Dependable Mesh Networking Patterns

Jürgen Dobaj
Graz University of Technology
Graz, Austria
juergen.dobaj@tugraz.at

Markus Schuss
Graz University of Technology
Graz, Austria
markus.schuss@tugraz.at

Michael Krisper
Graz University of Technology
Graz, Austria
michael.krisper@tugraz.at

Carlo Alberto Boano
Graz University of Technology
Graz, Austria
cboano@tugraz.at

Georg Macher
Graz University of Technology
Graz, Austria
georg.macher@tugraz.at

ABSTRACT

In our daily life, we are increasingly relying on connected systems ranging from smart health care devices to industrial and intelligent transportation systems, as well as smart homes and cities. The unavailability or malfunctioning of these systems could threaten human life, cause environmental damage, and significant financial loss. To prevent such large scale and mission-critical systems from malfunctioning, it is of utmost importance to establish and guaranty reliable connections to attain a dependable networked system. Generally, mesh networking technologies are used for building such systems since mesh networks provide the best performance characteristics regarding fault-tolerance, throughput, resource usage, and service level flexibility.

In this paper, we summarize the major challenges in dependable network design, to subsequently present three patterns that approach redundancy on the hardware level, software-defined networking, and cross-cutting concerns like monitoring and service discovery within distributed networked systems. These three patterns should help designers and engineers in choosing the appropriate technologies for building dependable networked systems at all scales. Since dependable network engineering requires a holistic system-wide design and engineering approach, we also present a pattern map guiding to complementary and closely related patterns. System architects and system engineers responsible for building mixed-criticality systems, internet-of-things (IoT), and industrial Internet-of-Things (IIoT) systems are the target audience of the patterns presented in this paper.

CCS CONCEPTS

• **Applied computing** → **Service-oriented architectures**; **Command and control**; • **Software and its engineering** → *Ultra-large-scale systems*;

EuroPLoP '19, July 3–7, 2019, Irsee, Germany

ACM ISBN 978-1-4503-6206-1/19/07…$15.00
https://doi.org/10.1145/3361149.3361174

KEYWORDS

service mesh pattern, logical mesh pattern, physical mesh pattern, network architecture, network design, cloud architecture, microservices, SoA, service-oriented architecture, IaaS, infrastructure-as-a-service, dependability, IoT, IIoT, Industry 4.0

ACM Reference Format:
Jürgen Dobaj, Markus Schuss, Michael Krisper, Carlo Alberto Boano, and Georg Macher. 2019. Dependable Mesh Networking Patterns. In *24th European Conference on Pattern Languages of Programs (EuroPLoP '19), July 3–7, 2019, Irsee, Germany.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3361149.3361174

1 INTRODUCTION

We increasingly rely on highly inter-connected networked systems in our daily life, ranging from intelligent transportation systems [15] and industrial systems [46] to smart homes and smart cities [14]. Since we are relying on these critical systems in such multitude manner, it is of utmost importance to establish trust into these systems, summarized in the statement that these systems should be dependable [2]. Dependability is defined by multiple attributes (availability, reliability, safety, confidentiality, integrity, maintainability) that must be maintained and assured at a sufficient level. This is commonly achieved by considering the risk of potential threats (faults, errors, failures), followed by applying adequate risk reduction mechanisms (fault prevention, fault tolerance, fault removal, fault forecasting).

Challenge of the reliable and dependable system and service connectivity. The Threat Horizon 2017 report [17], for example, lists *"death from disruption to digital services"* as one potential threat, especially in the medical and mobility domain. The newest Threat Horizon 2019 report [21] claims that the cause of service disruption may come *"from an over-reliance on fragile connectivity"*. In order to reduce the risk of digital service disruption, it is, therefore, essential to establish and guaranty both, reliable and dependable inter-connections of the devices within mission-critical networked systems.

Challenge of highly dynamic and flexible connectivity. By introducing internet-of-things (IoT) and cyber-physical system (CPS) concepts into multiple industrial domains, the industry is undergoing enormous change towards globalizing, flexibilizing, decentralizing, and digitizing manufacturing processes [27, 41, 42]. This trend is embodied under the *Industry 4.0* revolution (which

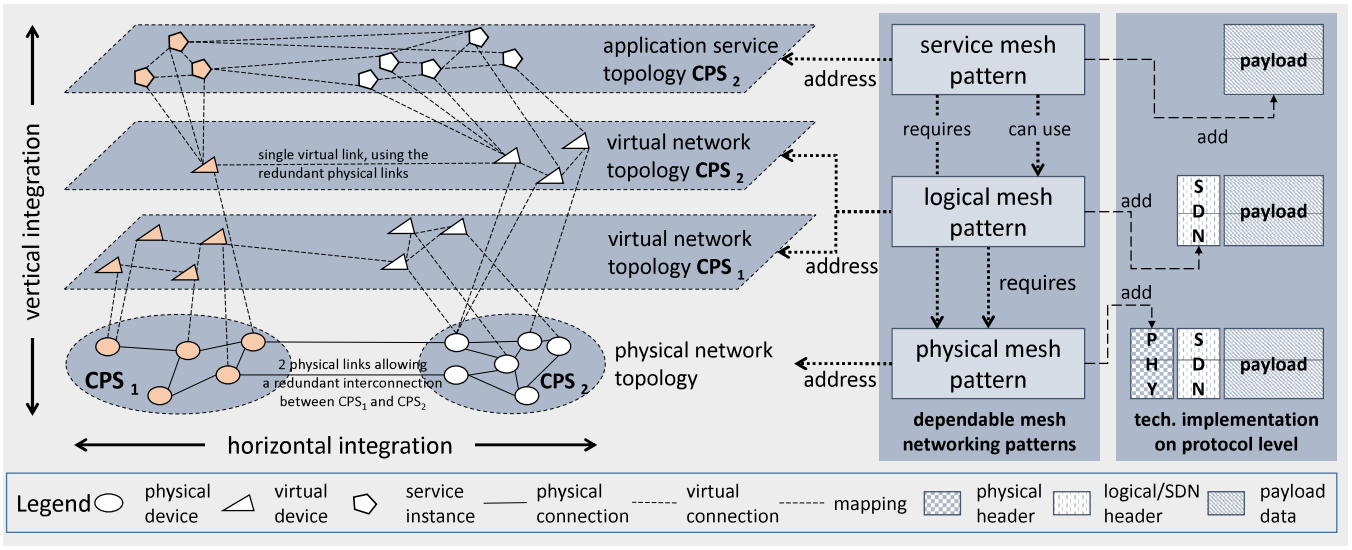

Figure 1: Illustration of the dependable networking context, which can be separated into three levels: the physical, logical, and service level. Each level is addressed by one pattern presented in this paper.

