also points to the elaboration of a performance management framework.

Since its inception in 2014, Kubernetes has put itself as a de facto standard for orchestrating and managing containerized workloads in the cloud. While its scheduler is well optimized for the cloud environment, where

compute nodes are powerful and network connections have consistently high throughput, it largely lacks features needed for allocating resources and scheduling containers in edge environments. Some solutions offer Kubernetes distributions specifically tailored for the edge. Particularly, KubeEdge [112] intends to extend the cloud capabilities to the edge, connecting and coordinating two computing environments for applications leveraging both layers of computing resources. It provides the network protocol infrastructure and the same runtime environment on the edge as on the cloud, which allows the bidirectional communication of applications with components running on edge nodes as well as on cloud servers. Nevertheless, none of these solutions provide adequate support for considering SLOs for workload scheduling at the edge.

These container orchestration solutions usually only consider the availability and current usage of the computational and memory resources in the nodes, but, since the edge nodes are usually geographically distributed, the edge-edge and the edge-cloud communications may need to flow through unstable public networks. Other factors should be considered when facing service deployment in a cloud-edge continuum, such as the **computational capability** of each node, the **resource contention** on each node, **data locality** and the **communication overhead**, all of them affecting the end-to-end latency of an application. The authors of [107] identify three main challenges that have to be resolved to ensure meeting the SLOs of a microservice-based mission-critical application while minimizing the used resources in the compute continuum:

1. The communication through a public network (cloud-edge) may result in a long latency. A method is required to effectively map the microservice stages to the cloud and edge nodes, while minimizing the communication overhead.
2. The shared resources contention between microservices on the same node could result in a QoS violation, since allocating more resources to a microservice may slowdown other microservices on the same node.
3. The optimal microservice deployment could vary accordingly to the load of an application.

Since the shared resource contention on each node and the load of an application are only exactly known at runtime, **an offline method is not able to identify the optimal deployment** and when a service has to be migrated between layers in order to keep meeting SLOs. To this end, they propose Nautilus [107], a runtime system for the cloud-edge continuum comprising a communication-aware microservice mapper, a contention-aware resource manager and a load-aware microservice scheduler. Its mapper places microservices that frequently interact with data on the same node to alleviate network communication overhead, while its resource manager and microservice scheduler ensure that the microservices use the minimum computational and network resources while ensuring the QoS.

The sources reviewed for summarizing the state of the art of this research topic span from 2008/2009, narrating the arrival of cloud computing and terms such as service level agreement [100] [102], through 2017, dealing with the then unfulfilled requirements that edge computing intends to address [108], until 2020/2021, when references to SLO enforcement in the compute continuum first appear [106] [107] [97]. This displays that we are facing a **topic**, that was only recently tackled. This challenge makes our research to become fully innovative exciting but, at the same time, it also denotes that **no full solution exists nowadays**, neither in the academic field nor in commercial spheres. The solutions proposed by the **MDP for automatic redeployment maximizing QoS** [106] and the **Nautilus runtime system for the compute continuum** [107], respectively, currently look as the most promising choices and open the way for further advances in search of more complete and sophisticate solutions.

4.3.5 Industrial practices

4.3.5.1 Philips Medical Systems: SLAs for Cloud and Edge - HealthSuite

HealthSuite digital platform provides an infrastructure solution for medical environments like hospitals to migrate from on-premises computation to cloud.

Regarding HealthSuite Digital Platform Service Agreement:

The HealthSuite digital platform (HSDP) offers a native cloud-based infrastructure based on Cloud Foundry and the core platform services needed to develop and run a new generation of connected healthcare applications. HealthSuite has a generic SLA, which has following characteristics:

- Defines service expectations and responsibilities between the Client and HSDP for HealthSuite's Services.
- Service Performance: General service availability target of 99.99% availability, where availability is specific for the service characteristics, e.g., accessibility of data and retention periods.
- Security Breach Notification: Client notification within 48 hours of determining existence of a Security Data Breach.
- Hosting services: For services that host client applications, the SLAs focus on performance of the underlying infrastructure, but due to high amount of customization there is a clear client-side co-responsibility to guard performance and security.
- Backup/Disaster Recovery: Shared responsibility model. (Platform vs. Client Applications). Depending on the nature of the service, there are specific Recovery Time Objectives (RTOs), and Recovery Point Objective (RPO) defining the interval between backups.
- API: APIs which are deprecated/changes are announced and supported for 18 months.

Monitoring aspects are covered in Section 4.2.

4.4 Operational mode management

The TRANSACT architecture should offer support for systems and applications that are comprised of several modes of operation. This section will detail what is considered as an operational mode and some state-of-the-art research in regards of the schedulability analysis of multi-mode systems.

4.4.1 The concept of operational mode

There has been an evolution in traditional real-time systems with specialized dedicated hardware, where they need to interact with dynamically changing environments or scenarios. For instance, many applications in the real world operate in different clearly defined modes of operation [113]. Serving as an example, the Adaptive Cruise Control System (ACCS) of a car provides an intelligent way to adjust vehicle speed to guarantee safety properties. Particularly, in ACCS there are two real-time modes to be found: Non-Critical Mode (NC Mode) and Safety-Critical Mode (SC Mode). In the first mode, the parameters of the system are rapidly changing, while that is not the case in the latter mode. Both modes are mutually exclusive and the vehicle is in either one mode at any time instant while driving. The characteristics of the system, particularly

the executed task set in this scenario, may vary over time when a change between modes happens. This is typically referred to as mode transition and the system must also maintain schedulability during this period.

Each operational mode represents a different goal under the current environment it is facing. Hence, not all software components or tasks may be active in all possible modes of a system. Instead, each mode will contain a specific subset of them. As a result, a mode transition may require the activation of a new task, and the deactivation or changes in existing tasks. It is during this transition phase where tasks belonging two both modes may coexist, which in turn may result in deadline misses that would typically be avoided in their static modes. Figure 11 illustrates the different types of tasks and their behaviour upon a transition between two modes after a mode change request (MCR) has been issued.

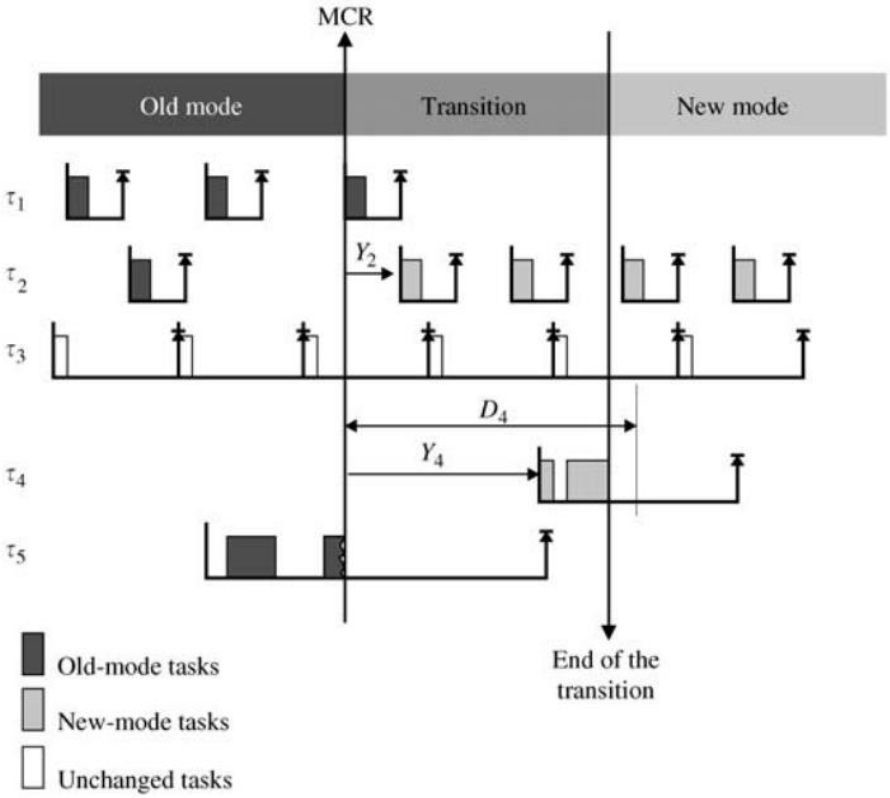


Figure 11: Behaviour of tasks during a mode transition (From [114])

In conclusion, in order to guarantee that a real-time system works correctly, it is not only necessary to ensure that each of the tasks that are executed within a particular mode are executed within the appropriate time restrictions, but that this continues to happen also during the transition phase between two different modes.

4.4.2 Analysis of multi-mode systems scheduling

All timing constraints must be satisfied in safety-critical systems. In the case of systems with several modes of operation, like the ACCS described in the previous section, each mode may have its own requirements in terms of schedulability. It is important to guarantee that these requirements are met in all modes and in all transitions between them. Analytical methods can help in the process of checking the feasibility of a system. This section will briefly summarize some state of the art in that regard.

Several studies have been carried out on the analysis of task planning in multi-mode systems and their mode change protocols [114]. Tasks can be classified according to their behaviour during the transition between

modes. For instance, a task belonging only to the old mode, can either be allowed to complete its whole execution after a mode change request (MCR) or be aborted. Meanwhile, tasks that belong two both old and new modes can have their temporal parameters (i.e., activation period and/or deadline) changed or unchanged. Lastly, there are those tasks that belong only to the new mode.

Protocols can be classified depending on how they handle the aforementioned task types. A protocol with periodicity, for example, preserves the activation pace of unchanged tasks. Meanwhile, a protocol without periodicity delays the activation of unchanged tasks, therefore affecting them during the transition. This might be necessary in order to preserve the feasibility of the transition.

Additionally, a synchronous protocol does not allow new-mode tasks to be released until all the old-mode tasks have completed their last activation. On the other hand, asynchronous protocols allow a combination of both to be executed at the same time during the transition. These protocols require specific schedulability analysis, since the workload during the transition phase can be higher than in the steady states.

Real et al. [114] proposed an asynchronous protocol with periodicity for single-processor, fixed-priority (FP), pre-emptively scheduled real-time systems. In said protocol, they propose that after a MCR arrives, old-mode tasks are allowed to complete normally and that both new-mode tasks and changed tasks are introduced after a delay relative to the request time. On the other hand, unchanged tasks are preferably introduced without an offset. They also propose a solution of how to calculate said offset.

Stoimenov et al. [115] introduced methods for scheduling analysis during mode transitions that guarantee the timing behaviour in multimode systems with both FP and Earliest Deadline First (EDF) scheduling policies. Said method can also be applied to any hierarchical combination of both scheduling policies and to any arbitrary task activation pattern.

Many embedded media systems (e.g. smartphones and digital video recorders) must support multiple simultaneously-executing subsystems on a shared execution platform, where temporal isolation is desirable to ensure its specified quality-of-service, even in the presence of mode changes. To assist in the design of such systems, Fisher et al. [116] propose a hard-real-time schedulability analysis for subsystems executing in a temporally-isolated environment under both resource and application-level mode changes.

Regarding multicore systems, some research has proposed a mode-change protocol and its corresponding analysis for multiprocessor scheduling of multimode systems with mode-independent tasks using global-EDF [117].

ACCEPTOR [118] is a model and an asynchronous aperiodic mode-change protocol for multi-mode applications on reconfigurable heterogeneous platforms. The protocol is composed of an offline phase and a run-time phase. The offline phase computes the required reconfigurations that must be done for each mode change phase. During the run-time phase, the protocol must successfully schedule the remaining jobs of the old mode and reconfigure the processors as computed in the offline phase.

In recent works, Baek et al. [119] present a new response time analysis (RTA) framework for mode transitions in real-time multiprocessor systems. To improve the analysis, they enforce the transition order of tasks, and propose a transition sequence assignment algorithm by deriving the useful properties of an effective transition order under a given restricted condition. A challenge yet to be addressed is to adapt this analysis technique to multi-mode distributed systems so that it could be adopted by the TRANSACT architecture.

4.5 Software updates in safety-critical systems

Cyber-Physical Systems have components connected to network infrastructures and used to interact with the physical world. These components are susceptible to vulnerabilities and security attacks, which can produce negative consequences, such as loss of availability or problems in the integrity of the system. Software updates are an essential strategy to improve security and fix safety issues during the life cycle of these systems. They are also useful for including new features or adapting to new requirements. Updating is also an essential part of an DevOps cycle.

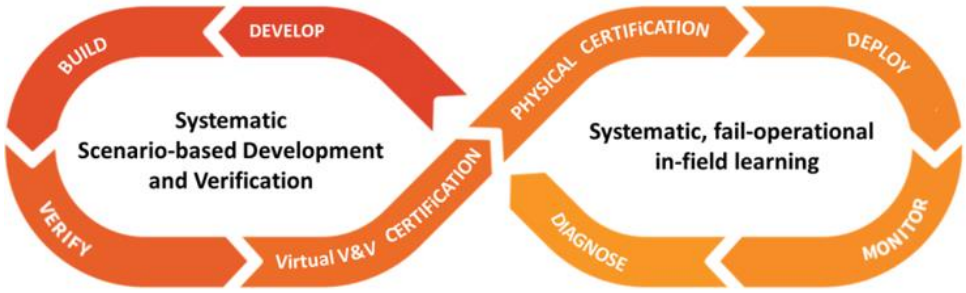


Figure 12 DevOps cycle

One of the most widely used practices is Over-the-Air (OTA) updates, which define how to update a device from a remote location. In recent years, several studies with taxonomies and surveys have been developed [120], defining the different mechanisms involved in the OTA update process: secure and safe downloading, updates management, code installation, system recovery, updates scheduling and software packaging. Others review the key principles and phases to remote update of an embedded device [121], such as compiling and verifying the software, the secure updating, and the installation and activation.

The complexity associated with the update process, including authentication, confidentiality and integrity of the software image, was reason for the Cloud Security Alliance IoT Working Group to compile a list of key recommendations for establishing a secure and scalable IoT update process [122]. With the same focus on security challenges, the authors in [123] present a compilation of research works based on IoT software updates, comparing client-server-based and blockchain-based architectures.

In recent years, several projects have been presented to solve the difficulties of OTA updates. The SUIT project [124] is a proposal of standard for secure IoT firmware updates, evaluating the performance of the implementation on a variety of commercial off-the-shelf. The presented architecture provides a solution using open standards in security and communications. The ASSURED project [125] is a secure and scalable update framework for IoT. ASSURED includes all stakeholders in a typical IoT update ecosystem, while providing end-to-end security between manufacturers and devices. And also, the GUITAR project [126], a generic extension for IoT architectures, that enables dynamic application and network level upgrades in an efficient way. In this project the authors introduce a number of design concepts essential for supporting remote updates. Additionally, an architecture is proposed that can be applied to existing operating systems, making them upgradable. In the following sections we will highlight the state of the art of Over the Air (OTA) Updates from the perspective of different domains. We will discuss the automotive, manufacturing industry and maritime domain.

4.5.1 Automotive

There is currently no single uniform standard for Over the Air (OTA) updates in the automotive domain. Each manufacturer has its own methods for this. Most manufacturers currently install and check safety-relevant updates during garage visits. In the area of comfort functions such as navigation or infotainment, updates and bug fixes are already carried out outside the garage.

With the introduction of the automatic emergency call (eCall) on 31 March 2018, OTA updates would also be possible via the communication interface with SIM card as an alternative to Wi-Fi. At the moment the use of eCall interface is not intended for updates. With regard to OTA updates in Europe, safety and security requirements and solutions are discussed by experts. The United Nations Economic Commission for Europe (UNECE) has initiated a task force for developing a set of guidelines for cybersecurity and OTA updates in vehicles, called CS/OTA [127].

The automotive industry clearly sees the challenges of future application in this domain. With the AUTOSAR Adaptive initiative, the classic AUTOSAR platform for supporting the requirements arising from these applications is to be enriched. Although the implementation of online updates has been identified as a requirement [128], major challenges such as central validation issues and diagnostic options (so far) remain unconsidered.

Uptane [129] is an example for update security considerations, which can be used in Autosar. Uptane is a standardized framework to deploy updates to the ECUs of an automotive vehicle. The goal of Uptane is to make the update process secure against potential attacks. Those include not only attacks on the channel between update server and client but also attacks by accessing the hardware in the vehicle as well as attacking the update repositories in the manufacturing chain. Uptane extends existing security frameworks to address characteristics of ECUs like a wide range of hardware capabilities and particularly ECUs with limited processing power for security checks. However, Uptane does not address safety, variant management or other issues of the update itself.

The eSync Alliance [130] is a consortium of automotive OEMs and suppliers which has developed an architecture for OTA data services between cloud-borne eSync Servers and vehicle-borne eSync Clients [131]. The eSync Server manages updates of ECU software as campaigns driven by policies. Policies manage the conditions for distribution of an update. The eSync Client receives the update and distributes it to the ECUs in the vehicle. The architecture also provides a back channel from the ECUs over the eSync Client back to the eSync Server. Services are provided for authentication, security and package management. The safety of the update itself is not considered.