is in the process of happening) driving the need for cross-domain interaction, highly dynamic system change, and graceful system evolution. However, traditional production and automation systems are designed as static silos, which are generally configured once, to support the (dependable) information processing within the specific system, but rarely across system and domain boundaries [33].

Need for holistic and system-wide design and engineering approaches. The increasing need for highly dynamic and inter-domain production and automation systems pushes the demand for solutions that facilitate a multitude of attributes, including: (i) **system interoperability**, to support the seamless vertical and horizontal interaction and integration of heterogeneous devices and systems across level and domain boundaries; (ii) **system autonomy**, to successfully manage the complexity and dynamics of largely distributed CPS, where (timely) human intervention is extremely limited; and (iii) **system dependability**, to ensure the reliable and safe interaction between the cyber and the physical world. The building, operation, and maintenance of such large-scale networked systems are rather complex because such systems generally consist of various networking technologies. These networking technologies specify how the network elements communicate with each other by forming an organizational framework consisting of information-, routing-, and communication protocols. This framework directly influences the interconnection (i.e., topology) of the network elements and the traffic flow through the network [26]. **Therefore, a holistic system-wide design and engineering approach is essential for appropriately choosing both, the network technologies and the network topology.**

Our contribution: Dependable Mesh Networking Patterns. In this paper, we present three architectural patterns: (1) the PHYSICAL MESH PATTERN, (2) the LOGICAL MESH PATTERN, and (3) the SERVICE MESH PATTERN. Each pattern addresses the challenges mentioned above at specific technical levels with its focus on dependable networking. Additionally, each pattern establishes the context for the application of various related patterns; hence, providing a solid starting point for a holistic system-wide design and engineering approach. Figure 1 shows the technical context and the relationship between the patterns presented in this paper, while the pattern map in Figure 2 sets the context and connections to related patterns.

The remainder of this paper is organized as follows. In Section 2, we introduced the basic design principles in dependability engineering, followed by the explanation of typical characteristics, qualities, and requirements of modern mixed-criticality networked systems in Section 3. In Section 4 two patlets are presented, which are used by the subsequently epxlained PHYSICAL MESH PATTERN in Section 5, the LOGICAL MESH PATTERN in Section 6, and the SERVICE MESH PATTERN in Section 7. A final conclusion is given in Section 8.

1.1 Related patterns

The discussed and presented design space in this work is only an excerpt of the overall design space, since networking and dependability are addressing huge system engineering and research fields. In the following, we provide a list of pattern languages and pattern books that we think are closely related to the topics discussed in this work. The relationships between these patterns are shown in Figure 2.

Networking and cloud-computing patterns. The following patterns focus on building distributed service-oriented applications, the coordination between the services, and the efficient management of data and computing resources.

- Engineering software for the cloud [5, 38–40]
- Patterns for software orchestration on the cloud [6]
- A pattern language for microservices [32]
- A pattern language for distributed computing [7]