Modular updates of central control functions are already provided for the autopilot function in the Tesla Model S. The update function is limited by the manufacturer to the specific vehicle architecture and there is no semantically sound approach to hardware abstraction that enables transfer to other platforms, systems and application domains.

In the USA, Ford decided to use Wind River Edge Sync technology for OTA updates. Other OEMs, such as Mercedes, Audi and Toyota have also already included OTA capabilities into their products [132]. However, most of current existing update features are limited to non-safety-critical systems with the exception of Tesla vehicles.

ROS [133] and its recently released beta version ROS2 [134] are middleware for robots and robot systems. ROS currently does not provide explicit support for dynamic updates. However, updates can be implemented "manually" on the basis of existing, non-real-time Linux services. ROS2 goes further in this respect because it is completely based on DDS (Data Distribution Service) and offers support for real-time features. Updates are not explicitly mentioned here either, but are technically easier to integrate than in ROS. ROS and ROS2 are

currently seen as an alternative to AUTOSAR Adaptive, especially in the automotive sector. The lightweight mechanisms underlying the ROS, but effective - especially with regard to functional interaction and integration - make it attractive as the basis for a reference architecture. However, many important aspects of a secure, updatable middleware remain open, such as the ability of the (verifiable) self-description of safety-relevant properties.

The project PASS develops a standardized runtime environment for the extension of vehicle functions by apps, as well as a corresponding development tool chain. In the implementation a separation of the apps is used by using virtualization techniques based on a MILS platform (Multiple Independent Levels of Security). The PASS project addresses apps for infotainment and communication that only interact with non-critical functions of vehicle electronics. However, updates of safety-critical functions are not covered.

Existing solutions from the automotive sector (e.g., escrypt [135], Infineon [136], as well as SWUpdate [137], libostree [138] or Mender [139] for embedded Linux systems) offer protection of updates by signing updates and concepts for key management. However, they usually replace the entire firmware of individual ECUs with a new version. Modular updates or changes to the software during operation, be it through rollback to a "known-good" configuration after the occurrence of errors or the migration of software across ECU boundaries, are not taken into account. Although the framework for securing updates in the automotive sector presented in 2016 by the Uptane [140] research project offers not only modularity along the supply chain by distributing the roles for compiling and signing updates, but also robustness against key compromise, it does not take into account security during operation, which can be impaired by attacks and malfunctions [141].

The EU-funded Horizon 2020 project TAPPS[3] has developed an open platform for CPS software, which consists of an "App Store" for security-critical and security-uncritical applications (or functions) as well as the software and hardware layers necessary for the assurance of security and real-time properties. Security-critical applications are developed in a model-driven process [142]. Based on a state machine specification, both program code and a model for checking properties expressed in linear temporal logic (LTL) are generated using symbolic model checks. In addition, the Worst-Case Execution Time (WCET) is determined. The information about the formal properties fulfilled by the application and the WCET are summarized together with a description of the required resources and authorizations (e.g., access to API functions of the middleware or hardware functions) in an "application container". This is connected to the program code and the manufacturer of the application by means of signatures, so that the origin and integrity of the application and its metadata can be verified by the system. Secure Shell (SSH) is used to secure the transmission and Keyed-Hash Message Authentication Code (HMAC) [143] is used to verify the integrity. When the software is installed, various checks are performed, including whether the number of tasks resulting from the installation is schedulable. The project focuses on ensuring the security and real-time properties at runtime. Virtualization techniques were used to run a Linux and a FreeRTOS system on the same hardware. The applications running on the Linux system are also isolated from each other by KVM. The communication between the partitions of different criticality is realized by middleware services, which enforce access restrictions. However, a complete and systematic threat analysis is not part of the project. The focus is on technologies that are prototypically demonstrated in the medical technology, automotive and Industry 4.0 domains.

The Step-Up!CPS [144] project developed cross domain software methods, technologies and processes for secure, modular CPS updates. To this end contracts are utilized to formally describe the behaviour of updated

[3] Towards Trusted Apps platforms for Open CPS, C. Prehofer, O. Horst, R. Dodi, A. Geven, G. Kornaros, 2016 International Workshop on Emerging Ideas and Trends in Engineering of Cyber-Physical Systems (EITEC)

components. Automated Virtual Integration Tests can be performed to verify the update in different phases of the update life cycle. A hardware virtualization and services in a middleware are used to develop updates for systems that use a wide range of different hardware configurations. The Step-Up!CPS process allows a continuous improvement of CPS by monitoring their operation. Based on the specification runtime monitors are generated, which allow a safe operation of the system by enabling mechanisms for fail operational concepts. The information collected via the monitoring is fed back to the developers to enable future improvement of the system. As the approach itself is domain agnostic the results are demonstrated prototypically in use cases from the automotive, maritime and industrial domain.

The Up2Date [145] project is addressing the challenges induced by Mixed-Criticality Cyber-Physical Systems (MCCPS) deployed in critical domains like automotive and railway starting to use Over The Air Software Updates (OTASU) for functionality improvement, bug fixing, and solving security vulnerabilities (among others). Applying OTASU on such systems entails several difficulties regarding safety, security and availability aspects. Additionally, computing performance needs are bigger and therefore complex hardware platforms based on multicore processors and accelerators are used in MCCPS. Such complex hardware platforms' software applications are subject to intricate dependences in their functional and non-functional behaviour. Bringing together these two trends, OTASU and complex hardware platforms, in MCCPS is the main motivation that inspired UP2DATE to work on a new software paradigm for SAfety and SEcurity (SASE) software updates for intelligent and resource intensive MCCPS, promoting a safety and security concept that builds around composability and modularity as main properties to enable a dynamic (post-deployment) validation of SASE properties.

4.5.2 Manufacturing Industry

In most cases, no update is performed in industrial plants because functioning plants are generally not touched. One reason for this is that companies often do not have in-house developers for their assets. In addition, they occasionally have online connections in the "Industrial IoT" context at the level of all Cloud IoT devices.

Necessary updates are made by the equipment manufacturer as part of maintenance contracts. This happens with proprietary solutions or in the programming environment of the ecu manufacturer. Siemens, on the other hand, has comparatively advanced software for this domain that can manage a complete fleet of equipment. An industry-wide classification does not exist. Currently, manufacturers seem to have no particular interest in a standardized deployment/update methodology, since industrial plants are still mainly customized systems. However, with the rise of cloud services and intelligent and connected plants, central management for whole product-lines including OTA as key for cost-efficient and flexible machine maintenance will be needed in the near future.

Currently, validation methods are usually not used. Only basic quantities are checked, such as "Does the update binary fit into the device memory?" or "Do the fieldbus messages (size / interval) not exceed the available bandwidth?"

4DIAC [146] is a development and deployment environment for distributed systems in industrial automation and smart grids according to IEC61499. 4DIAC includes the FORTE middleware, which can be used to run IEC61499 applications. The concepts on which IEC61499 is based offer a very good possibility for updates and redistribution of functions to different ECUs. So far, there is no support for checking updates either in the standard or in the existing implementations.

4.5.3 Maritime

Other than in the automotive domain, ship-side and on-shore bridge system architectures in the maritime domain face a different set of characteristics. Maritime systems are handled by trained professionals, often in teams, that underwent extensive education and training. Furthermore, maritime systems usually face longer time horizons for reaction times, are regularly subject to dead times in control and have to integrate various system dynamics such as masses, inertia, momentum and kinetic energy within a hydrodynamic environment and under six degrees of freedom (6DoF). Moreover, other than with the embedded systems widely used in the automotive domain, computational space and processing power is no limiting factor when it comes to safety-critical control systems. Additionally, the systems are equipped with environmental sensors exclusive to the maritime domain such as AIS, echo sounders or ECDIS and stay in close interaction with vessel traffic services (VTS) [147]. Therefore, the hard- and software products and consequentially the update processes in the maritime domain differ from those in the automotive and those used in production lines.

Traditionally, software maintenance processes on ship-side systems are performed infrequently. Due to the long life-cycles of the vessels in use, the shipboard systems are often equipped with legacy hardware & software systems. Reasons for the continuous use of outdated software products are the extensive certification processes maritime systems have to pass before being released. Thus, changes to a certified product would necessitate a (re-)certification of the altered artifact and the system of systems containing it. Therefore, updates to system components are applied seldom and if done, performed manually, on-site without appropriate supervision [148].

To align different stakeholders maintaining active (sub-)systems included in on- and off-shore systems, a Joint Working Group composed of the Comité International Radio-Maritime (CIRM) and the world's major shipping association BIMCO introduced an 'Industry Standard on Software Maintenance of Shipboard Equipment' in 2017 [149]. The standard addresses shipowners, bridge equipment manufacturers, service providers as well as system integrators and was accompanied by companies such as BP shipping, Emarat Maritime, Furono, Kongsberg Maritime, Maersk Line, MAN Diesel & Turbo, Radio Holland, or Sperry Marine. The maintenance process in the maritime domain covers power generation systems, propulsion systems, control and alarm systems, navigation and communication systems, steering and ship management systems, and support systems for water, fire or performance supervision.

Software maintenance processes can be differentiated between onboard maintenance, on-shore maintenance, and remote maintenance. With the industrial standard, the working group is aiming to reconcile the individual efforts of all stakeholders involved. For that purpose, a software maintenance process flow was introduced. The flow is subdivided into a sequence of four consecutive steps: Event initiation, planning, execution, and after service. The initiation of the software update process can be caused by preventive maintenance of the system, a corrective update that corrects faulty behavior, alignment of a software product with regulatory compliance, as well as a new feature or improvement of the component. The initiation phase should be followed by publishing all relevant meta-information about the update, including the category of the update, the changes made and instructions on how to apply it. The following execution step demands a diagnostic mechanism that shows how the updated system as component of an integrated system of systems is behaving and whether any negative interferences occur. In such a case, a rollback functionality is needed. Finally, in the after-service phase, service reports and on-board software logs need to be updated and an evaluation of the result should be communicated.

With future maritime systems being more and more interconnected, reliant on software functionality and increasingly autonomous, updates to individual system components become indispensable. New approaches making OTA updates possible in the maritime domain are currently being discussed, both from the

certification and classification perspective [150] as well as for the technological change within the maritime industry [151].

4.5.4 Industrial practices

4.5.4.1 Kumori Systems: Kumori platform

Kumori Platform is a Platform as a Service built on top of the open-source Kumori Kubernetes distribution. As a platform it simplifies and streamlines many aspects of building, deploying and managing a service application whose scalability is automatically managed by the platform itself based on a series of indications provided by the service integrator/developer. A more detailed description of Kumori Platform is given in Section 4.7.

When updating services, Kumori Platform seeks to avoid or minimize service down-time, by performing ordered updates in a subset of instances of each affected microservice. Instances to be replaced are given some time, a grace period, to finish handling any ongoing requests.

All Kumori service model elements (components, services applications, deployment configurations) are versioned using semantic versioning, and communication between microservices (the communications topology) allows providing protocol information. Future versions, will enforce connected microservices are compatible using metadata such as the protocol or protocol version.

Multiple versions of a microservice can be deployed as part of a service application, so that a new version can be tested by directing only a portion of the traffic to it. This is accomplished by defining VersionSets in the service application specification.

Kumori Platform keeps track of the update history of each service, and provides a simple mechanism to navigate between previously deployed versions.

4.6 Architectures for distributed safety-critical solutions

In the following section, we discuss architectures with a focus on distributed safety-critical solutions. We start by discussing reference architectures for safety-critical cyber-physical systems from the automotive, avionics, manufacturing and edge computing domain. Afterwards, we discuss general styles, patterns and tactics from the literature hat are applicable for distributed cyber-physical systems and analyse them with respect to their impact on safety. Finally, we discuss safety pattern that are a means to improve the safety of a particular function.

Discussing these concepts serves different purposes within this document: from the reference architectures we can learn about practices in related fields that serve similar (but not identical) problems that are tackled by the TRANSACT reference architecture. The architecture styles like the layered architecture provide general rules for the shaping of systems or sub-systems. They can be used as a blueprint to organize e.g. the device, edge or cloud tier or groups of services. Finally, the safety pattern are closer to the design level and can be utilized when designing an individual (safety-critical) function.

4.6.1 Reference Architectures for Distributed Safety-Critical Systems

4.6.1.1 Automotive

In recent years, the role and importance of software in the automotive domain has changed dramatically [152]. In the past, safety-critical functions were related to pure mechanical parts and did not involve any software or electronics. However, this has changed dramatically. More and more safety-related functionalities are implemented in software or are only possible enabled by software (such as automated driving functions) [153].

Furthermore, vehicles get connected with other vehicles and digital services operated by digital platforms. Today, a vehicle is no longer an isolated system. Instead, a vehicle is connected to a multitude of other systems. Modern cars offer connections to smartphones using Android Auto or Apple Carplay. Furthermore, vehicle-to-vehicle or more generically vehicle-to-X connections can be used (e.g., for intersection assistants, which enable cars to coordinate the priority at intersections among themselves). Applications for these vehicle-to-X communications are numerous and practitioners have only started to investigate the potentials of these technologies. In the past, the software of a vehicle could be seen as an isolated entity, interacting with its environment only indirectly via the laws of physics. However, with vehicle-to-X connections, the software interacts directly with the environment. This makes a vehicle one system in a system of systems [153].

4.6.1.1.1 Evolution of E/E Architectures

This change in the importance of software and electronics can also be observed in the evolution of the electric and electronics (E/E) architecture of cars that started with a handful in the late 1970s and 1980s. The 1990es were dominated by an increase of the number of ECUs (electronic control units) from a few dozens to 100s of ECUs up to a point where managing the complexity of the overall system architecture was no longer feasible. This period was followed by a consolidation of the number of ECUs, starting with domain-based architectures [154] that organize the overall vehicle bus around more powerful (but still classical embedded) domain controllers. The last years have seen a trend towards even more consolidated ECUs. An example is Audi's Zfa (Zentrales Fahrer-Assistenzsystem), which includes a main processor, a graphic processing unit, and a safety system – all on one integrated circuit. Those highly integrated circuits offer computational power comparable to (or exceeding) the computational power of modern business PCs.

4.6.1.1.2 AUTOSAR Classic

Besides these changes on the E/E-architecture, we see also a big change in software architecture: In the past, automotive development was ECU-centric, i.e., suppliers developed the software for an ECU either from scratch or on top of manufacturer-specific operating systems. Recent years have seen a change from these manufacturer-specific architectures towards AUTOSAR [160]. The main goal of AUTOSAR is to enable the development of hardware-independent application functions (AUTOSAR application software) that can be executed in any AUTOSAR environment. AUTOSAR offers a basic layer that abstracts from the underlying hardware like microcontrollers and CPUs. On top of this basic layer, the AUTOSAR RTE implements a common API for the communication between application software components, regardless of them being deployed on the same ECU or on different ECUs.

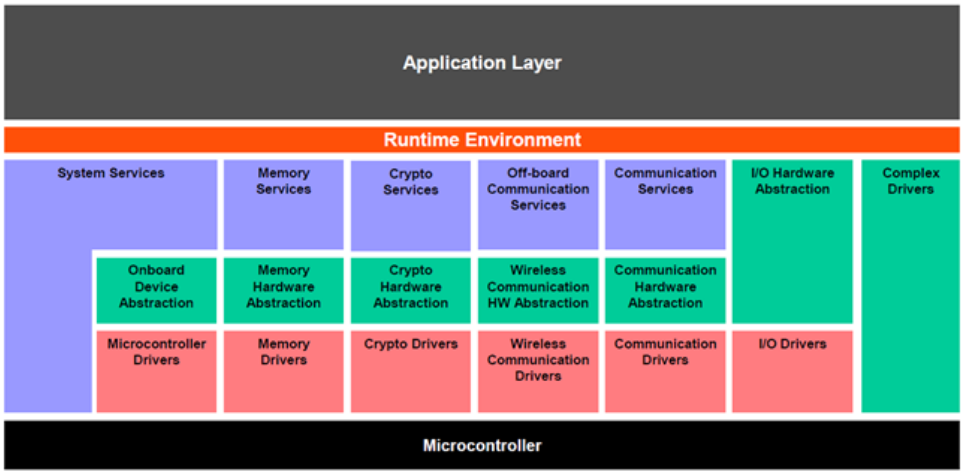


Figure 13 AUTOSAR Classic Software Stack

4.6.1.1.3 AUTOSAR Adaptive

The next generation of AUTOSAR, which is currently under development, is called "AUTOSAR adaptive" [155]. It addresses the needs of modern cars like vehicle-to-X connections. Besides other things, AUTOSAR adaptive offers means for service-oriented architectures, which leads to a change in APIs from rather simple C-based APIs in AUTOSAR classic towards C++-based APIs describing services. Thus, while AUTOSAR fosters reuse and hence generally leads to a reduction of development effort, this comes at the price of more complex APIs and more complex toolchains, for example to synchronize the interaction between application functions and the AUTOSAR basic layer.

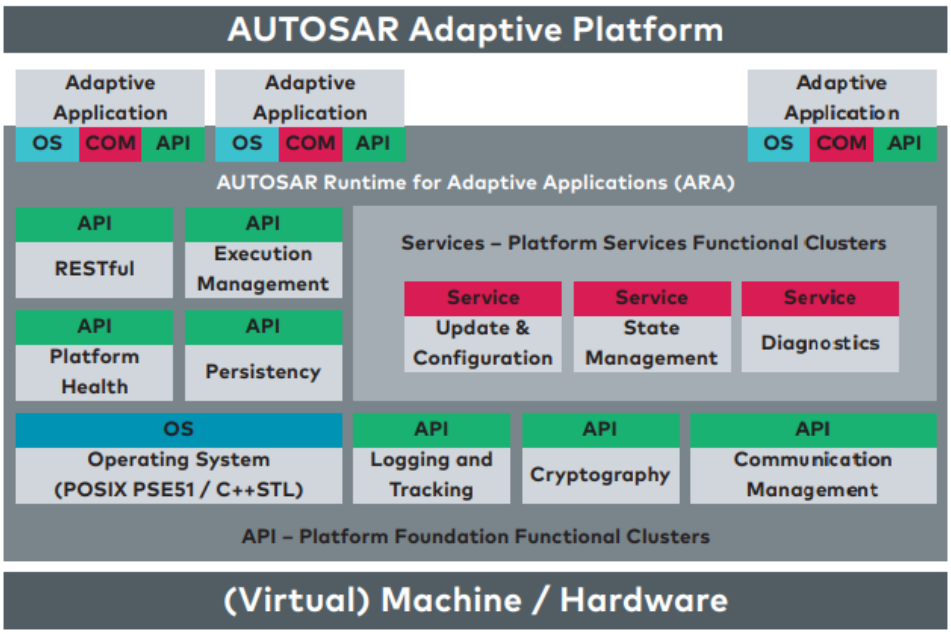


Figure 14 AUTOSAR Adaptive Software Stack

4.6.1.2 Avionics

Avionics is a word created from aviation and electronics. It is used as a synonymous for the electronic systems used by aircraft, spacecraft and satellites. Avionics includes a wide range of systems, such as displays,

communication and navigation as well as any other electronic equipment used by aircrafts. In the following paragraph, we discuss several reference architectures from the automotive domain. In the presentation, we follow the respective section on avionics reference architectures from [152].

Since the introduction of the first avionics systems in the mid-'50s special attention is paid to the quality of these systems as a failure in one of these systems can lead to fatal consequences. Consequently, there is an especially strong focus on Reliability, Availability, Maintainability and Safety (RAMS). Previously, avionics systems were always built according to the Federated Architecture model. In this model, each function has its own subsystem with its own sensors, actuators and computing power. This gives each function a guaranteed processing time and deterministic access to input/output (I/O) resources, ensuring controlled latency and predictable jitter. [156] [157] However, in recent decades there have been increasing demands for functionality, performance and safety. The increase in aircraft functions has pushed the federated architecture to its limits. [158]

4.6.1.2.1 The Integrated Modular Avionics Model

To meet the demands of modern avionics, the Integrated Modular Avionics (IMA) architecture [159] model was designed, which is now considered SoA in this field. With the IMA model, several functions are integrated in the same execution platform. The IMA model focuses on modularity and standardization. For intercommunication, it relies on multiplexed high-speed networks to replace point-to-point computer communication [160].
By their nature, IMA systems allow extensive reusability of software and hardware components. The general structure is based on a central Core Module that is able to execute several software applications in a way that each one is unaware and unaffected by another. It communicates with sensors, actuators and display units through high-speed multiplex networks. The Core Module is called avionics computer. It provides the computing power for all functions, such as Flight Control System (FCS), Inertial Reference System (IRS), Vehicle Management System (VMS) and Terrain Avoidance and Warning System (TAWS), among others.

In order to guarantee a flawless operation of the functions and due to intrinsic resources sharing, the reference architecture also considers operating system level and even makes recommendations for the underlying hardware. E.g., among others, the following requirements needs to be satisfied by the underlying hardware:

1. The processor is able to restrict the access to different memory spaces and IO ports e.g., by a Memory Management Unit
2. The processor has access to time resources to implement time services
3. The processor provides a mechanism to transfer the control flow back to the RTOS (in case a Partition performs an invalid operation)

One key characteristic of an IMA system is the robust partitioning. All avionic functions must be able to properly run in a concurrent manner. To provide the necessary isolation between the aircraft functions that access shared resources the term partitioning is defined in two dimensions: temporal and spatial partitioning. Temporal partitioning is the assurance of a function's access to a prescribed set of hardware resources for a defined period of time and the consistency of its execution order during each execution cycle. Spatial partitioning is the protection of a function's program, data and I/O resources so that any persistent storage locations are writable only by one function. Besides the aforementioned requirements on hardware, robust partitioning as demanded by IMA system needs to be supported by an appropriate Real-time Operating System (RTOS) which is discussed in the following subsection in greater depth.

4.6.1.2.1.1 The Avionics Application Software Standard Interface

The ARINC 653 Specification "Avionics Application Software Standard Interface" [161] defines the interface requirements between avionics software applications and the underlying RTOS. The core concept is called APplication/EXecutive (APEX) interface [162], which is an execution environment that allows creating platform independent software that can abstract the underlaying RTOS and hardware. Furthermore, it specifies the structure of all the data that is exchanged statically (via configuration tables) or dynamically (via services) between the parties.

As indicated in Figure 15, the software as part of an IMA system can be broken down into *Partitions*, *System-Specific Functions* and the *Operating System* (*OS*) *Kernel*. Partitions are further subdivided into *Application-* and *System-* and *Pseudo-Partitions*. *Application Partitions* are containing the avionics functions and are restricted to APEX system calls

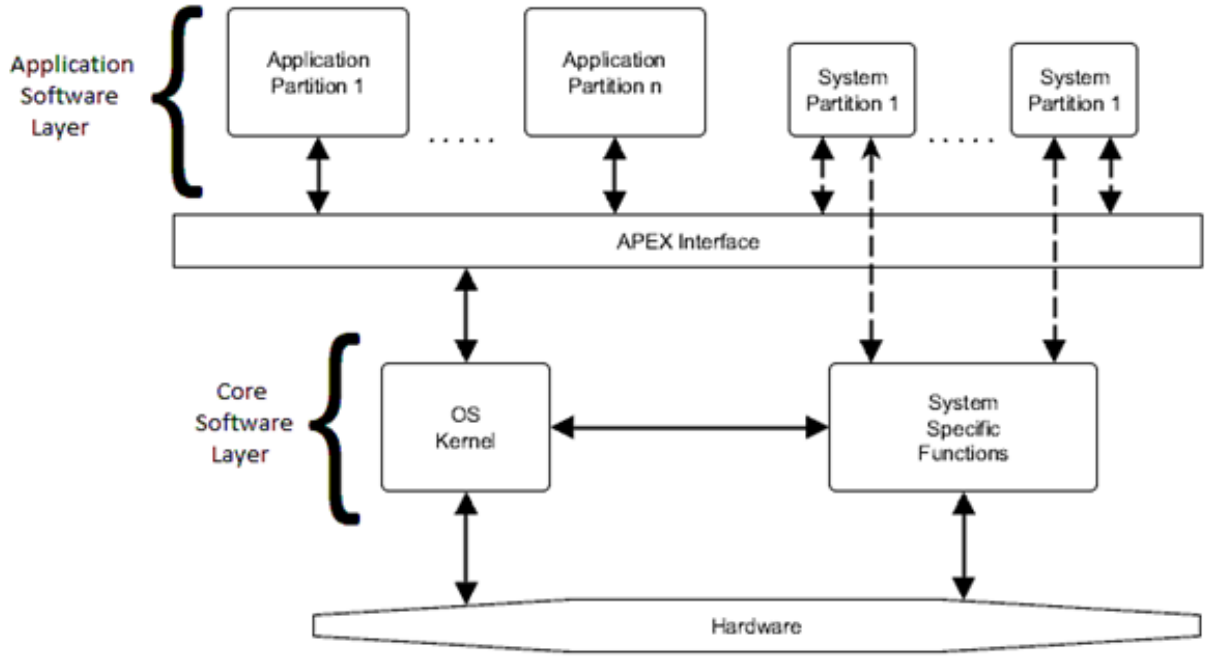


Figure 15: Core Module Parts

System Partitions must access services that are not provided by the APEX interface. However, they are still constrained by the same temporal and spatial partitioning mechanisms. They are optional and usually associated with hardware dependent functions.

Pseudo-Partitions are any devices, subsystems or other software applications which can communicate with any hosted partition but is external to the module. From a module's view any partition residing another Module is considered a Pseudo-Partition.

System-Specific Functions include hardware interfaces such as device drivers, software upload, maintenance data download and software debug ports, among others.

The *OS Kernel* implements the APEX interface and enforces the partitions' processing time, memory access rights and usage of shared resources.

The RTOS manages the partition processes and their intra-partition communication facilities. At module-level it manages the partition resources and the communication within and across module boundaries. To achieve this, the application requirements in terms of memory access, processor utilization, communication ports needs to be defined for each software during the system integration process by standardized XML tables [163]. When running, the RTOS has four operating modes. During power-up, the RTOS is in the *Cold Start* mode, initializing all partitions according to the XML tables. After the boot sequence it enters the *Normal* operating mode where it remains until power-off or an error handling action forces it to move to the *Idle* mode. Each partition remains in this mode until it has been (re-)initialized. In addition to the states already mentioned, there is also the *Warm Start* state, in which partition initialization also is carried out, but in the event that some of the partitions are already initialized and therefore not all need to be initialized.

4.6.1.3 Manufacturing Industry

4.6.1.3.1 State of the Art: The Automation Pyramid

The dominating architecture in manufacturing seen today is the "automation pyramid" reference architecture as defined by IEC 62264 [164] that is shown in Figure 16.

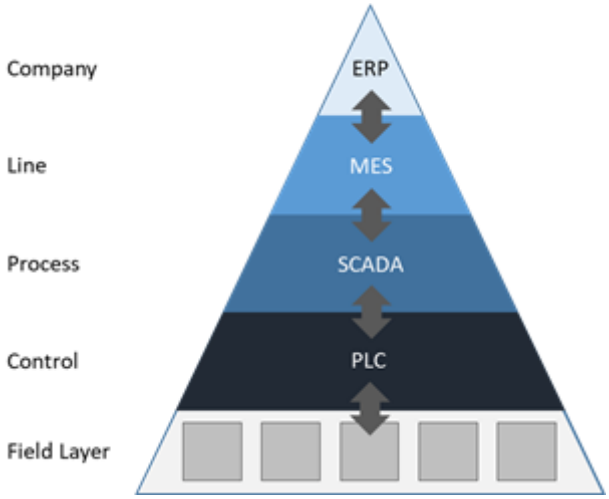


Figure 16: The automation pyramid

This architecture consists of separate layers. The lowest level is the field layer that includes the devices that interact directly with the industrial process like sensors and actuators. Usually, the control of the process is executed by PLCs that perform the specific steps in predetermined cyclic programs. Supervisory control and data acquisition (SCADA) realize the distributed control of several PLCs. Manufacturing Execution Systems (MES) manage the overall production by controlling high-level manufacturing steps. Enterprise Resource Planning (ERP) systems manage resources, for example the number of items in stock. Cross layer interaction in the automation pyramid is difficult, as every layer can only access information that is provided by lower layers. Changes therefore require modifications in all layers. E.g., if some process parameters need to be changed, the code on the PLC needs to change, which may affect cycle times and therefore real-time constraints of the devices. Manufacturing processes usually execute production steps in a fixed order and hence PLCs assume a fixed cycle time of the predecessor steps. In consequence, a change of a single PLC may produce side effects in other PLCs and also in other Layers of the automation pyramid.

4.6.1.3.2 Industry 4.0

Industry 4.0 is a term coined in Europe, notably in Germany, that is associated with the fourth industrial revolution. In the US, the term industrial internet of things is more common. In contrast to earlier revolutions, this is not about the introduction of a single new technology, but about a much more ground-breaking change, namely the end-to-end digitization of manufacturing processes. The main driver behind Industry 4.0 is the flexible production: to adopt quickly to changing customer demands to produce customized goods with small lot sizes. Industry 4.0 is mainly driven by innovations in software systems for industrial plants and by the transfer and transformation of knowledge from the IT sector into production.

According to [161], several reference architectures applicable for industry 4.0 has been proposed.

RAMI 4.0 [162] is a domain-specific architecture that is driven mainly be the German government's industry 4.0 initiative led by the Association of German Engineers (VDI) and German Electrical and Electronic Manufacturer's Association (ZVEI). It establishes. IVRA [163] (Industrial Value Chain Reference Architecture) is driven by the Japanese Industrial Value Chain Initiative. Both RAMI4.0 give a conceptual framework for the design of industry 4.0 system architectures by providing the necessary concepts and an overall organization of smart factories by describing all crucial components of a smart factory. In contrast to that, SITAM (Stuttgart IT-Architecture for Manufacturing) [165] is an academic, multi-layered architecture that focuses on integration and interoperability in smart factories by presenting three middleware for Integration, Mobility and Analytics. It contains five buses to connect several systems along the whole product lifecycle from product engineering to support, as well as physical devices and IT Systems. Similarly, LASFA (LAsim Smart Factory) [166] also focuses on integration. It is based on the layers, views and concept of the RAMI4.0 framework, but at a lower level of abstraction then RAMI4.0. The IIRA [167] (Industrial Internet Reference Architecture) that has been developed by the IIC (Industrial Internet Consortium) is a domain-independent architecture that includes manufacturing, healthcare, smart city and others.

Currently, all mentioned reference architectures are on a very high level of detail and are not detailed enough to be easily used in industry projects. However, they all share the concepts of digital twins and cyber-physical systems integrating IT and physical devices. Realization of these concepts requires a complete rethinking of the architecture and infrastructure of industrial plants. Instead of the layered architecture of the automation pyramid, Industry 4.0 propagates peer-to-peer communication in order to enable device-to-device communication and the dominating architecture style is a service-based architecture. The basic idea of service-oriented architectures is that each component (e.g. a factory device) provides services to the other components through a communication protocol over a network. Note that a service can be any functionality within a factory: typical services are pick or place services offered by industry robots, other machines might offer drilling, milling or painting services.

Notable Realizations of the Industry 4.0 concepts in Research Projects are the Basys4.0 [168] Project in Germany and the EU-funded project Arrowhead. Eclipse BaSyx[4], the reference implementation of Basys 4.0, builds upon RAMI4.0 and leverages the concept of the Asset Administration Shell, realizing the digital twin concept, to specify data models for a service-based production. It is an open-source communication middleware realizing a virtual automation bus that integrates older technologies like Modbus with SoA communication protocols like OPC-UA or MQTT. Besides integrating different communication protocols, it also offers means to realize a service-oriented architecture like service discovery and device discovery. The Arrowhead [169] project also provides a middleware that can be used to realize the digital twin concept.

[4] https://www.eclipse.org/basyx/

Eclipse Arrowhead[5] is the open source implementation, providing a software framework to realize digital twins of assets (e.g. physical devices) and providing their functionality as services. Similar to Eclipse BaSyx, a major focus is also on interoperability. Therefore, Eclipse Arrowhead provides adapters for typical communication protocols like OPC-UA or MQTT.

4.6.2 Architecture Strategies, Styles and Pattern applicable for Distributed Safety-Critical Systems

4.6.2.1 Architecture Styles for Safety-Critical Systems

There exists a lot of (software) architecture styles that are applicable for safety-critical systems and there is a multitude of literature that discusses them in depth like the seminal works of Buschmann et al. on pattern-oriented software-architecture [170] [171] [172] [173]. However, the styles are most of the time discussed with an information-system background and therefore, the applicability in the context of safety-critical systems is not clear. Therefore, we discuss in this paragraph some classical architecture styles that can also be applied for cyber-physical systems together with a preliminary analysis about their impact on safety.

4.6.2.1.1 Layered Style

Problem / Context

The layered style [170] structures architectures and designs that can be decomposed into groups of subcomponents in which each group of subcomponents is at a particular level of abstraction.

Solution

To that end, the software architecture is separated into ordered layers, wherein a component in one layer may obtain services only from a layer below it. Each layer provides services through an interface that is accessed by components residing within the layer directly above it. Often this strict layering is relaxed such that components from layer 3, for instance, may also use services from layer 1.