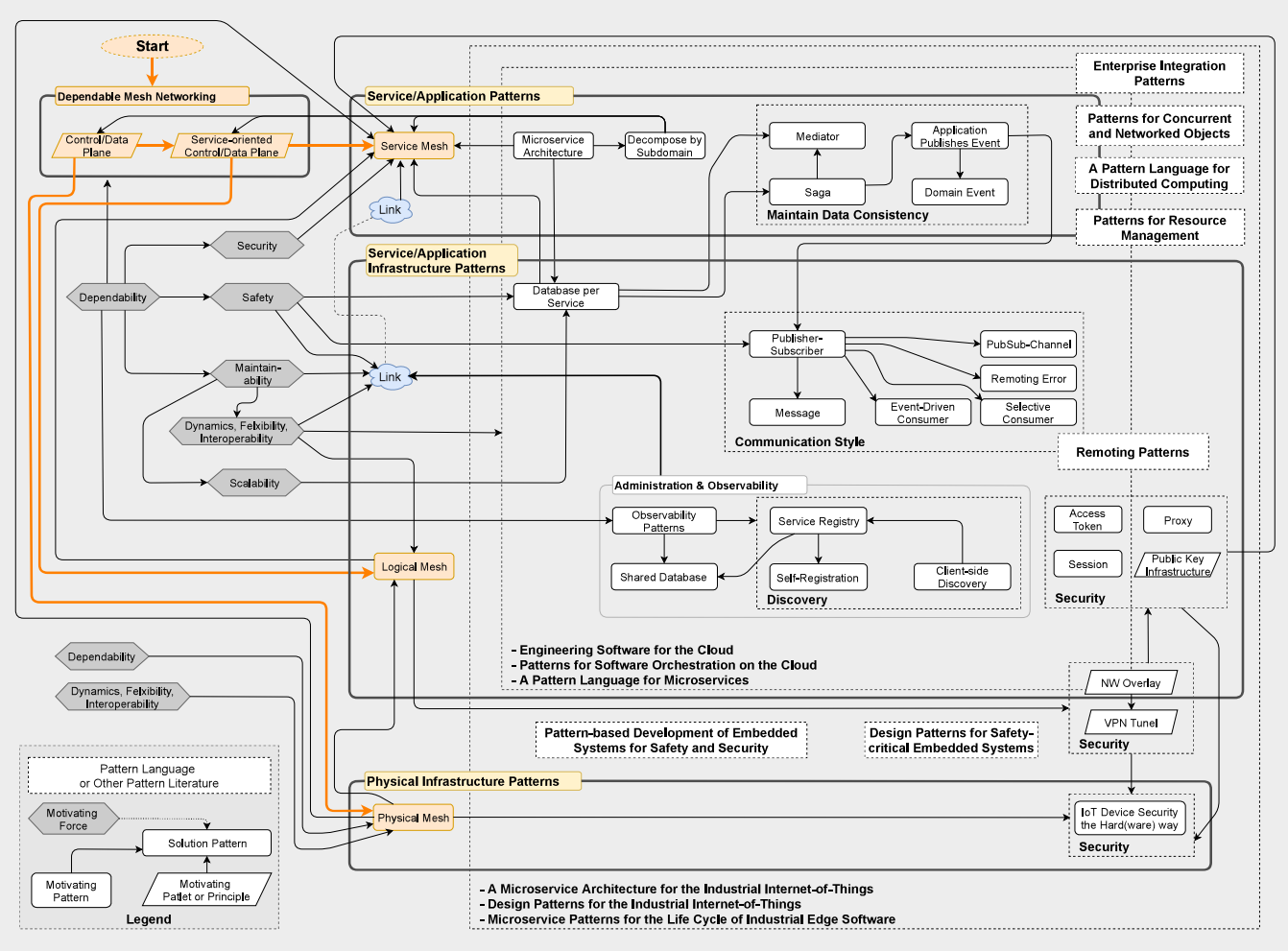

Figure 2: Illustration of a pattern map showing an excerpt of related patterns and pattern languages. The references to the shown patterns are provided in Section 1.1.

- Patterns for concurrent and networked objects [34]
- Patterns for parallel programming [28]
- Patterns for Resource Management [22]
- Remoting patterns: design reuse of distributed object middleware solutions [47]
- Enterprise integration patterns[1]
- Messaging patterns[2]

IoT and edge-computing patterns. The following patterns focus on the integration and interconnection of distributed and embedded and resource constraint devices, which are generally used to connect the physical world with the cyber world in the context of Industry 4.0 applications and CPS.

- Microservice patterns for the life cycle of industrial edge software [24]
- Design patterns for the industrial Internet of Things [4]
- A microservice architecture for the industrial internt-of-things [8]
- IoT Device Security the Hard(ware) way [35]
- IoT design patterns: computational constructs to design, build and engineer edge applications [31]
- Pattern-based development of embedded systems for safety and security [30]
- Design patterns for safety-critical embedded systems [1]
- On patterns for decentralized control in self-adaptive systems [43]

2 DESIGN RATIONALE

In general, various means are required, and various aspects must be considered to attain a dependable networked system. In this section, we briefly introduce these aspects, describing the rationale behind dependable mesh networking and the patterns presented in this paper. Section 3 provides a more detailed discussion of dependable mesh networking.

[1] https://www.enterpriseintegrationpatterns.com/
[2] https://docs.microsoft.com/en-us/azure/architecture/patterns/category/messaging

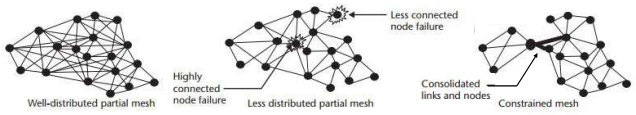

Figure 3: Variations of mesh topologies (adapted from [26]).

2.1 Mesh Networking in the Context of Dependable System Engineering

In this work, we specifically focus on the networking aspects that must be considered when building large-scale mission-critical CPS. One of the vital design-principle is **redundancy**, which is generally applied for obtaining highly available and reliable networks and systems. The principle of **diverse redundancy** is commonly applied in safety-critical domains to reduce the risk of common cause failures. Safety-critical and mission-critical systems, often require the implementation of **fault-tolerant mechanism**. In the case of distributed or large-scale systems, where (timely) human intervention is limited, it is generally necessary to achieve a certain level of **system autonomy** to automatically or autonomously execute **fail-operational or self-healing mechanisms** for ensuring continues system operation.

Compared to all other network topologies, **the mesh topology provides the best performance characteristics regarding fault-tolerance, resource usage, service flexibility, as well as throughput** [26], which is the primary reason for presenting dependable mesh networking patterns in this work. Another advantage of mesh typologies is that they can be incrementally built, allowing to start with simpler network topology, and whenever the network traffic increases or additional redundancy is required, links and nodes can be dynamically added. Figure 1 shows such a mesh topology, where redundant links enable the establishment of multiple routes between a shared set of nodes. These links can either be used to increase the overall throughput of the network or to enhance the fault-tolerance of the network.

Despite all these advantages, the mesh topology is typically the most expensive topology to implement [26], because a mesh topology requires the highest amount of resources compared to all other topologies (e.g., bus, ring, start), which is manifested in the high amount of nodes and links. To this fact, the scalability of mesh topologies is limited, and since every hop in the network adds additional communication latency, mesh networked systems generally require additional intelligence to ensure the efficient traffic routing and guaranteed end-to-end (e2e) quality of service (QoS) provisioning.

In order to address these constraints, variations of mesh topologies have been defined, which are shown in Figure 3. The degree of difference between mesh topology variations is usually measured in terms of the distribution of connected nodes [26].

- In a flat mesh, the node distribution is uniform.
- In a partial mesh, the distribution of connected nodes is less uniform, which might result in a network breakdown, if a highly connected node fails.
- In a constrained or bounded mesh, links, and nodes are consolidated, allowing the re-routing of traffic in the case of a failure, only by using the remaining consolidated links and nodes as a fallback.