Impact on Safety

The dependencies between layers can be exploited to improve safety. In particular when there is strict layering, the increase of safety in a base layer trickles through the layers that depend on it.

On the other hand, however, layering can complicate the design and implementation of safety mechanisms. Safety mechanisms, e.g. for failure detection and failure management, tend to crosscut across the layers. Layering can also complicate analysis. The layering overhead can increase latencies and timing uncertainties. It could also be more difficult to handle mixed criticality when there are a large number of dependencies inside a layer.

4.6.2.1.2 Microkernel Style

Problem / Context

The microkernel architectural style [170] can be used when several applications use core functionalities that are expected to remain stable over changing requirements. It offers flexibility when new functionality is added to the core functionality.

Solution

[5] https://projects.eclipse.org/projects/iot.arrowhead

This style separates the microkernel, holding a minimal core of functions, from extended functionality. As a result, the microkernel is a relatively small code base, which can be developed for the assurance of higher integrity levels. The microkernel uses internal services (IS)that can only be accessed by the microkernel. These services can be modified to adapt to changing environments, such as a change of hardware platforms. The external services (ES) provide application-specific functionality. Client applications may only use these services. The microkernel can support software product lines provided they are properly accounted for during the microkernel's development.

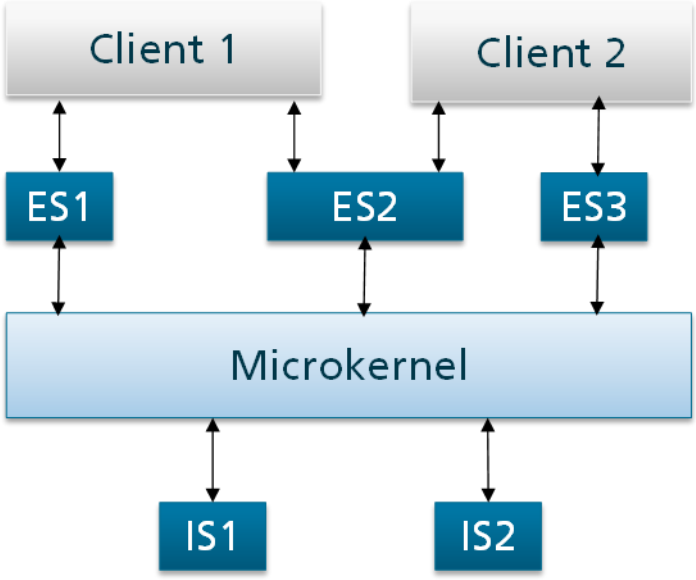


Figure 17: The Microkernel Pattern

Impact on Safety

Compared to monolithic approaches, the microkernel itself is a minimum code base that facilitates qualification. And once the microkernel is certified, this certification can be (partially) reused. It also facilitates expansion to infrastructural safety mechanisms, in particular when the kernel is developed for safety (e.g., safety kernel).

The main concern is that the microkernel is a potential single point of failure and could also be the source of common cause failures.

4.6.2.1.3 Pipe & Filter Style

Problem / Context

The Pipes & Filters pattern [170] provides a structure for systems that process a stream of data. It is assumed that the processing of data can be decomposed into several processing steps.

Solution

This pattern encapsulates each processing step as a filter component. To that end, a filter must be independent of the other services, which means that it is free from side effects. Data is passed through pipes between adjacent filters.

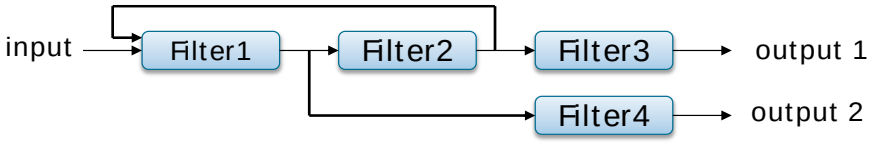


Figure 18: The Pipes & Filters Pattern

Impact on Safety

The absence of side effects in the filters facilitates data and control-flow analysis, and subsequently also failure propagation analysis. The pattern also fosters reuse of filters, especially when they have become trusted through qualification. Furthermore, the pattern follows a structure similar to many physical models, and as such the corresponding sanity checks can be easily incorporated. It also facilitates the addition of safety mechanisms inside a filter or as a separate filter.

This pattern does, however, lack a global failure state. Take Figure 18 as an example. The failure state of the speed sensor would need to be propagated across filters until the responsible filter is reached that handles the failure. This increases the complexity of the design and essentially negates the absence of side effects in the filters and the associated benefits.

4.6.2.1.4 Event Bus Style

Problem / Context

The Event Bus pattern [174] can be used in a distributed system where components have to communicate with each other. Direct communication links between all components hamper maintainability, since this makes components depend on each other so that changes in one component may lead to ripple effects, i.e., changes of many other components.

Solution

The Event Bus pattern introduces an event bus. All components communicate through this event bus by emitting events and snooping events on this bus.

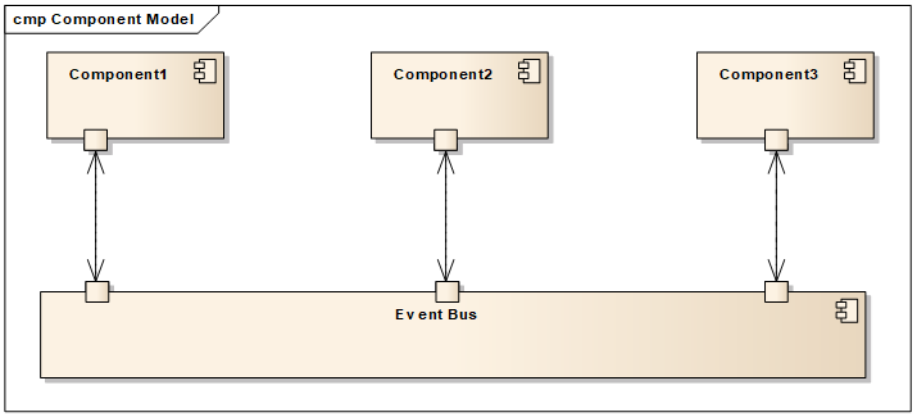


Figure 19: The Event Bus Pattern

Impact on Safety

Safety mechanisms can be hooked into this pattern by adding failure containment mechanisms that respond to failure events that are sent out by the failure detection mechanism. It thus decouples those two mechanisms from each other. The event bus can also be used to manage redundancies using component activation and deactivation events.

The event bus pattern makes the execution more dynamic, however, which in turn complicates safety analysis. The safety relevance of components might propagate and conflict unless precautions are taken. The bus itself is a potential single point of failure and also a potential source of common cause failures.

4.6.2.2 Safety Tactics and Patterns

Safety generally relates to conditions of being protected against hazards. This can be addressed from various dimensions such as environmental safety, food safety, and (technical) functional safety. Functional safety addresses the specific hazards that are caused by malfunctioning system behavior, i.e., system failures. This is also the scope of, e.g., the ISO 26262 (Road Vehicles – Functional Safety). We furthermore define and distinguish the terms fault, error and failure according to the nomenclature of ISO 26262:

- A *fault* is an abnormal condition that can cause an element or an item to fail.
- An *error* is a discrepancy between a computed, observed or measured value or condition, and the true, specified or theoretically correct value or condition.
- A *failure* is the termination of the ability of an element to perform a function as required.

Note that an event can be both a fault and a failure depending on the perspective it is reasoned from. For example, a subsystem failure (e.g., battery failure) can be the fault of a system-level failure (e.g., system power loss). This kind of reasoning is typical for fault propagations.

There is a broad diversity of safety tactics, principles, patterns, and techniques discussed in the literature. This demonstrates that there is no single overall solution that improves safety while keeping complexity and costs in check. Typically, a carefully deliberated selection of strategies, principles, patterns, and techniques is combined to meet the system and safety requirements as well as the overall project constraints. In this subsection, we give a summary of strategies and patterns discussed in the literature and refer the interested reader to the original sources for more information.

4.6.2.2.1 Safety Tactics [175]

Failure Avoidance

The failure avoidance tactics aims at increasing safety by increasing the reliability of the nominal system. The principle is to avoid or remove the fault that underlies the failure such that it cannot manifest during runtime. This tactic is only applicable during the early development lifecycle and is mandatory practice for safety-relevant systems. The key advantage of this tactics is that it typically does not increase the complexity of the architecture (or the design). It is, however, considered to be expensive and time-consuming, especially when it is applied to systems with high integrity levels. Furthermore, it is only applicable to systematic failures.

Failure Detection

The failure detection tactics aims at indicating the occurrence of a failure. However, it is typically insufficient as a fault tolerance principle on its own and, as such, delegates the responsibility for further handling of the failure to the user or to other components/systems. As such, this tactic is often applied as part of the decoupling of the fault tolerance mechanism into a two-part system comprising a detection mechanism and a containment mechanism. This decoupling enables modularization of the fault tolerance mechanisms.

Failure Containment

The failure containment tactics increases safety by adding mechanisms for tolerating failures during runtime. This is in contrast to failure avoidance, where failures no longer occur during runtime. It becomes applicable after failure detection. This tactic can stop failures from propagating and can ensure that safety goals are met. This typically comes at increased costs, decreased performance, and increased architecture/design complexity.

4.6.2.2.2 Safety Pattern [176]

Protected Single Channel

The protected single channel pattern is a lightweight means for increasing the safety of a sensor-actuator channel by adding checks and optionally corrective actions in the processing stages of the channel.

Homogeneous Redundancy

The homogeneous redundancy pattern provides alternative processing channels in case a failure occurs in the main processing channel. It is characteristic for this pattern that the alternative channels are exact duplicates of the main channel.

Triple Modular Redundancy Pattern

The problem addressed by the Triple Modular Redundancy (TMR) Pattern is to provide protection against random faults (failures). To that end, the TMR pattern provides an odd number of channels operating in parallel. The results (e.g. resulting actuation signals) are compared and if the results differ in the three channels, a majority-vote is applied to determine the final result.

Multi-Channel Voting

In the multi-channel voting pattern, multiple active channels are used to compute outputs control signals. A voting mechanism such as majority voting is then used to detect or tolerate failures in the channels. It is a generalisation of the triple modular redundancy pattern

Heterogeneous Redundancy

The heterogeneous redundancy pattern provides alternative processing channels in case a failure occurs in the main processing channel. Contrary to homogeneous redundancy, the alternatives channels are distinctly different from the main channel and thus add protection from systematic failures

Monitor-Actuator

In the monitor-actuator pattern, an additional independent sensor and an independent monitoring channel monitoring the actuator for indications of a failure are added.

Sanity Check

The sanity check pattern is structurally the same as the monitor-actuator pattern. The key difference lies in the precision. The sanity check pattern is realized using less precise (and typically cheaper) sensors, as well as a simpler (and typically cheaper) monitoring channel.

Watchdog

A watchdog tracks whether internal processing aspects (e.g., memory use, execution time) proceed as expected. This is in contrast to the sanity check pattern, which tracks the actual output using an additional sensor.

Safety Executive

In the safety executive pattern, a safety executive component is added that responds to failures via a coordinated (complex) action sequence across multiple channels. It is typically an umbrella safety mechanism for handling fail-safe modes at the system level.

4.7 Microservice-based architectures and configurations

Deploying microservice-based architectures leveraging the cloud-edge continuum benefits system quality attributes such as latency and performance [177]. While running tasks on edge nodes improves response latency and bandwidth consumption, the cloud layer provides resource availability to perform compute-intensive tasks. However, the design and deployment are becoming very challenging due to the heterogeneity of computing infrastructures and communication topologies in such cloud-edge architectures. Even more challenging is the management of microservices at runtime to ensure optimal performance while facing changing operational conditions. In addition, most of the state of the art is not fully aware of the specification of microservices-based architectures for cloud-edge as well as runtime configuration.

Tools and orchestrators such as Kubernetes and Docker swarm enable the modeling of the architecture and deployment of container-based microservices applications. Typically, by specifying a YAML file the system architecture is configured including deployments, services, network protocols, and basic security components. However, these orchestrators are focused on the cloud layer where computational resources are ideally unlimited (the opposite of the edge layer). Lightweight Kubernetes distributions such as K3S and KubeEdge were released for container orchestration on the edge, but these tools do not enable complex runtime system adaptation configuration.

Several approaches have leveraged Model Driven Engineering (MDE) to support systems engineering based on microservices architectures. Some studies propose domain-specific languages (DSLs) or metamodels for system modeling and specification (although these are not edge-focused). For example, in [178], Guidi et al. present Jolie, a service-oriented language for the implementation of service-based software systems. Jolie separates concerns between behavior and deployment of the system architecture. Four main concepts can be modeled with Jolie: Interfaces, Ports, Workflows, and Processes. For this imperative language, standard constructs such as assignments, conditionals, and loops are combined with constructs dealing with distribution, communication, and services. Petrasch [179] proposes three models for specifying the system's microservices architecture and messaging between them, using Enterprise Integration Patterns (EIP). The domain class model is defined using UML class diagram. The specification of the microservices is through a UML profile and stereotypes to describe the properties of microservices. Although this approach enables modeling publish/subscribe communications, it does not address the modeling of behavioral specifications at runtime. In addition, these two approaches are dedicated to system modeling, but not to implementation or code generation for those purposes.

Other studies propose more complete MDE-based approaches, including not only the DSL, but also model transformation and code generation. For instance, Rademacher et al. [180] propose two approaches to support the engineering of distributed software systems based on microservice architecture (MSA): (1) AjiL, a graphical modeling approach involving an Eclipse-based editor to design architecture topology diagrams of the MSA and detailed diagrams of the integral components of each service. AjiL integrates a code generator to produce code for service implementation (Java and Spring Cloud as a target technologies); and (2) a textual modeling approach to capture relevant domain concepts, to construct microservice models, and to specify service deployments. Terzic et al. [181] propose MicroBuilder, a model-driven software tool for the REST microservice architecture specification and program code generation. MicroBuilder is composed of a MicroDSL for specification of microservice architecture, and micro-generator to produce executable program

code for Java and the Spring Cloud framework. However, it is an approach specifically focused on REST applications. In [182], El Khalyly et al. propose a Platform Independent Model (PIM) to build an IoT system based on microservices and supported by tools DevOps. The IoT system architecture and deployment model can be specified according to a proposed metamodel that merges relevant concepts from the IoT, Ansible, Docker, Kubernetes, and Microservices domains. Some restrictions must be met for deployment. For example, each microservice is deployed as a Docker pod on a node of a Kubernetes cluster created by Ansible playbooks. In [183], Yussupov et al. propose MICO (Microservices Composition Approach), an approach that enables modeling the architecture (pipe and filter style), integration, and deployment of systems based on microservices. MICO generates source code and the necessary artifacts to deploy and run the system. Although this approach enables the modeling of publish-subscribe and point-to-point communication patterns, it is focused exclusively on the use of the pipe and filter architectural style.

Finally, frameworks, tools, and architectures have been proposed to address runtime self-adaptations of microservices-based systems. Although these studies do not involve system modeling and deployment, they focus on ensuring QoS despite dynamic conditions. For example, De Sanctis et al. [184] propose a self-adaptive architecture for microservice based IoT systems. This architecture consists of three layers (edge, fog, and cloud). The cloud layer executes learning models to predict, choose adaptation strategies, and applying adaptations to the fog and edge infrastructure including scaling microservices, auto-rollback, and restarting microservices. Although this architecture provides runtime system self-adaptation to ensure QoS levels, it does not address microservices-based system architecture modeling and deployment. De Guzmán et al. [185] propose PipeKit, a container-based application orchestrator for microservices-based applications. This tool extends Docker Compose and enables (with a YAML file) the specification of the microservices to be deployed, the Docker image, ports, environment variables, and a set of conditions that must be witnessed before the service is started. However, Pipekit is not optimized for edge node system deployments. In [186], Sampaio et al. propose REMaP (Runtime Microservices Placement), a platform for runtime reconfiguration of microservices placement based on their communication affinities and resource usage. REMaP follows the MAPEK cycle to do the analysis (historical and runtime data) and runtime adaptation of the system. Although REMaP optimizes the placement of microservices at runtime, the deployment of the system, modeling of architectural styles, and specification of the interaction between microservices is out of scope.

To sum up, a better support for defining multi-layered architectures based on microservices, its communication topologies, microservices interaction, and runtime configuration is necessary. Most of the studies are focused on the cloud context ignoring the complexity of managing microservices at the edge and fog layers. Not only is the heterogeneity of technologies and communication protocols challenging, but also the resource limitations of edge and fog nodes are factors that can impact system performance.