The node and link consolidation in a constraint mesh topology generally reduces the implementation cost of the mesh network by still providing a sufficient level of fault-tolerance. Constrained mesh topologies are therefore more typical in practice since they provide a balance between cost and performance [26]. The example in Figure 1 is also a depiction of such a constrain mesh network, where two physical links between CPS_1 and CPS_2 are consolidated on the virtual network layers for redundancy reasons.

In summary, it can be said that a holistic system-wide design and engineering approach is essential for efficiently building dependable mesh networked CPS. Therefore, this paper presents three patterns: the PHYSICAL MESH PATTERN, the LOGICAL MESH PATTERN, and the SERVICE MESH PATTERN. Each pattern addresses design challenges on another technical level and topology layer, which further establishes the context for related patterns, effectively facilitating holistic system engineering.

2.2 Pattern and Patlet Mining

The patterns and patlets presented in the remainder of this work were mined out of system architectures that are commonly used in various networking domains including the Internet, telecommunications, web services, and cloud computing [6, 11, 12, 16, 37], as well as from IoT, industrial Internet-of-Things (IIoT), and Industry 4.0 production and automation systems [8, 24, 27, 41, 42]. In these domains the **"separation of policy from mechanisms"**-principle is broadly used for organizing and structuring the network-centric and cloud-based computing systems, for facilitating the management and operation of the distributed (network-)infrastructure. The most prominent examples are the software defined networking (SDN) principle, and the service mesh principle. The SDN principle is heavily used by service providers in the Internet and telecommunications domains for infrastructure management and network service provisioning to multiple tenancies. The service mesh principle, on the other hand, is a relatively new approach, which has its origin in the cloud-computing domain, where service meshes are used to mediate, configure, monitor, and control the in-coming and out-going data traffic of all services that participate in the service mesh.

3 DEPENDABLE MESH NETWORKING

In Section 2.1, we introduced the basic design principles in dependability engineering for distributed systems, as well as the importance of redundancy and fault-tolerance mechanisms, which are best supported by mesh network topologies. In this section, we provide a more detailed discussion of dependability engineering, summarizing the state-of-the-art and recent trends that can be observed in industry and research.

In Section 3.1, we first explain the characteristics of a modern mixed-criticality networked system. In Section 3.2, we then explain the system requirements and system qualities that must be attained in order to achieve the desired system characteristics.

3.1 System Characteristics

Service-oriented system architectures. Nowadays, modern large-scale applications are no longer developed as monoliths. Instead, their design is based on a distributed service-oriented architecture (SoA)-style facilitating multiple dependability attributes such as system availability and reliability, as well as system scalability and maintainability. The best-known examples of such systems are modern cloud-deployed applications, which use containerization and virtualization in combination with microservice and serverless architectures [4, 32, 38, 39]. The ever increasing demand for connectivity is the primary driver why these design principles are no longer limited to cloud applications only. By introducing IoT and CPS concepts into multiple industrial domains, the industry is undergoing enormous change towards highly interconnected and globally distributed automation and control systems [10, 23, 25, 29, 44, 45]. Following this trend, the industry is facing interoperability challenges between devices and systems driving the demand for flexible and dynamic communication infrastructure [8, 44].

System dynamics. Traditionally, the configuration of automation and control systems did not significantly change during operation. However, with the introduction of IoT concepts, the upcoming CPS become highly dynamic, embodied in loosely connected devices that come together as temporary configurations of smaller systems, which again dissolve and give place for new configurations making the number of configurations over the system lifetime unknown and potentially infinite. Therefore, the implementation of automated (and even autonomous) system adaption becomes necessary to cope with the increasing system dynamics and potentially unexpected behavior and changes. Especially in the context of large-scale, distributed, and mission-critical systems, where (timely) human intervention in the case of failures is limited, it is vital to have autonomous self-healing mechanism in place to guaranty continues system operation.

System autonomy. Modern large-scale and mission-critical systems become ever more complicated due to the increasing number of connections and inter-domain dependencies. Handling this system complexity often requires the autonomous (no human in the loop) establishment and maintenance of reliable machine-to-machine (M2M) and service-to-service (S2S) communication channels to guaranty system dependability and the stringent QoS requirements. These requirements include the timely and guaranteed delivery of data packets within the networked system, even over unreliable channels, with guaranteed latency boundaries and high throughput ratios.

3.2 System Qualities and Requirements

Dependability. Dependability is defined by multiple attributes that must be maintained and assured at a sufficient level, including (i) availability, the readiness for correct service; (ii) reliability, the continuity of correct service; (iii) safety, the absence of catastrophic consequences on the user(s) and the environment, (iv) integrity, the absence of improper system alterations;(v) maintainability, the ability to undergo modifications and repairs; and (vi) confidentiality, the absence of unauthorized disclosure of information. These attributes can be only attained if dependability engineering is seen as a holistic system-wide engineering approach considering fault prevention means, for preventing the occurrence or introduction of faults; fault tolerance means, for avoiding service failures in the presence of faults; as well as fault removal means, for reducing the number and the severity of faults [3].

In dependability engineering, especially the idea of safety and security co-design has become a significant trend of recent publications. Moreover, this trend is expected to appear more often in the future, since the upcoming security standards for safety-critical domains, and the requirements on communication and coordination between safety and security (i.e., safety vs. availability, integrity, confidentiality). Like dependability engineering, safety and security engineering both focus on system-wide features and need to be adequately addressed during system design, since they frequently appear to be in total contradiction to the overall system features.

Formerly, security played only a secondary role in industrial automation systems, since these systems were not connected to the Internet or the outer world. Future IIoT systems, however, will be globally distributed and will rely on private and public cloud infrastructures to realize cross-domain services, which puts security at the center of IIoT systems and the Industry 4.0 era. Hence, the integration of security mechanisms across all system hierarchy levels is required to prevent unwanted access or malicious attacks on these systems.