4.7.1 Industrial practices

4.7.1.1 Kumori Systems

4.7.1.1.1 Kumori Platform

Kumori Platform is a Platform as a Service built on top of the open-source Kumori Kubernetes distribution. As a platform it simplifies and streamlines many aspects of building, deploying and managing a service application whose scalability is automatically managed by the platform itself based on a series of indications provided by the service integrator/developer.

Kumori Platform Introduces a series of basic easy to understand concepts, which we will present in what follows. These concepts are supported through a set of mechanisms and tools that permit developers and integrators to easily place in production nontrivial services.

4.7.1.1.2 Kumori Service Model

Modern Services built for the cloud are based on the concept of separating functionalities into many "independent" units referred to as microservices. The intent is multiple. On the one hand we want to decouple functionalities. This is the well-established approach to separate functional concerns.

On the other hand, there is the realization that different tasks may need different amounts of resources, and, thus, different scaling strategies. It follows, thus, that it is best to separate functions into their own autonomous Microservices, so that we also separate operational concerns.

In Kumori Platform Microservices are ultimately implemented as Components. A Component is just a program that can be autonomously ran within its own environment. When ran, such a program becomes a microservice. Moreover, in a microservices environment, running a Component may actually result in the execution of several autonomous instances of that program. This makes sense when we need the aggregated power of those instances to carry out a heavy computational task.

Another scenario when such multiplicity of executions makes sense is when we need the aggregated storage/bandwidth resources of such a collection of instances to actually satisfy the demand of storage made by other parts of a service.

Yet another scenario when multiple instances of a program are needed is when we need to support failures while providing continuity of the service. Multiple instances, when well thought out and properly placed, make it possible for an instance of a component to take over the job of another, failed, instance, without disruption to the service.

Services are thus formed by running multiple microservices within the constraints of the Service deployment architecture, describing the intercommunication patterns among those services.

When a service is first activated, one of the problems that needs to be solved is ensuring each one of the microservices can find those microservices with whom it needs to communicate. This is often referred to as service discovery.

If a service was static, i.e., never suffered changes in its configuration, like increasing/decreasing the number of instances a microservice has, or never suffering failures requiring launching new instances of microservices, then, we could provide each instance of each microservice with the IP address of those instances needed to carry out communications.

The above scenario is, however, unrealistic in current cloud computing conditions, where loads are dynamic and services must adapt dynamically to all sorts of changes.

To facilitate managing changes in the structure of a running service, Kumori Platform introduces the concept of a Service Application. In a nutshell, a service application brings together the set of components that will form a service, together with the interconnections that must be established so that their instances can discover each other.

In addition, a service application also declares the set of configuration parameters it expects, as well as the set of resources it needs to be provided with.

Finally, services are the result of running deployment configurations of service applications. Kumori Platform provides very simple mechanisms for service discovery, allowing all deployed instances of microservices to access other microservice instances deployed within the same service.

Kumori Platform brings these important functionalities to the table:

- A service model and formalism to express deployable services on the platform

- A Kubernetes distribution, composed of open-source modules, tested to work harmoniously.
- A cluster management framework, enabling cluster operations to proceed effortlessly and smoothly
- A service management framework, permitting service owners to manage their deployments, and their life cycle.
- An automation facility, capable of making scaling decisions, automating adaptation to changing loads

Kumori Platform uses a declarative specification of the platform configuration, and provides tools for:

- installing the platform
- managing the platform lifecycle (adding/removing nodes)
- monitoring the platform status and the services deployed on top of it

4.8 Big Data infrastructure and machine learning components

Over the years, data have become widely available and many innovations are based on the utilisation of data. The Internet of Things enables the availability of large volumes of data, while people create data using social media. Nowadays, a worldwide exchange of data continuously influences society, resulting in new business opportunities and new services. This has already resulted in the rise of "data science", which aims at better understanding how to use all these data and benefit from them. Furthermore, the presented circumstances create a need of Big Data Analytics tools: instruments to deal with the vast amount of data available. This section dwells into the topic of Big Data in the compute continuum, particularly the management in upper architectural layers of huge volumes of data collected from CPSs and IoT end devices. This process leads to the concept of Big Data Analytics as a Service, that is also studied here.

4.8.1 Big Data as a Service in Cyber-Physical Systems

CPS data sources usually produce a huge quantity of data that is difficult to manage and process in a centralized way due to a number of limitations [187] [188]. These limitations can be physical, technical, legal, etc. Instead, an organized decentralized mechanism to manage and process these data is a more efficient solution. It may be the only feasible solution in cases of extremely high-volume multi-sources flows of data.

The definition of Big Data Analytics [189] can be summarized as the set of techniques needed to process (collect, organize, analyze, etc.) these huge amounts of data in order to obtain knowledge. Moreover, it is possible to differentiate among big data descriptive analytics (which describe the existing data), big data predictive analytics (which forecasts trends of future situations) and big data prescriptive analytics (which advises on actions to take or behaviors to adopt).

This processing can be done in a decentralized way, as stated above. Such decentralization is also motivated by the need to reconcile two opposite facts, namely, the increasing complexity of Big Data technologies and processes, and the need to increase the acceptance level and the usability of such technologies by a general public [190].

This arises the need of a Big Data Analytics as a Service (BDaaS) model [191], that consists of a set of automatic tools and methodologies that allow users to manage Big Data Analytics and deploy a full Big Data pipeline addressing their goals. User's requirements, that are transformed in a ready-to-be-executed Big Data pipeline, can be defined in five different conceptual areas: data preparation, data representation, data analytics, data processing and data visualization and reporting.

Users with different skills and expertise can benefit by adopting the BDaaS paradigm. Users lacking expertise proper of data scientists can use a BDaaS solution for preparing the real analytics, reason on data to find out hidden patterns and information, and solve business problems. User lacking expertise proper of data engineers can use a BDaaS solution to automatically identify and deploy the proper set of technologies that accomplish their requirements. Users lacking both expertise can still use BDaaS solutions for a proper initiation in the Big Data realm.

A key relevant trait of the BDaaS model, when applied to CPS and IoT scenarios and meeting the decentralized approach aforementioned, is that the management and processing of the huge volume of data produced is moved to a cloud level outside of the device level where such data usually originates. The integration between the CPS world and the cloud world poses a number of challenges. For instance, according to [192], these challenges are related to synchronization, standardization, balancing, reliability and management, among others.

Another step forward to ease such integration is the so-called edge computing [193]. Informally, this paradigm consists in placing an additional level of on-premises computing resources next to the local device level, which acts as an intermediator to the cloud level. This intermediate level is suitable to host real-time processing and on-the-fly analysis. Thus, it offers a number of advantages to mission-critical applications that have low-latency or real-time requirements, are location-aware, have mobile components, are geographically distributed, integrate large-scale sets of nodes (e.g., sensors), are heterogeneous and have interoperability requirements, or generally speaking, interact a lot with the cloud level [194].

The use of an edge level in a cyber-physical setting presents a number of advantages [192]. First of all, the edge nodes are closer to the physical devices and the data acquired. This helps to achieve a better performance and a lower energy cost at accessing such data. In addition, the nodes are simpler and lighter than the cloud servers. This means that it is easier and cheaper to distribute them geographically and scale to better adapt to the existing constraints, the user needs, etc. This also helps to support the mobility patterns of both the users and the physical devices. Moreover, the fact that edge nodes are an intermediate level between the devices and the cloud plane makes this layer to become an effective interoperability and standardization level.

4.8.2 Choosing technologies for Big Data Analytics and Machine Learning

When the time comes to choose a platform for the execution of Big Data Analytics processes according to the BDaaS paradigm, two main characteristics to consider are the data ingestion rate, which should be as high as possible, and the ability to efficiently manage and process large amounts of data. Such a platform must allow its users to manage their tasks (in this context, called “experiments”), specifically: manage the life cycle of experiments, design and implement experiments, command the execution of one or multiple experiments and monitor the status of experiments. It should also provide an interface for the management and control, both individual and grouped, of the experiments on execution, along with their usage of computational resources. Furthermore, the platform must allow the simultaneous execution of different experiments over different datasets.

Big Data Analytics and Machine Learning experiments, due to the nature of the computations they perform and the volume of data they process, are likely to be very demanding in terms of their usage of computational resources. Therefore, they may collapse the underlying infrastructure if a proper mechanism for control and distribution is not implemented. This mechanism’s goal is to make possible that all processes being executed concurrently run smoothly, which is achieved by tracking the experiments active at a given time and their resources consumption.

Another important aspect to think about when choosing a technology is its usability. Since it may be utilized by a range of users with a completely different level of knowledge on the subject of Big Data Analytics and Machine Learning, it must encompass all these grades of expertise. Options for users with a lower degree of expertise must be complete and provide assistance based on the experiment's use case and the type of data. As well, these options must be powerful enough to completely fulfil the needs of advanced users, allowing them to carry out more specific experiments and supporting programming languages typically used for data analytics and the inclusion of third-party libraries.

When considering the BDaaS solution to adapt in the cloud and edge layers of TRANSACT's architecture, in the light of all the factors having been discussed, three particularly relevant technologies have been identified, which will be next analyzed: BigML, H2O (along with H20 Driverless AI) and RADIATUS (along with AITANA).

4.8.2.1 BigML

BigML[6] is a Machine Learning platform dedicated to novice users, exposing a REST API from which it can be used. Regarding data processing and storage, BigML's maximum data ingestion is 64GB, which becomes a limitation in many projects. BigML supports importing text data from files, but does not support image processing. Even if it provides complete information about the data imported, statistics and missing data, its support to data pre-processing is limited to simple operations such as splitting or joining the data set, not being able to normalize and standardize data or set categories for discrete data.

In order to support multiple simultaneous users, BigML features an organizational system and, in the context of an organization, both public and private projects. Each of these projects comprises a set of experiments, whose lifecycle allows them to be created, modified, deleted, executed and monitored. On the other hand, BigML's documentation does not provide details at all about the management of system resources, independently of the deployment setting.

Regarding usability, BigML provides an intuitive interface for performing research tasks. It features an extensive catalog of techniques and algorithms for AI and ML that can be easily applied, bringing the ability to experiment to inexperienced users. These algorithms are divided into two groups, the first of them covering classification and regression tasks, while the second one is dedicated to clustering tasks. Nevertheless, some advanced algorithms and techniques like Deep Learning are not available, neither own-developed algorithms and third-party libraries can be integrated, thus restricting some more complex processing.

It is not free neither open-sourced, featuring different subscription plans in terms of the task size, how many concurrent tasks that can be executed, the number of servers onto which the solution will be deployed and the aggregated number of their cores.

4.8.2.2 H2O and Driverless AI

H2O[7] is an AI platform provider whose proclaimed intention is to bring AI to every user, although actually a wide degree of knowledge is required to understand and use their applications. H2O provides various tools, both open-source and private, the most remarkable of which are the homonymous H2O (open-sourced) and DriverlessAI (private).

[6] BigML, https://bigml.com

[7] H2O.ai | AI Cloud Platform, https://www.h2o.ai

H2O[8] is a platform based on Python Notebooks. It is characterized by its assistants for developing algorithms, which can be implemented in both Python and R, and the integration with distributed systems and in-memory processes. It provides a tool for exporting the models generated in a proprietary format and deploy them for production. Furthermore, H2O includes a system for automatic machine learning. H2O also provides some tools that enable users to manage their experiments and their lifecycle, as well as their control and monitoring while they are being executed. Regarding concurrent users, H2O allows them to make use of the same deployment simultaneously and collaborate in the same set of experiments. When it is deployed on-premises, it allows to control the deployment infrastructure, but fails in providing internal mechanisms to manage system-level resources, such as specifying the resources to be assigned to a concrete experiment. Summarizing and in terms of usability, H2O displays great functionalities, such as the inclusion of algorithms developed by the user and third-party libraries, various ready-to-be-executed algorithms, support to deep learning and other advanced techniques, and automatic machine learning. However, it is directed to experts and looks too complex for inexperienced users to take much advantage of it.

Driverless AI[9] is an assisted platform completely oriented to AutoML with a polished interface, easily interpretable result graphics and different data sources. This platform leverages intensive use of CPUs and GPUS, as well as optimized data ingestion from HDF, S3, Spark, etc. Driverless AI can be deployed in H2O's servers and on-premises infrastructure, but both deployment models imply an economical cost, as data storage and processing depend of the license quota signed. Multiple users can use the same cluster simultaneously, but Driverless AI is user-licensed, preventing users to collaborate by accessing other experiments than their own. Driverless AI provides a good deal of educational material, such as guided methods, tutorials and great visual support intended to ease its adoption by beginner users. Its algorithm set and workflow is quite complete and offers powerful yet simple options. Finally, although its documentation refers that it is possible to develop and execute custom code, this feature's implementation is far from being straightforward, being set aside for more experienced users.

4.8.2.3 RADIATUS and AITANA

Radiatus[10] is a Big Data Analytics as a Service platform that makes easier the configuration and deployment of distributed architectures featuring the latest technologies for data analytics, both in batch and streaming modes. It features tools for training Machine Learning and Deep Learning models, and includes technologies such as Jupyter, Zeppelin, Spark, Flink, HDFS, Cassandra, CockroachDB, Kafka, Kibana, etc.

AITANA[11] is a platform whose purpose is to make Machine Learning and Deep Learning processes and workflows to be easily interpretable, providing the flexibility degree needed by users to develop and integrate custom algorithms and third-party libraries. It integrates with Radiatus and, as a sole system, can be deployed on the infrastructure of a cloud provider or on a local cluster. Neither of them holds a license-based pricing.

[8] H2O | Open Source, Distributed Machine Learning for Everyone, https://www.h2o.ai/products/h2o/

[9] H2O Driverless AI | Award-winning Automatic Machine Learning (AutoML) Platform, https://www.h2o.ai/products/h2o-driverless-ai/

[10] Radiatus | Elastic Infrastructure for Big Data Analytics in the Cloud, https://radiatus.iti.es/

[11] AITANA, the European project ITI is working on and that will make easy the work of data scientists, https://www.iti.es/noticias/aitana-el-proyecto-en-el-que-trabaja-iti-y-que-facilitara-la-labor-de-los-cientificos-de-datos/

Radiatus' multitenancy management allows the assignation of concrete computational resources to a group of users. A user can just deploy an instance of the DistributedML service and use AITANA to launch multiple experiments isolated from the other users' perspective and resources. New resources and components can be installed by a global administrator and associated to a group of users, whose "local" administrator can, in turn, dynamically and elastically manage the resources assigned to each user.

Regarding experiment management, AITANA can import data in various formats and from different sources. These data can be pre-processed and being subsequently used in a training process. The training and evaluation algorithms can be chosen by the user, who, in addition to predefined algorithms, can specify custom algorithms that might be developed in a code editor provided by the platform itself. AITANA projects are conceived to keep multiple experiments, so that the user can compare different experiments in a visual fashion and choose the one to be deployed for production. This perspective also provides an interface for managing the experiments' lifecycle.

Both tools, Radiatus and AITANA, are focused on easing the access, usage and deployment of Big Data Analytics technologies to non-technical users, providing mechanisms that are easy to assimilate and utilize. Nevertheless, they do not impose limits to advanced users that could need to develop and execute their own solutions.

4.8.3 Technology comparison

The following tables compare the technological solutions having been discussed. This comparison is first carried from a perspective focused on experiment execution and, next, from a system and resources management approach.

Experiment execution	BigML	H2O	Driverless AI	RADIATUS & AITANA
Machine Learning	YES	YES	NO	YES
Deep Neural Networks	NO	YES	YES	YES
Huge data volume management	64 GB	YES	YES	YES
Hyperparameter optimization	OptiML	YES	AutoML	Brute force
Inclusion as a library	NO	YES	NO	YES
Third-party libraries support	NO	YES	NO	YES
Custom code	NO	YES	NO	YES

Command line interface	REST API	YES	YES	YES
Graphical user interface	YES	YES	YES	YES
Model development via workflows	NO	NO	NO	YES
Result visualization	YES	YES	YES	YES
Model training	YES	YES	YES	YES
Model deployment	YES	YES	YES	YES
GPU exploitation	NO	YES	YES	YES
Usability (0 hard, 10 easy)	7	4	6	9
Usable without programming expertise	YES	NO	YES	YES
On-premises deployment	YES	YES	YES	YES

Table 3: Comparison of technological solutions, regarding experiment execution

System and resources management	BigML	H2O	Driverless AI	RADIATUS & AITANA
Data ingestion	64 GB	Illimited	Illimited	Illimited
Data management and pre-processing	64 GB	Illimited	Illimited	Illimited
Simultaneous users	YES	YES	YES, in different experiments	YES
Experiment concurrency	YES	YES	YES	YES
User-managed experiments	YES	YES	YES	YES

Design and develop experiments	YES	YES	YES	YES
Execute experiments	YES	YES	YES	YES
Monitor experiments	YES	YES	YES	YES
Manage experiment resources	-	-	-	YES
Manage system resources	-	-	-	YES
Easy experimentation	YES	Some options	YES	YES
Advanced experimentation	NO	YES	YES	YES

Table 4: Comparison of technological solutions, regarding system and resources management.