Failure detection and failure recovery. Although failure detection and failure recovery are already (partially) targeted by the dependability force, it is worth to mention that in the distributed system context it is of utmost importance to prevent a network or service failure from cascading to other services.

Scalability. A trend in the Industry 4.0 era is the utilization of advanced data analytics tools to realize entirely new thinking about production management and factory transformation [23]. By installing appropriate sensors, various signals (e.g., vibration, pressure, heat development) can be extracted from the process under control for predictive analytics, product optimization, and more "informed" strategic decisions. The resource-intensive data analytics tools generally operate in cloud environments, which requires efficient data transfer from the field and edge devices to cloud systems. While these concepts are currently under development in industrial application domains, the consumer industry has already implemented these concepts using sensors integrated into smart wearables.

Another particular scalability challenge of IoT and industrial automation systems lies in the application itself: As more and more data points can be captured, smart and scalable data processing concepts are required to master the data exchange, storage, and processing efficiently, all in a distributed manner [33].

Interoperability and Connectivity. Supporting efficient system-wide communication in large-scale and mission-critical networked systems is challenging since these systems are equipped with different hardware resources, may operate at different frequencies and under different timing constraints. Additionally, the industry is facing interoperability challenges between devices and systems due to the market and technology fragmentation. The main

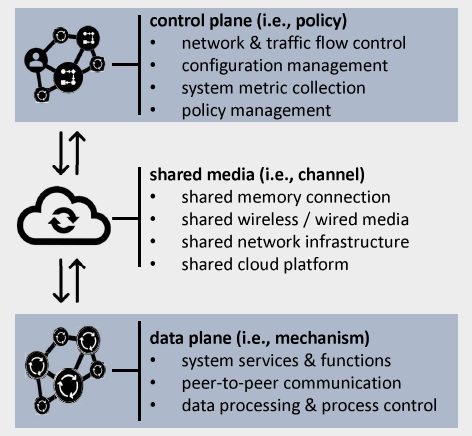

Figure 4: Concept of the control/data plane patlet in the context of dependable CPS networks.

issues that must be addressed are the horizontal interoperability between the hierarchy levels within one business domain, as well as the vertical interoperability between different business domains.

Graceful system evolution. This vertical and horizontal interoperability is exemplified in the interconnected CPS scenario shown in Figure 1. In such scenarios, it is essential that the systems can efficiently adapt to future changes in both technology and system/-functional requirements. This necessity of adaptation might arise due to technology advances, standardization, dynamic device integration unknown at the design time, and due to security flaws, which might result in changes of the communication infrastructure, protocols, or the software architecture itself. These adaptations may be performed manually by system developers or in an automated fashion by implementing continuous integration (CI)/continuous deployment (CD) pipelines and by supporting self-adaptiveness.

Architectural and behavioral self-adaptation, together with CI and CD will play a significant role in future IIoT systems to facilitate efficient system changes, updates, and graceful system evolution, while still maintaining dependability [9, 19, 20, 23].

4 DEPENDABLE MESH NETWORKING PATLETS

In this section we present two patlets, which are used by the subsequently explained PHYSICAL MESH PATTERN, the LOGICAL MESH PATTERN, and the SERVICE MESH PATTERN. The two patlets are based on the *"divide and conquer"-principle* or the *"separation of policy from mechanisms"-principle*, which are established design-principles in the Internet and telecommunications domain [11, 12].

4.1 The Control/Data Plane Patlet

In order to ensure the reliable and dependable operation of a distributed system, it is essential to support numerous cross-cutting concerns. To minimize the effects of cross-cutting concerns and to obtain a loose coupling between the actual system implementation and its management functions, **apply the architectural principle - *"separation of policy from mechanisms"***, which splits the networked system into two isolated planes: the control plane and the data plane. Figure 4 illustrates the CONTROL/DATA PLANE PATLET in the context of networked systems, highlighting the different responsibilities of

- the **control plane**, the policy;
- the **data plane**, the mechanism; and
- the **shared media**, the communication channel connecting the two planes.

The *"layers pattern"* [7] and the *"separation of processing and coordination in computer systems pattern"* [18] are two closely related patterns, which are also based on the *"divide and conquer"-principle* or the *"separation of policy from mechanisms"-principle.*

4.2 The Service-oriented Control/Data Plane Patlet

The SERVICE-ORIENTED CONTROL/DATA PLANE PATLET extends the CONTROL/DATA PLANE PATLET with service-oriented features, providing a generic service-oriented framework that allows to apply the **"separation of policy from mechanism"** design principle on both, hardware and software level. Figure 5 illustrates this framework, which also consists of two planes:

- the data plane, comprised of several distributed and interconnected (hardware/software) components or services; and
- the control plane, comprised of several distributed (hardware/software) controllers and an (optional) orchestrator service.

All components are interconnected via (service-oriented) interfaces abstracting the implementation details of the partners, enabling a loose coupling between the components. The responsibility of the orchestrator service is the coordination and management of the controllers on the control plane, which themselves are responsible for configuring, monitoring, and managing the (hardware/-software) components on the data plane. Depending on the timing constraint imposed on the controller functionality, the framework distinguishes between a real-time controller, capable of executing a certain control command within a guaranteed time window, and a (non real-time) controller providing no guaranties about execution and response times to requested control commands.

5 THE PHYSICAL MESH PATTERN: APPROACH REDUNDANCY ON HARDWARE LEVEL

5.1 Context

You are designing the physical network topology of a distributed, large-scale, or mission-critical system, where highly reliable and dependable network and system operation is essential.