As the tables above display, the system obtained by the combination of RADIATUS and AITANA outperforms the other three options in both categories: experiment execution and system and resources management. Its position as the best option compared stands out specially in terms of its usability and its management of resources at the experiment and system levels. The facts that its usage is license-free and the tools that Radiatus provides are open-source also entail a significant advantage when compared with other alternatives whose usage could be bound to a private license.

4.8.4 Industrial practices

4.8.4.1 DENSO Automotive

In current industrial practices, many open-source frameworks for machine learning, data analysis and Big Data are used. These include for Bit Data the Apache Spark (https://spark.apache.org) and Apache Flink (http://flink.apache.org) solutions , including their software stack (cluster management etc), as well as built in and add-on solutions. There is considerable community support for setup and operation of these, as well as commercial, hosted solutions from many cloud vendors. The work in these systems is typically done with interactive, browser based “notebooks”, e.g. https://jupyter.org.

For machine learning, there is a large variety of options. Many data scientists use the ecosystem of the R programming language, more from statistical side. (https://www.r-project.org/). For machine learning with neuronal networks, there exist several open-source solutions like TensorFlow.

For the above, it is possible to install and operate this in local premise, or to use hosted cloud services. These can be used for storage and computing, while some only use the computing, either just on virtual machine level or using hosted services. The latter are fully managed by the cloud service providers, often using existing

above open-source packages. This will also include provisioning, service management as well as updates and security management.

4.9 Eclipse Open-Source IoT and Edge Technologies

The **Eclipse IoT[12] and Edge[13] Working Groups** are collaborations of industry and academic partners who are building a set of open-source technology for the Edge and IoT ecosystem. The focus of these collaborations is on building 1) open-source implementations of IoT and Edge standards and protocols, 2) open-source frameworks and services that will be used by IoT and Edge solutions, and 3) tools for IoT and Edge developers.

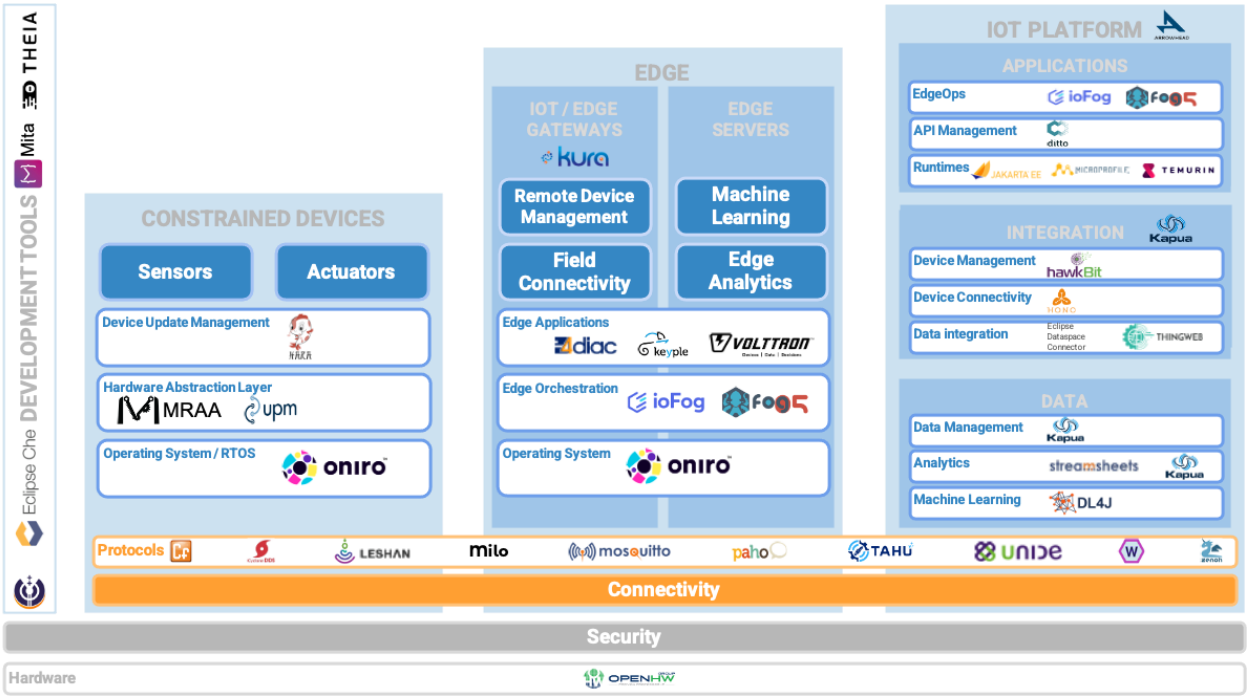


Figure 20: Eclipse IoT and Edge landscape

Currently, the Eclipse IoT and Edge Working Groups comprise around 50 member companies who collaborate on more than 40 open-source projects that help building professional applications for the Edge-Cloud Continuum.

Figure 20 shows an overview of the Eclipse IoT landscape, mapping projects to functional concerns of an IoT architecture. One of the most prominent examples is **Eclipse Mosquitto**[14], an open-source message broker that implements the MQTT protocol versions 5.0, 3.1.1 and 3.1. Mosquitto is lightweight and is suitable for use on all devices from low power single board computers to full servers. The MQTT protocol provides a lightweight method of carrying out messaging using a publish/subscribe model. This makes it suitable for

[12] https://iot.eclipse.org/

[13] https://edgenative.eclipse.org/

[14] https://mosquitto.org/

Internet of Things messaging such as with low power sensors or mobile devices such as phones, embedded computers or microcontrollers. Other protocol implementations are Eclipse Californium for CoAP, Eclipse Cyclone DDS for DDS, Eclipse Leshan and Wakaama for LwM2M, Eclipse Milo for OPC UA, and Eclipse Amlen and Paho also for MQTT.

In large, distributed IoT applications, updating devices is a challenge. **Eclipse hawkBit™** is a domain independent back-end framework for rolling out software updates to constrained edge devices as well as more powerful controllers and gateways connected to IP based networking infrastructure. hawkBit offers a direct device integration via HTTP or a device management federation API which allows to connect devices with different protocol adapters. Users can make use of the graphical user interface and other service can interact with hawkBit through the RESTful management API. hawkBit supports an easy and flexible rollout management which allows to update a large amount of devices in separated groups.

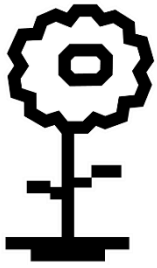
The Eclipse IoT Working Group provides **Eclipse IoT Packages™**[15], an effort to create easy to deploy Eclipse IoT based, end-to-end scenarios, on top of Kubernetes and Helm. The Eclipse IoT Packages™ project provides a home for use case focused IoT deployments, based on Eclipse IoT technology. It takes the building blocks that the different projects provide, and adds the necessary glue code, needed to create more complete setups. Packages not only provide deployment scripts, but also descriptions what the benefit of the integrated package is, and some initial, tutorial like steps, to play with the setup, once it is deployed. Because integrating different software components is already hard enough, the IoT packages project chose Kubernetes as its cloud side deployment platform. Currently two IoT packages are available:

- **Cloud2Edge,** a package connecting and managing sensor style devices. Connecting sensors to the cloud and processing data with a digital twin platform. Cloud2Edge includes **Eclipse Hono** and **Eclipse Ditto**.
- **Telemetry end-to-end,** a package showing telemetry data acquisition end-to-end: Microcontroller firmware to cloud side data processing, using Drogue IoT and Apache Kafka in the process. It includes **Eclipse Ditto**, **Eclipse Kura**, **Eclipse Mosquitto**, and **Eclipse Streamsheets**.

[15] https://www.eclipse.org/packages/

5 Open-Source components

In addition to the solutions proposed by the Eclipse IoT and Edge working groups, presented in Section 4.9, some of the participating partners in TRANSACT have proposed a collection of open-source technologies and components. These, either are planned to be actively exploited in TRANSACT's development and instantiation of the reference architecture or are potentially relevant to TRANSACT's scope and should be taken into consideration, since their adoption could become beneficial. They are listed below, along with a brief description for each one.

Name	Flower
Product Owner	Adap
License	Apache 2.0
Description	Flower (flwr) is a framework for building federated learning systems. It provides a unified approach to federated learning, analytics, and evaluation, enabling the federation of any workload, machine learning framework and programming language. The design of Flower is based on a few guiding principles: customizability, extendability, framework-agnosticism and understandability.
Website	https://github.com/adap/flower
Figure	
Proposed by	AVL

Name	gRPC
Product Owner	Google
License	Apache 2.0
Description	GRPC is a modern open-source high performance Remote Procedure Call (RPC) framework that can run in any environment. It can efficiently connect services in and across data centers with pluggable support for load balancing, tracing, health checking and authentication. It is also applicable in last mile of distributed computing to connect devices, mobile applications and browsers to backend services.
Website	https://grpc.io

Figure	
Proposed by	AVL

Name	HiveMQ CE
Product Owner	Hive MQ
License	Apache 2.0
Description	HiveMQ CE is a Java-based open-source MQTT broker that fully supports MQTT 3.x and MQTT 5. It is ideal for developers that need to embed a MQTT broker into a Java application and for creating IoT pilots and PoC.
Website	https://www.hivemq.com/developers/community/
Figure	
Proposed by	AVL

Name	Eclipse POOSL
Product Owner	Eclipse Foundation
License	Eclipse Public License 2.0
Description	Eclipse POOSL (Parallel Object-Oriented Specification Language) offers a general purpose method for describing and simulating (embedded) system architectures for the early evaluation of key structural and behavioral concepts, requirements and performance. This lightweight modeling and simulation approach shortens the development time of complex high-tech systems by providing fast insights into requirements and early design decisions, thereby reducing the risk of expensive iterations during design, integration and testing.
Website	https://projects.eclipse.org/projects/modeling.poosl
Figure	
Proposed by	TNO, TU/e

Name	Eclipse TRACE4CPS
Product Owner	Eclipse Foundation
License	Eclipse Public License 2.0
Description	Eclipse TRACE4CPS is a customizable, domain-independent and source-independent Gantt chart viewer with mathematically-founded analysis support. Eclipse TRACE4CPS supports the visualization of activities on resources as a function of time (Gantt charts), as well as the visualization of continuous signals. Eclipse TRACE4CPS also supports several analysis techniques to identify bottlenecks, check formally-specified (performance) properties, and analyze resource usage. A key feature of Eclipse TRACE4CPS is the ability to configure the identification, selection and visualization of such information to match any specific application domain.
Website	https://projects.eclipse.org/projects/technology.trace4cps
Figure	
Proposed by	TU/e

Name	OpenTelemetry
Product Owner	Cloud Native Computing Foundation
License	Apache 2.0
Description	OpenTelemetry provides high-quality, ubiquitous, and portable telemetry to enable effective observability. It can be used to instrument, generate, collect, and export telemetry data (metrics, logs, and traces) to help the analysis of software's performance and behavior. It is delivered as a collection of tools, APIs and SDKs.
Website	https://opentelemetry.io
Figure	

Proposed by	TU/e

Name	Prometheus
Product Owner	Eclipse Foundation
License	Apache 2.0
Description	Prometheus is an open-source systems monitoring and alerting toolkit. It records real-time metrics in a time series database (allowing for high dimensionality) build using a HTTP pull model, with flexible queries and real-time alerting. Metrics information and data is stored with the timestamp at which it was recorded, alongside optional key-value pairs called labels.
Website	https://prometheus.io/
Figure	
Proposed by	ITI

Name	Apache Superset
Product Owner	Apache Software Foundation
License	Apache 2.0
Description	Apache Superset is a modern data exploration and visualization platform. It is fast, lightweight, intuitive, and loaded with options that make it easy for users of all skill sets to explore and visualize their data, from simple line charts to highly detailed geospatial charts.
Website	https://superset.apache.org
Figure	
Proposed by	ITI

6 Conclusions

This deliverable document provides an initial version of TRANSACT's reference architecture, describing its three tiers, its main characteristics and its overall purpose. It also carries out a study on related topics that gives an insight of the current state of the art, enabling a further development and definition of the reference architecture. Useful open-source technologies, relevant to TRANSACT's scope, are also listed and briefly described, aiming at their integration, if advisable, in instantiations of the reference architecture here presented.

Now that an early version of the reference architecture has been presented and the base knowledge for its further improvement has been introduced, the next steps go along the path of advancing this architecture and its detailed specification. It provides the framework that particular implementations of the architecture will be based on. Therefore, as it progresses, subsequent tasks must ensure that it does not deviate from its concept and purpose, at the same time it keeps up with its requirements and fulfils its role of becoming a valid foundation for horizontal and use case demonstrators.

In the context of TRANSACT's WP2, a next version of this deliverable document D2.1, forthcoming at the end of the project, will provide and describe the final version of the reference architecture. It also will explain how the development process of TRANSACT, from the contents contained in the current version of the document, along with the tasks comprised by various stages of the project, will have come to define an architecture that fulfils all the expected requirements.

7 References

[1] E. B. Fernandez, Security Patterns in Practice: Designing Secure Architectures Using Software Patterns, John Wiley & Sons, 2013.

[2] N. Marko, “SECREDAS Deliverable Report: Design Pattern Descriptions v2,” 2021.

[3] Y. Bartal, A. Mayer, K. Nissim and A. Wool, "Firmato: A Novel Firewall Management Toolkit," *ACM Transactions on Computer Systems,* vol. 22, no. 4, pp. 381-420, November 2004.

[4] S. Pozo, R. M. Gasca, A. M. Reina-Quintero and A. J. Varela-Vaca, “CONFIDDENT: A model-driven consistent and non-redundant layer-3 firewall ACL design, development and maintenance framework,” *Journal of Systems and Software,* vol. 85, no. 2, pp. 425-457, February 2012.

[5] S. Martínez, J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia and J. Cabot, “Model-Driven Extraction and Analysis of Network Security Policies,” in *Model-Driven Engineering Languages and Systems*, 2013.

[6] OASIS, “eXtensible Access Control Markup Language (XACML) Version 3.0,” in *OASIS Standard*, 2013, pp. 1-154.

[7] E. Yuan and J. Tong, “Attributed based access control (ABAC) for Web services,” in *IEEE International Conference on Web Services (ICWS'05)*, 2005.

[8] R. Sandhu, D. Ferraiolo and K. Richard, “The NIST Model for Role-Based Access Control: Towards a Unified Standard,” in *Fifth ACM Workshop on Role-Based Access Control (RBAC '00)*, Berlin, Germany, 2000.

[9] M. Alam, M. Hafner and R. Breu, “Constraint based role based access control in the SECTET-framework: A model-driven approach,” *Journal of Computer Security,* vol. 16, no. 2, pp. 223-260, January 2008.

[10] T. Mouelhi, F. Fleurey, B. Baudry and Y. Le Traon, “A Model-Based Framework for Security Policy Specification, Deployment and Testing,” in *Model Driven Engineering Languages and Systems*, 2008.

[11] J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, S. Martinez and J. Cabot, “Management of Stateful Firewall Misconfiguration,” *Computers & Security,* pp. 64-85, November 2013.

[12] J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia and S. preda, “MIRAGE: A Management Tool for the Analysis and Deployment of Network Security Policies,” in *Data Privacy Management and Autonomous Spontaneous Security*, 2011.

[13] S. Prabavathy, K. Sundarakantham and S. M. Shalinie, “Design of cognitive fog computing for intrusion detection in Internet of Things,” *Journal of Communications and Networks,* vol. 20, no. 3, pp. 291-298, June 2018.

[14] I. Bica, B.-C. Chifor, S.-C. Arseni and I. Matei, “Multi-Layer IoT Security Framework for Ambient Intelligence Environments,” *Sensors,* vol. 19, no. 18, p. 4038, September 2019.

[15] M. Spörk, M. Schuß, C. A. Boano and K. Römer, “Ensuring End-to-End Dependability Requirements in Cloud-based Bluetooth Low Energy Applications,” in *EWSN*, 2021.

[16] M. Di Francesco, G. Anastasi, M. Conti, S. K. Das and V. Neri, “Reliability and energy-efficiency in IEEE 802.15. 4/ZigBee sensor networks: An adaptive and cross-layer approach,” *IEEE Journal on selected areas in communications,* vol. 29, pp. 1508-1524, 2011.

[17] P. Park, S. C. Ergen, C. Fischione, C. Lu and K. H. Johansson, “Wireless network design for control systems: A survey,” *IEEE Communications Surveys & Tutorials,* vol. 20, pp. 978-1013, 2017.