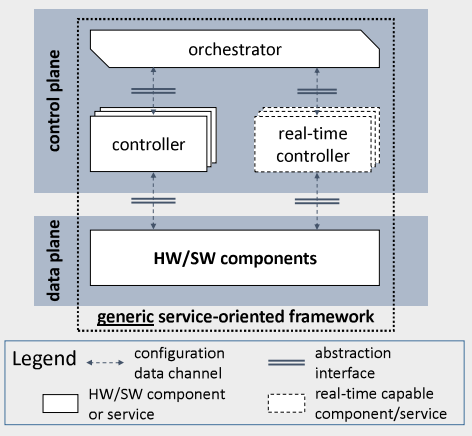

Figure 5: Concept of the generic service-oriented control/-data plane patlet.

5.2 Problem

In order to guarantee system reliability and dependability, it is necessary to establish dependable communication channels on the physical network level. These dependable channels must be established between various heterogeneous devices that share a common (heterogeneous) physical media, network infrastructure, or computing platform.

5.2.1 Forces.

- **Resilience:** The loss of a single connection must not cause communication disruptions or failures that propagate through the whole network.
- **Performance:** Multiple connections should be aggregated transparently into a single connection providing higher throughput.
- **Unattended Operation:** The above operations must be performed continuously and autonomously according to a given configuration (e.g., use one out of four available connections, and keep the remaining as redundancy).
- **In-band Signaling:** Devices are not allowed to use additional communication links for network, link, and communication coordination.
- **Co-existence:** User data must remain unaffected by the specifically used hardware link or device implementation.

5.3 Solution

Physically redundant network links are either turned off and reserved as spare links or, the redundant links are aggregated into a single higher-performance link (i.e., increased throughput). To that purpose, apply the CONTROL/DATA PLANE PATLET to separated the network traffic into two parts (as shown in Figure 6):

- **the processing part (i.e., the data plane), the sending of user data (i.e., the payload data); and into**
- **the control part (i.e., the control plane), the management of connections and links via header data (i.e., the PHY header).**

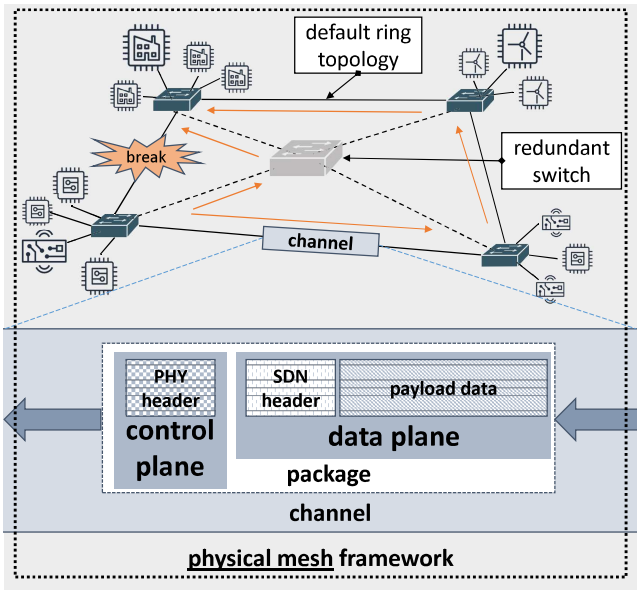

Figure 6: Concept of the physical mesh pattern showing the physical network topology and how protocol data is used for traffic routing.

To ensure the uninterrupted operation of the network, physically redundant network links are used, which generally requires dedicated hardware support, e.g., a seamless failover between links in the case a link becomes unavailable or overloaded. A redundant network link might either be turned off, transfers redundant data, or can be aggregated into channels that perform balancing to split the traffic across all available links in the mesh network, effectively increasing the network throughput.

Beside seamless network operation, it is also possible to provide an even load distribution across all devices by either traffic routing, or by considering non-functional metrics such as cost or energy consumption representing a link as an objective function. The delivery of network packets across a set of available up-links to the internet, for example, utilizes the link with the lowest cost.

As a result, the two forms of traffic (i.e., control and user traffic) can be interpreted as two distinct communication planes (i.e., communication layers). However, the physical mesh pattern does not assign specific resources to each plane; instead, the pattern encapsulates both planes within a shard media or connection, as exemplified in Figure 6. **Unlike the LOGICAL MESH PATTERN (see Section 6), the physical mesh pattern cannot partition or isolate resulting networks from each other, nor does the pattern hide the control plane traffic from other network participants that share the same media.**

The ultimate goal of the physical mesh pattern is, to maintain link redundancy by in the first place removing redundant links for cost-function optimization, and secondly making the removed links immediately available again, in the case they are required, without the user noticing these changes.

5.3.1 Consequences.

+ **Resilience**: Redundant links are marked as spare in case a link becomes unavailable. In such an event, one of the spare connections seamlessly takes over for the failed link.
+ **Performance**: Redundant links can be aggregated into a single, higher performance link. This is achieved by balancing the load across all available links, exposing only a single, more powerful link to the user.
+ **Unattended Operation**: The maintenance of reliable connections is managed by each device individually, without a human in the loop.
+ **In-band Signaling**: Applying the CONTROL/DATA PLANE PATLET allows the same link to be used for all communication. No additional links are required to exchange control data between devices.
- **Co-existence**: While the user data is entirely separate from the control data, no true separation exists between control and data traffic. A user with access to a network segment can inject malicious control data, which could jeopardize the entire network.

5.4 Known uses

Dependable networks traditionally rely on wired infrastructure, typically modified versions of Ethernet to ensure reliable and uninterrupted communication. The spanning tree protocol (STP), originally standardized as IEEE 802.1D, is a well-established mechanism to turn of redundant connections between devices like, e.g., switches can be used to avoid loops within a network. The protocol has been extended for faster response times with the IEEE 802.1w rapid spanning tree protocol which has since become part of the IEEE 802.1Q-2014 standard. The STP is primarily used to avoid switching loops, where a packet would be forwarded (in the worst case indefinitely) along a loop within a network. This is often caused by additional cables that are installed for failover operation. These redundant links would have to be marked manually as not used by disabling the port on the device. However, the STP addresses this issue, by initially blocking all user traffic when a link becomes available, only sending out control traffic. The link with the lowest metric, typically the fastest available, is then chosen as preferred and all redundant links remain unavailable for user traffic. If all links are equally "expensive" (e.g., all links are 1GBps connections), redundant links are turned off and only a single, e.g., 1GBps connection between the devices remains. The redundant links are spare links, which are only activated, if the active connection experiences node or link failures, as shown in Figure 6.

An alternative approach is, to aggregate the links into what is typically referred to as a channel. Link aggregation has been implemented by many vendors but has since moved to a set of standardized protocols. The link aggregation control protocol (LACP), which is part of the IEEE 802.1ax standard, can negotiate channels automatically, quite similar to the STP, which is used to configure spares. However, the LACP enables redundant links to be used simultaneously allowing to provide a 2GBps channel, if two 1GBps links are available.

Commonly wireless communication technologies are not used for the management and control of critical infrastructure. However, specific tasks, such as monitoring or sensing, can be performed via wireless communication channels. WirelessHART [36] and 6TSCH [13], for example, are well-known implementations of wireless protocols for such use cases in industrial environments. While these protocols are designed to be suited for real-time control, their real-time operation cannot be guaranteed due to the shared nature of the freely available radio frequency (RF) spectrum (industrial, scientific, and medical (ISM) band) as the shared transmission medium. This lack of guaranteed (real-time) data transmission, is one essential reason for not adopting wireless technologies in safety-critical environments, where guaranteed timely and predictable behavior is generally inevitable.

A special consideration when using wireless protocols is the circumstance of having only one truly shared channel, which can be accessed by any radio. However, similarly to frequency modulation (FM)-radio stations, where a band of the spectrum (in many nations in the area of 80 MHz to 100 MHz) is further subdivided into channels (which, e.g., would be a single station for FM-radio), only specific frequency ranges can be used free of charge. The frequency ranges must be obtained from the appropriate authorities like cellular bands are auctioned off by individual nations to telecommunication providers. There may also be additional restrictions on the use of specific frequencies, such as being only allowed to use the medium for a specific portion of the time or with limited transmission power. In these use-cases, a control plane is used to coordinate neighboring devices to avoid collisions.

For example, protocols, such as 6TSCH or WirelessHART, are based on IEEE 802.15.4 and use multiple channels as individual connections. The standard specifies 16 channels, each separated by 5 MHz in the range of 2.405 GHz to 2.48 GHz. While the shared nature of the medium would typically make it impossible for multiple devices to send at the same time, the separation into different channels makes it possible for neighboring devices to send data at the same time by using different frequencies which do not interfere with one another. The control plane is used to coordinate which nodes send or receive on which channel.

6 THE LOGICAL MESH PATTERN: APPROACH SOFTWARE DEFINED NETWORKING

6.1 Context

You have applied the PHYSICAL MESH PATTERN to establish dependable communication channels on the physical network level/topology.

6.2 Problem

Often this physical network topology is rather complex, experiences frequent change, is shared by multiple tenancies, and is comprised of various (heterogeneous) devices, computing platforms, and (sub-)networks.

6.2.1 Forces.

- **Abstraction of complex topologies**: Devices need to be able to communicate without the user knowing about the

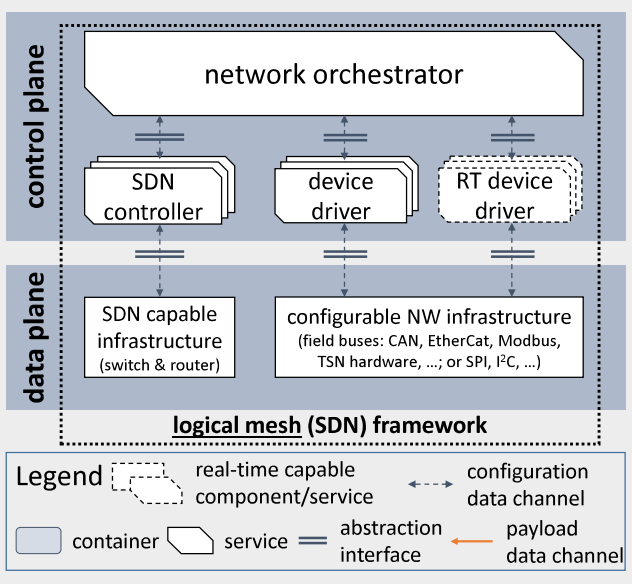

Figure 7: Concept of the logical mesh pattern.

underlying physical network topology. Change of physical network infrastructure/topology should be transparent to applications and services that utilize the network.

- **Virtual networking**: Addresses in the logical network can be assigned as if they were on the same physical network.
- **Multi-tenancy**: Operators need to be able to allow multiple tenants to share a physical network without the ability to observe or modify other user data or control traffic.
- **Security**: The network traffic of individual users must be protected from alteration or access by others. This holds true, even if the underlying network is not trustworthy, e.g., the Internet.
- **Quality of Service**: To provide guaranteed e2e QoS provisioning, it is often necessary to prioritize data packets according to specific operator needs.

6.3 Solution

Create a logical network overlay by establishing logical communication channels on top of the physical network topology, which abstracts the underlying physical network topology and infrastructure (i.e., the links and nodes), finally providing a homogeneous and transparent view on the network to the applications running on the service level. To that purpose, apply the SERVICE-ORIENTED CONTROL/DATA PLANE PATLET to separated the network traffic into two parts (as shown in Figure 7):

- **the processing part (i.e., the data plane), the sending of user data; and into**
- **the control part (i.e., the control plane), the sending of control data for the management of the logical connections/links.**

The logical networks created by applying the logical mesh pattern are entirely separated from one another, making inter-network attacks infeasible, which effectively reduces the potential attack surface. To this goal, it is generally required to support logical network overlays on the operating system level as well as on the network interface level, which enables the configuration of drivers and dedicated network hardware residing on the physical level, which is also shown in Figure 7. Besides the separation of networks, the logical mesh further provides prioritized message/package delivery and band-limits specifiable for all network participants.