[18] M. Zimmerling, L. Mottola and S. Santini, “Synchronous transmissions in low-power wireless: A survey of communication protocols and network services,” *ACM Computing Surveys (CSUR),* vol. 53, pp. 1-39, 2020.

[19] F. Ferrari, M. Zimmerling, L. Thiele and O. Saukh, “Efficient Network Flooding and Time Synchronization with Glossy,” *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks,* pp. 73-84, 2011.

[20] B. Al Nahas, S. Duquennoy and O. Landsiedel, “Concurrent Transmissions for Multi-Hop Bluetooth 5,” in *EWSN*, 2019.

[21] S. Kumar, M. P. Andersen, H.-S. Kim and D. E. Culler, “Performant TCP for Low-Power Wireless Networks,” in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020.

[22] K. Pretz, “The next evolution of the internet,” *IEEE Magazine,* vol. 50, no. 5, 2013.

[23] J. Kuusijärvi, R. Savola, P. Savolainen and A. Evesti, “Mitigating IoT security threats with a trusted Network element,” in *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2016.

[24] S. Campadello, “Peer-to-peer security in mobile devices: a user perspective,” in *Proceedings. Fourth International Conference on Peer-to-Peer Computing, 2004. Proceedings.*, 2004.

[25] S. Chollet, L. Pion, N. Barbot and C. Michel, “Secure IoT for a Pervasive Platform,” in *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2018.

[26] “Alliance for Internet of Things Innovation,” [Online]. Available: https://ec.europa.eu/digital-single-market/en/alliance-internet-things-innovation-aioti accessed 21st Feb 2019.

[27] “IoT 2020: Smart and secure IoT platform”.

[28] Study Group 13, “Next Generation Networks – Frameworks and Functional Models: Overview of the Internet of Things,” International Telecommunication Union, Geneva, 2012.

[29] ZVEI – German Electrical and Electronic Manufacturers’ Association, “Industrie 4.0: The Reference Architectural Model Industrie 4.0 (RAMI 4.0),” Krankfurt, 2015.

[30] Industrial Internet Consortium, “Industrial Internet Reference Architecture (Version 1.7),” Object Management Group, Needham, MA, US, 2015.

[31] "Internet of Things – Architecture Consortium, The IoT Architectural Reference Model (ARM) - D1.3," European Commission, Luxembourg, 2012.

[32] "Fiware open specifications," [Online]. Available: https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Summary_of_FIWARE_Open_Specifications. [Accessed 13 January 2018].

[33] [Online]. Available: https://www.trusted-iot.org/ .

[34] "Amazon Web Services," [Online]. Available: https://aws.amazon.com/documentation/iot/. [Accessed 13 January 2018].

[35] J. Guth, U. Breitenbücher, M. Falkenthal, F. Leymann and L. Reinfurt, "Comparison of IoT platform architectures: A field study based on a reference architecture," in *2016 Cloudification of the Internet of Things (CIoT)*.

[36] "IPSO Alliance, later OMA SpecWorks," [Online]. Available: https://www.omaspecworks.org/. [Accessed 21 February 2019].

[37] "The Internet Engineering Task Force (IETF)," 21 February 2019. [Online]. Available: http://www.ietf.org/.

[38] "ETSI M2M/Smart M2M," 21 February 2019. [Online]. Available: http://www.etsi.org/.

[39] "One M2M forum," 21 February 2019. [Online]. Available: http://www.onem2m.org/.

[40] Y. Ahmad, B. Berg, U. Cetintemel, M. Humphrey, J. Hwang, A. Jhingran, A. Maskey, O. Papaemmanouil, A. Rasin, N. Tatbul, W. Xing, Y. Xing and S. Zdonik, "Distributed operation in the Borealis stream processing engine," in *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA.

[41] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein and J. Widow, "STREAM: The Stanford Stream Data Manager," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2003.

[42] T. Akidau, A. Balikov, K. Bekiroglu, S. Chernyak, J. Haberman, R. Lax, S. MacVeety, D. Mills, P. Nordstrom and S. Whittle, "MillWheel: Fault-Tolerant Stream Processing at Internet Scale," *Proc. VLDB Endow.,* vol. 6, no. 11, pp. 1033-1044, August 2013}.

[43] "Apache Storm," [Online]. Available: http://storm.apache.org/.

[44] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy and S. Taneja, "Twitter Heron: Stream Processing at Scale," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2015.

[45] M. Zaharia, T. Das, H. Li, T. Hunter, S. Snenker and I. Stoica, "Discretized Streams: Fault-Tolerant Streaming Computation at Scale," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013.

[46] "Apache Samza," [Online]. Available: http://samza.apache.org/.

[47] W. Van Raemdonck, T. Van Cutsem, K. S. Esmaili, M. Cortes, P. Dobbelaere, L. Hoste, E. Philips, M. Roelands and L. Trappeniers, "Building Connected Car Applications on Top of the World-Wide

Streams Platform: Demo," in *Proceedings of the 11th ACM International Conference on Distributed and Event-Based Systems*, Barcelona, June 2017.

[48] "World Wide Streams website," [Online]. Available: http://worldwidestreams.io. [Accessed January 2018].

[49] "Apache Quarks," [Online]. Available: http://quarks.incubator.apache.org/ .

[50] L. Amini, H. Andrade, R. Bhagwan, F. Eskesen, R. King, P. Selo, Y. Park and C. Venkatramani, "SPC: A Distributed, Scalable Platform for Data Mining," in *Proceedings of the 4th International Workshop on Data Mining Standards, Services and Platforms}*, New York, NY, USA, 2006.

[51] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu and M. Doo, "SPADE: The System s Declarative Stream Processing Engine," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2008.

[52] "Apache Hadoop," [Online]. Available: http://hadoop.apache.org/.

[53] M. Isard, M. Budiu, Y. Yu, A. Birrell and D. Fetterly, "Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, 2007.

[54] Y. Yu, M. Isard, D. Fatterly, M. Budiu, U. Erlingsson, P. K. Gunda and J. Currey, "DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language," in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, 2008.

[55] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu and R. Murthy, "Hive - a petabyte scale data warehouse using Hadoop," in *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, 2010.

[56] R. Pike, S. Dorward, R. Griesemer and S. Quinlan, "Interpreting the Data: Parallel Analysis with Sawzall," *Scientific Programming,* vol. 13, no. 4, pp. 277-298, October 2005.

[57] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *6th Symposium on Operating Systems Design & Implementation (OSDI 04)*, Berkeley, CA, USA, 2004.

[58] B. Theeten and N. Janssens, "CHive: Bandwidth Optimized Continuous Querying in Distributed Clouds," *{IEEE Transactions on Cloud Computing,* vol. 3, no. 2, pp. 219-232, 2015.

[59] J. {Latvakoski, A. Iivari, P. Vitic, B. Jubeh, M. B. Alaya, T. Monteil, Y. Lopez, G. Talavera, J. Gonzalez, N. Granqvist, M. Kellil, H. Ganem and T. Väisänen, "A Survey on M2M Service Networks," *{Computers},,* vol. 3, no. 4, pp. 130-173, 2014.

[60] J. Latvakoski, Small world for dynamic wireless cyber-physical systems: Dissertation, VTT Technical Research Centre of Finland, 2016.

[61] I. Roussaki, M. Chantzara, S. Xynogalas and M. Anagnostou, "The virtual home environment roaming perspective," in *IEEE International Conference on Communications, 2003. ICC '03*, 2003.

[62] M. S. Familiar, J. F. Martínez, I. Corredor and C. García-Rubio, "Building service-oriented Smart Infrastructures over Wireless Ad Hoc Sensor Networks: A middleware perspective," *Computer Networks,* vol. 56, no. 4, pp. 1303-1328, 2012.

[63] P. T. Eugster, P. A. Felber, R. Guerraoui and A.-M. Kermarrec, “The Many Faces of Publish/Subscribe,” *ACM Comput. Surv.,* vol. 35, no. 2, pp. 114-131, 2003.

[64] P. Saint-Andre, “Extensible Messaging and Presence Protocol (XMPP) Core. IETF RFC3920,” 2004.

[65] P. Saint-Andre, “Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. IETF RFC3921,” 2004.

[66] P. Millard, P. Saint-Andre and R. Meijer, “XEP-0060: Publish-Subscribe,” 2019.

[67] “ISO/IEC 20922: 2016 Information technology Message Queuing Telemetry Transport (MQTT) v3.1.1,” Springer, Cham, Germany, 2016.

[68] OASIS, “MQTT 3.1.1 Specification,” 2015.

[69] J. Rosenberg, E. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley and E. Schooler, “SIP: Session Initiation Protocol, IETF RFC 3261,” 2002.

[70] X. Zhang, C.-F. Law, C.-L. Wang and F. Lau, “Towards pervasive instant messaging and presence awareness,” *International Journal of Pervasive Computing and Communications,* vol. 5, no. 1, pp. 42-60, 2009.

[71] J. Latvakoski and J. Heikkinen, “A Trustworthy Communication Hub for Cyber-Physical Systems,” *Future Internet,* vol. 11, no. 10, p. 211, 2019.

[72] R. Román, J. Zhou and j. López, “On the features and challenges of security and privacy in distributed Internet of Things,” *Computer Networks,* vol. 57, no. 10, pp. 2266-2279, 2013.

[73] A. Ouaddah, H. Mousannif, A. Abou Elkalam and A. Ait Ouahman, “Access control in the Internet of Things: Big challenges and new opportunities,” *Computer Networks,* vol. 112, pp. 237-262, 2017.

[74] Á. Alonso, F. Fernández, L. Marco and J. Salvachúa, “IAACaaS: IoT Application-Scoped Access Control as a Service,” *uture Internet,* vol. 9, no. 4, 2017.

[75] S. Kaluvuri, A. Egner, j. Hartog and N. Zannone, “SAFAX - An Extensible Authorization Service for Cloud Environments,” *Frontiers in ICT,* 2015.

[76] U.S. Department of Health and Human Services, “Health Insurance Portability and Accountability Act of 1996,” 1996. [Online]. Available: https://aspe.hhs.gov/reports/health-insurance-portability-accountability-act-1996.

[77] HITRUST Alliance, “HITRUST CSF,” HITRUST Alliance, [Online]. Available: https://hitrustalliance.net/csf-license-agreement/.

[78] Amazon Web Services, Inc., “Shared Responsibility Model,” [Online]. Available: https://aws.amazon.com/compliance/shared-responsibility-model/.

[79] Internet Research Task Force, “ChaCha20 and Poly1305 for IETF Protocols,” June 2018. [Online]. Available: https://www.rfc-editor.org/rfc/pdfrfc/rfc8439.txt.pdf.

[80] P. R. Lewis, A. Chandra, F. Faniyi, K. Glette, T. Chen, R. Bahsoon, J. Torresen and X. Yao, "Architectural Aspects of Self-Aware and Self-Expressive Computing Systems: From Psychology to Engineering," *Computer,* vol. 48, no. 8, pp. 62-70, 2015.

[81] S. Kounev, J. O. Kephart, A. Milenkoski and X. Zhu, Self-Aware Computing Systems, Springer, 2017.

[82] P. Arcaini, E. Riccobene and P. Scandurra, "Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation," in *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, IEEE, 2015, pp. 13-23.

[83] J. Andersson, R. De Lemos, S. Malek and D. Weyns, "Modeling Dimensions of Self-Adaptive Software Systems," in *Software Engineering for Self-Adaptive Systems*, 2009, pp. 27-47.

[84] K. Bellman, C. Landauer, N. Dutt, L. Esterle, A. Herkersdorf, A. Jantsch, N. TaheriNejad, P. R. Lewis, M. Platzner and K. Tammemäe, "Self-Aware Cyber-Physical Systems," *ACM Trans. Cyber-Phys. Syst.,* vol. 4, no. 4, pp. 1-26, 2020.

[85] F. Palumbo, T. Fanni, C. Sau, A. Rodríguez, D. Madroñal, K. Desnos, A. Morvan, M. Pelcat, C. Rubattu, R. Lazcano, L. Raffo, E. de la Torre, E. Juárez, C. Sanz and P. Sánchez de Rojas, "Hardware/Software Self-adaptation in CPS: The CERBERO Project Approach," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, Cham, Springer, 2019, pp. 416-428.

[86] M. Götzinger, D. Juhász, N. Taherinejad, E. Willegger, B. Tutzer, P. Liljeberg, A. Jantsch and A. M. Rahmani, "RoSA: A Framework for Modeling Self-Awareness in Cyber-Physical Systems," *IEEE Access,* vol. 8, pp. 141373-141394, 2020.

[87] SECREDAS, *Product Security for Cross Domain Reliable Dependable Automated Systems.*

[88] Y. Falcone, S. Krstić, G. Reger and D. Traytel, "A taxonomy for classifying runtime verification tools," *International Journal on Software Tools for Technology Transfer,* vol. 23, no. 2, pp. 255-288, 2021.

[89] G. Ehmen, B. Koopmann, Y. Bebawy and P. Ittershagen, "Measurement-based Online Verification of Timing Properties in Distributed Systems," in *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, 2020.

[90] E. Böde, M. Büker, W. Damm, G. Ehmen, M. Fränzle, S. Gerwinn, T. Goodfellow, K. Grüttner, B. Josko, B. Koopmann, T. Peikenkamp, F. Poppen, P. Reinkemeier, M. Siegel and I. Stierand, "Design Paradigms for Multi-Layer Time Coherency in ADAS and Automated Driving (MULTIC)," *FAT Series,* no. 302, 2017.

[91] G. Ehmen, K. Grüttner, B. Koopmann, F. Poppen, P. Reinkemeier and I. Stierand, "Coherent Treatment of Time in the Development of ADAS/AD Systems: Design Approach and Demonstration," in *SAE World Congress Experience (WCX'18)*, 2018.

[92] E. Böde, W. Damm, G. Ehmen, M. Fränzle, K. Grüttner, P. Ittershagen, B. Josko, B. Koopmann, F. Poppen, M. Siegel and I. Stierand, "MULTIC-Tooling," *FAT Series,* no. 316.

[93] M. García-Godillo, J. J. Valls and S. Sáez, "Heterogeneous Runtime Monitoring for Real-Time Systems with art2kitekt," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019.

[94] M. Kläs and L. Sembach, "Uncertainty wrappers for data-driven models," in *International Conference on Computer Safety, Reliability, and Security*, 2019.

[95] M. Kläs and L. Jöckel, “A framework for building uncertainty wrappers for AI/ML-based data-driven components,” in *International Workshop on Artificial Intelligence Safety Engineering (WAISE 2020)*, Lisbon, Portugal, 2020.

[96] M. Kläes, R. Adler, I. Sorokos, L. Joeckel and J. Reich, “Handling Uncertainties of Data-Driven Models in Compliance with Safety Constraints for Autonomous Behaviour,” in *17th European Dependable Computing Conference (EDCC)*, 2021.

[97] S. Nastic, T. Pusztai, A. Morichetta, V. C. Pujol, S. Dustdar, D. Vij and Y. Xiong, “Polaris Scheduler: Edge Sensitive and SLO Aware Workload Scheduling in Cloud-Edge-IoT Clusters,” in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, 2021.

[98] B. Beyer, C. Jones, J. Petoff and N. R. Murphy, Site Reliability Engineering: How Google runs production systems, O'Reilly Media, Inc., 2016.

[99] A. Cauševic, E. Lisova, M. Ashjaei and S. U. Ashgar, “On Incorporating Security Parameters in Service Level Agreements,” 2019.

[100 V. Stantchev and C. Schröpfer, “Negotiating and Enforcing QoS and SLAs in Grid and Cloud Computing,” in *International Conference on Grid and Pervasive Computing*, Berlin, Heidelberg, Springer, May 2009, pp. 25-35.

[101 N. Ibrahim and F. L. Mouel, “A Survey on Service Composition Middleware in Pervasive Environments,” 2009.

[102 N. Repp, A. Miede, M. Niemann and R. Steinmetz, “WS-Re2Policy: A policy language for distributed SLA monitoring and enforcement,” in *2008 Third International Conference on Systems and Networks Communications*, IEEE, October 2008, pp. 256-261.

[103 V. Beecher, D. Bradshaw, T. Das, V. Dinesh, A. Ghosh, M. Kennedy and D. Steiner, Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite, 11g Release 1.

[104 J. Singh, J. Bacon and D. Eyers, “Policy Enforcement within Emerging Distributed, Event-Based Systems,” in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, May 2014, pp. 246-255.

[105 E. Manoel, M. J. Nielson, A. Salahshour, S. S. KVL and S. Sudarshanan, “Problem Determination Using Self-Managing Autonomic Technology,” IBM International Technical Support Organization, 2005.

[106 P. Kochovski, V. Stankovski, S. Gec, F. Faticanti, M. Savi, D. Siracusa and S. Kum, “Smart Contracts for Service-Level Agreements in Edge-to-Cloud Computing,” *Journal of Grid Computing,* vol. IV, no. 18, pp. 673-690, 2020.

[107 K. Fu, W. Zhang, Q. Chen, D. Zeng, X. Peng, W. Zheng and M. Guo, “QoS-Aware and Resource Efficient Microservice Deployment in Cloud-Edge Continuum,” in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, May 2021, pp. 932-941.

[108 M. Satyanarayanan, “The Emergence of Edge Computing,” *Computer,* vol. I, no. 50, pp. 30-39, 2017.

[109 S. Nastic, A. Morichetta, T. Pusztai, S. Dustdar, X. Ding, D. Vij and Y. Xiong, “SLOC: Service Level Objectives for Next Generation Cloud Computing,” *IEEE Internet Computing,* vol. III, no. 24, pp. 39-50, 2020.

[110 T. Pusztai, S. Nastic, A. Morichetta, V. C. Pujol, S. Dustdar, X. Ding and Y. Xiong, “A Novel Middleware for Efficiently Implementing Complex Cloud-Native SLOs,” in *2021 IEEE 14th International COnference on Cloud Computing (CLOUD)*, 2021.

[111 C. Müller, M. Oriol, X. Franch, J. Marco, M. Resinas, A. Ruiz-Cortés and M. Rodríguez, “Comprehensive Explanation of SLA Violations at Runtime,” *IEEE Transactions on Services Computing,* vol. II, no. 7, pp. 168-183, 2013.

[112 Y. Xiong, Y. X. L. Sun and Y. Huang, “Extend Cloud to Edge with KubeEdge,” in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, IEEE, October 2018, pp. 373-377.

[113 Z. Guo, R. Liu, X. Xu and K. Yang, "A Survey of Real-Time Automotive Systems," Department of Computer Science Technical Report, University of North Carolina at Chapel Hill, 2015.

[114 J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," in *Real-Time Systems*, vol. 26, 2004, pp. 161-197.

[115 N. Stoimenov, S. Perathoner and L. Thiele, “Reliable Mode Changes in Real-Time Systems with Fixed Priority or EDF Scheduling,” in *Proceedings of the Conference on Design, Automation and Test in Europe*, European Design and Automation Association, 2009, pp. 99-104.

[116 N. Fisher and M. Ahmed, "Tractable real-time schedulability analysis for mode changes under temporal isolation," *IEEE Symposium on Embedded Systems for Real-Time Multimedia,* pp. 130-139, 2011.

[117 V. Nelis, B. Andersson, J. Marinho and S. Petters, “Global-EDF Scheduling of Multimode Real-Time Systems Considering Mode Independent Tasks,” in *2011 23rd Euromicro Conference on Real-Time Systems (ECRTS 2011)*, 2011, pp. 205-214.

[118 J. Goossens, X. Poczekajlo, A. Paolillo and P. Rodriguez, "ACCEPTOR: A Model and a Protocol for Real-Time Multi-Mode Applications on Reconfigurable Heterogeneous Platforms," in *Proceedings of the 27th International Conference on Real-Time Networks and Systems*, New York, NY: Association for Computing Machinery, 2019, pp. 209-219.

[119 H. Baek, K. G. Shin and J. Lee, “Response-Time Analysis for Multi-Mode Tasks in Real-Time Multiprocessor Systems,” *IEEE Access,* vol. 8, pp. 86111-86129, 2020.

[120 M. M. Villegas and A. H., “OTA updates mechanisms: a taxonomy and techniques catalog.,” *In XXI Simposio Argentino de Ingeniería de Software (ASSE 2020)-JAIIO 49 (Modalidad virtual).,* 2020.

[121 J. Bauwens, P. Ruckebusch, S. Giannoulis, I. Moerman and E. De Poorter, “Over-the-air software updates in the Internet of Things,” *An overview of key principles. IEEE Communications Magazine, 58(2),* pp. 35-41, 2020.

[122 B. Russell and S. e. a. Khemissa, “Recommendations for IOT firmware update processes. Cloud Security Alliance (CSA). Internet of Things (IOT) Working Group.,” 2018.

[123 N. S. Mtetwa, P. Tarwireyi, A. M. Abu-Mahfouz and M. O. Adigun, "Secure firmware updates in the Internet of Things: A survey.," *In 2019 International Multidisciplinary Information Technology and Engineering Conference (IMITEC),* pp. (pp. 1-7)., 2019.

[124 K. Zandberg, K. Schleiser, F. Acosta, H. Tschofenig and E. Baccelli, "Secure firmware updates for constrained iot devices using open standards: A reality check.," *IEEE Access, 7,* pp. 71907-71920, 2019.

[125 N. Asokan, T. Nyman, N. Rattanavipanon and A. Sadeghi, "ASSURED: Architecture for secure software update of realistic embedded devices.," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37(11),* pp. 2290-230, (2018).

[126 P. Ruckebusch, E. De Poorter, C. Fortuna and I. Moerman, "Gitar: Generic extension for internet-of-things architectures enabling dynamic updates of network and application modules.," *Ad Hoc Networks, 36,* pp. 127-151, 2016.

[127 [Online]. Available: https://wiki.unece.org/pages/viewpage.action?pageId=40829521.

[128 A. Consortium, "Specification of Update and Configuration Management," 2017.

[129 T. K. Kuppusamy, A. Brown, S. Awwad, D. McCoy, R. Bielawski, C. Mott, S. Lauzon, A. Weimerskirch and J. Cappos, "Uptane: Securing Software Updates for Automobiles.," *In 14th Embedded Security in Cars (Escar) Eu,* 2016.

[130 [Online]. Available: https://www.esyncalliance.org/..

[131 [Online]. Available: https://www.esyncalliance.org/downloads/.

[132 H. Dakroub and R. Cadena, "Analysis of Software Update in Connected Vehicles.," *SAE International Journal of Passenger Cars-Electronic and Electrical Systems 7 (2014-01-0256),* p. 411–17, 2014.

[133 [Online]. Available: http://www.ros.org.

[134 [Online]. Available: https://github.com/ros2/ros2/wiki.

[135 [Online]. Available: https://www.escrypt.com/en/solutions/secure-firmware-updates.

[136 [Online]. Available: https://www.infineon.com/cms/en/applications/automotive/automotive-security/sotware-update-over-the-air/.

[137 [Online]. Available: http://sbabic.github.io/swupdate/.

[138 [Online]. Available: https://ostree.readthedocs.io.

[139 [Online]. Available: https://mender.io/.

[140 [Online]. Available: https://uptane.github.io/.

[141 T. K. Kuppusamy, A. Brown, S. Awwad, D. McCoy, R. Bielawski, C. Mott, S. Lauzon, A. Weimerskirch and J. Cappos, "Uptane: Securing Software Updates for Automobiles".*The 14th escar Europe (escar EU 2016), Munich, Germany.*

[142 G. Kornaros, E. Wozniak, O. Horst, N. Koch, C. Prehofer, A. Rigo and M. Coppola, "Secure and Trusted Open CPS Platform," *Handbook of Research on Solutions for Cyber-Physical Systems Ubiquity, IGI Global,* pp. 301-324, 2018.

[143 G. Kornaros and S. Leivadaros, "Securing Dynamic Firmware Updates of Mixed-Critical Applications," *3rd IEEE International Conference on Cybernetics (CYBCONF),* 2017.

[144 [Online]. Available: https://stepup-cps.de/.

[145 [Online]. Available: https://cordis.europa.eu/project/id/871465.

[146 [Online]. Available: https://www.eclipse.org/4diac/.

[147 M. Brinkmann, E. Böde, A. Lamm, S. Vander Maelen and A. Hahn, "Learning from Automotive: Testing Maritime Assistance Systems up to Autonomous Vessels.," in *In OCEANS 2017 - Aberdeen, 1–8.*, 2017.

[148 T. Kimberly and K. D. Jones, "Maritime Cybersecurity Policy: The Scope and Impact of Evolving Technology on International Shipping.," *Journal of Cyber Policy 3 (2),* p. 147–64, 2018.

[149 C. J. W. Group., *Industry Standard on Software Maintenance of Shipboard Equipment V 1.0,* 2017.

[150 A. Karlsen, "D-Class Data Smart Classification.," in *In Proceedings of the Dynamic Positioning Conference. Houston, Texas.*, 2018.

[151 A. Hahn, S. Feuerstack, B. Weinert, N. Rüssmeier, W. Grafe and C. Cabos., *Maritime Safety and Highly Automated Systems - An Industrial Technology Roadmap.,* Institut für Informatik OFFIS e. V. https://www.emaritime.de/w, 2019.

[152 E. Y. Nakagawa and P. O. Antonio, Reference Architectures for Critical Domains: Industrial Uses and Impacts, Springer, 2022.

[153 M. Staron, Automotive Software Architectures: An Introduction, Springer, 2017.

[154 D. Reinhardt and M. Kucera, "Domain Controlled Architecture - A New Approach for Large Scale Software Integrated Automotive Systems," in *PECCS*, 2013.

[155 AUTOSAR, "Adaptive Platform," [Online]. Available: https://www.autosar.org/standards/adaptive-platform.

[156 H. Kevin Driscoll, “Integrated modular avionics (IMA) requirements and development,” in *ARTIST2 Workshop on Integrated Modular Avionics*, Rome, Italy, November 12-13, 2007.

[157 J. Rushby, "J. Rushby.: Partitioning in avionics architectures: Requirements, mechanisms, and assurance," NASA Langley Technical Report Server, 1999.

[158 F. Boniol, “F. Boniol.: New Challenges for Future Avionic Architectures,” in *Modeling Approaches and Algorithms for Advanced Computer Applications*, Springer, 2013, pp. 1-1.

[159 "DO-297, Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations," Radio Technical Commission for Aeronautics (RTCA), 2005.

[160 C. B. Watkins and R. Walter, "Transitioning from federated avionics architectures to Integrated Modular Avionics," in *2007 IEEE/AIAA 26th Digital Avionics Systems Conference*, 2007.

[161 E. Y. Nakagawa, P. O. Antonino, F. Schnicke, R. Capilla, T. Kuhn and P. Liggesmeyer, "Industry 4.0 reference architectures: State of the art and future trends," *Computers & Industrial Engineering,* vol. 156, p. 107241, 2021.

[162 *DIN specification "91345: 2016-04 Reference Architecture Model Industrie 4.0 (RAMI4. 0),* 2016.

[163 Industrial Value Chain Initiative, "Industrial Value Chain Reference Architecture," 2018.

[164 International Electrotechnical Commission, “IEC 62264-1 Enterprise-control system integration–Part 1: Models and terminology,” IEC, Genf, 2003.

[165 L. Kassner, C. Gröger, J. Königsberger, E. Hoos, C. Kiefer, C. Weber, S. Silcher and B. Mitschang, "The Stuttgart IT Architecture for Manufacturing. An Architecture for the Data-Driven Factory," in *IECIS*, 2016.

[166 M. Resman, M. Pipan, M. Šimic and N. Herakovič, "A new architecture model for smart manufacturing: A performance analysis and comparison with the rami 4.0 reference model," *Advances in Production Engineering & Management,* vol. 14, pp. 1-13.

[167 Industrial Internet Consortium, “Industrial Internet Reference Architecture (IIRA),” 2019.

[168 “Basys4.0 Website,” [Online]. Available: https://www.basys40.de/ .

[169 “Arrowhead Website,” [Online]. Available: https://www.arrowhead.eu/.

[170 F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M. Stal, Pattern-Oriented Software Architecture, Volume 1, A System of Patterns, Chichester, UK: Wiley, 1996.

[171 D. Schmidt, M. Stal, H. Rohnert and F. Buschmann, Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects, Chichester, UK: Wiley.

[172 M. Kircher and P. Jain, Pattern-Oriented Software Architecture, Volume 3: Patterns for Resource Management, Chichester, UK: Wiley.

[173 F. Buschmann, K. Henney and D. Schmidt, Pattern-Oriented Software Architecture, Volume 4: A Pattern Language for Distributed Computing, Chichester, UK: Wiley.

[174 R. N. Taylor, N. Medvidovic and E. Dashofy, Software Architecture: Foundations, Theory, and Practice, Hoboken, Wiley, 2010.

[175 W. Wu and T. P. Kelly, "Safety Tactics for Software Architecture Design," in *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004*, 2004.

[176 B. P. Douglass, Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems, USA: Addison-Wesley Longman Publishing Co., Inc..

[177 W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal,* vol. 3, no. 5, pp. 637-646, 2016.

[178 C. Guidi, I. Lanese, M. Mazzara and F. Montesi, "Microservices: a Language-based Approach," in *Present and Ulterior Software Engineering*, M. Mazzara and B. Meyer, Eds., Springer International Publishing, 2017, pp. 217-225.

[179 R. Petrasch, "Model-based engineering for microservice architectures using Enterprise Integration Patterns for inter-service communication," in *2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, NakhonSiThammarat, Thailand, 2017.

[180 F. Rademacher, J. Sorgalla, P. Wizenty, S. Sachweh and A. Zündorf, "Graphical and Textual Model-Driven Microservice Development," in *Microservices: Science and Engineering*, A. Bucchiarone, N. Dragoni, S. Dustdar, P. Lago, M. Mazzara, V. Rivera and A. Sadovykh, Eds., Springer International Publishing, 2020, pp. 147-179.

[181 B. Terzić, V. Dimitrieski, S. Kordić, G. Milosavljevic and I. Luković, "MicroBuilder: A Model-Driven Tool for the Specification of REST Microservice Architectures," in *In Proc. of the 7th Int. Conf. on Information Society and Technology (ICIST)*, 2017.

[182 B. el Khalyly, A. Belangour, A. Erraissi and M. Banane, "Devops and Microservices Based Internet of Things Meta-Model," *International Journal of Emerging Trends in Engineering Research,* vol. 8, pp. 6254-6266, September 2020.

[183 V. Yussupov, U. Breitenbücher, C. Krieger, F. Leymann, J. Soldani and M. Wurster, "Pattern-based Modelling, Integration, and Deployment of Microservice Architectures," in *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, Eindhoven, Netherlands, 2020.

[184 M. De Sanctis, H. Muccini and K. Vaidhyanathan, "Data-driven Adaptation in Microservice-based IoT Architectures," in *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*, Salvador, Brazil, 2020.

[185 P. Chico de Guzmán, F. Gorostiaga and C. Sánchez, "Pipekit: A Deployment Tool with Advanced Scheduling and Inter-Service Communication for Multi-Tier Applications," in *2018 IEEE International Conference on Web Services (ICWS)*, San Francisco, CA, USA, 2018.

[186 A. R. Sampaio, J. Rubin, I. Beschastnikh and N. S. Rosa, “Improving microservice-based applications with runtime placement adaptation,” *Journal of Internet Services and Applications,* vol. 10, no. 1, p. 4, 2019.

[187 P. P. Jayaraman, C. Perera, D. Georgakopoulos, S. Dustdar, D. Thakker and R. Ranjan, “Analytics-as-a-Service in Multi-Cloud Environment through Semantically-enabled Hierarchical Data Processing,” *Software: Practice and Experience,* vol. VIII, no. 47, pp. 1139-1156, 2017.

[188 X. Xu, S. Huang, L. Feagan, Y. Chen, Y. Qiu and Y. Wang, “EAaaS: Edge Analytics as a Service,” in *2017 IEEE International Conference on Web Services (ICWS)*, IEEE, June 2017, pp. 349-356.

[189 Z. Sun, H. Zou and K. Strang, “Big Data Analytics as a Service for Business Intelligence,” in *Conference on e-Business, e-Services and e-Society*, Cham, Springer, October 2015, pp. 200-211.

[190 B. Marr, “Big Data-as-a-Service is next big thing,” April 2019. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2015/04/27/big-data-as-a-service-is-next-big-thing/. [Accessed 24 September 2021].

[191 C. A. Ardagna, P. Ceravolo and E. Damiani, “Big Data Analytics-as-a-Service: Issues and Challenges,” in *2016 IEEE International conference on Big Data*, IEEE, December 2016, pp. 3638-3644.

[192 A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications,” *IEEE communications surveys & tutorials,* vol. IV, no. 17, pp. 2347-2376, 2015.

[193 F. Mehdipour, B. Javadi, A. Mahanti, G. Ramírez-Prado and E. C. Principles, “Fog Computing Realization for Big Data Analytics,” in *Fog and edge computing: Principles and paradigms*, 2019, pp. 259-290.

[194 F. Bonomi, R. Milito, J. Zhu, Addepalli and S., “Fog Computing and its Role in the Internet of Things,” in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, August 2012, pp. 13-16.

[207 “OpenFog Reference Architecture,” 2017. [Online]. Available: https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf. [Accessed 28 09 2021].

[208 AUTOSAR, "Classic Platform," [Online]. Available: https://www.autosar.org/standards/classic-platform.