As a result, the two forms of traffic, i.e., control traffic and user traffic, can be interpreted as to distinct communication planes (i.e., communication layers). **Unlike the PHYSICAL MESH PATTERN (see Section 5), the logical mesh pattern encapsulates the communication planes, even if they share the same network media or connection.** For example, the logical mesh pattern allows encapsulating raw Ethernet traffic within a TCP-connection atop an existing network, effectively creating a new logical network atop, which is called network overlay. This encapsulated network can be routed through e.g. the Internet to overcome boundaries of the local network, abstracting the underlying topology in the process. **Unlike the SERVICE MESH PATTERN (see Section 7), the logical mesh pattern does not know the concept of applications or services, which utilize the created overlay or logical mesh network.**

The ultimate goal of the logical mesh pattern is to provide a software-defined interface for network operators, providing them with partition capabilities and QoS policies to build several logical overlay networks, abstracting/hiding the actual topology of the underlying (physical and logical) networks, as illustrated in Figure 1.

6.3.1 *Consequences.*

+ **Abstraction of complex topologies**. An entire logical network is overlayed on top of the physical network, and the logical data traffic is encapsulated within packets. In an overlayed topology, typically all devices have a direct connection to each other device on that network, no matter the underlying topology and potentially occurring changes therein.
+ **Virtual networking**. To the physical network (i.e., the physical nodes), the additional control data from the logical network overlay is indistinguishable from any other data traffic that is sent through the network. For any physical device or service in the network, the logical network behaves identical to any other physical network, allowing to change logical network setting without affecting the physical network or service infrastructure.
+ **Multi-tenancy**. Data from multiple tenants can be transmitted simultaneously on the same physical media, as they are regular data traffic to the physical network. However, each tenant is only provided with traffic from his own logical network, either on a port on a switch or a network interface on a physical or virtual machine. This also hides all control data of the physical network from all tenants.
+ **Security**. As all the logical network traffic is transmitted as if it were normal user/payload data on the physical network, a user can still receive all data. However, the logical network can add an encryption layer making it infeasible to access the content of transmitted data. The additional encryption

layer does not impact the traffic routing through the network but adds additional computational effort on the sender and receiver nodes. If required, the logical network can be configured to assign each tenant an individual key.

- **Quality of Service**. On a single machine, QoS policies can be used to prioritize data from, e.g., a single virtual machine or specific processes on the host. Switches and routers, however, need specific support for QoS policies on a per-packet level. The coordination of these QoS policies across shared networks, such as the Internet, is not possible in all cases meaning, that e2e QoS provisioning still depends on both, the physical network support and the logical network support for QoS policies, which both must be coordinated across multiple networks, as well as the physical and the logical network topologies.

6.4 Known uses

While the communication range of wired devices may seem trivially bound to the reach of the cable used to communicate, abstractions such as IP already allow multi-hop communication. In large data-centers, where multiple tenants use the same physical infrastructure to run virtual machines, hosted physical machines, and logical networks of their own, to e.g., run a web server with a separate database and authentication server. When considering the changes imposed by cloud computing in general (i.e., offloading ones own IT services to a third party's data-center), a single data-center can be shared by hundreds or thousands of tenants, many of which are actually competing with one another. In addition, to optimize services on a global scale, it may be required to link multiple data-centers, located in vastly different parts of the world. This is done by establishing a (secure) tunnel between sites/data-centers. For the above purposes of on the one hand splitting a local network among many tenants and on the other combining multiple sites as if they were directly connected, VxLAN is often used. It allows the overlay of multiple networks through the Internet between sites and while (virtual) machines may be separated by vast distances across multiple data-centers, appear to be on the same network. There are a number of solutions, such as OpenStack[3] and CloudStack[4], which include the necessary tools to manage logical mesh networks.

In wireless communication, achieving multi-hop communication requires more effort. In the case of single-hop communication, devices need first to discover one another, synchronize and agree on the usage of both, the frequency and the timing channel. For multi-hop wireless networks, these efforts need to be coordinated not only between neighboring devices, but they must be coordinated within the whole network. 6TSCH [13], for example, creates a mesh network in conjunction with RPL – the routing protocol for low-power and lossy networks. Therefore, 6TSCH creates multiple directed acyclic graphs, which are overlaid to indicate the flow of communication towards a router or destination device. Each device must learn its distance to the destination and pass it on to the devices it relays messages for.

A different approach is used by Bluetooth Mesh[5], which does not rely on network model learning, instead Bluetooth Mesh utilizes cryptography to separate the forwarding of messages and application data by using different keys to encrypt the data. Bluetooth Mesh builds on top of Bluetooth Low Energy (BLE) Core specification and defines a Bearer Layer to facilitate communication between the individual devices. Typically the advertisement feature is of BLE is used to communicate among the mesh devices, while the Generic Attribute Profile (GATT) is sometimes used to communicate with legacy devices, which do not natively support Bluetooth Mesh. This layer represents our physical mesh pattern, as up to this layer, each device in the communication range can receive the messages sent by a device. The network layer takes care of the relay and forwarding functionality while the transport layers handle the encryption, authentication as well as the segmentation of messages. These layers represent the logical mesh pattern, as they provide a virtual network on top of the physical mesh, which can reach beyond the bounds of a single device (multi-hop communication). The encryption in the transport layer uses a key which is shared across all devices within the logical mesh. The access layer allows multiple services, defined in the model layers, to communicate across the logical mesh, with each service using a different key on top of the key used by the transport layer. Therefore, while all devices within the logical mesh can decrypt the messages for forwarding, the application data can only be decrypted by devices which also have the correct application key.

7 THE SERVICE MESH PATTERN: APPROACH CROSS-CUTTING CONCERNS

7.1 Context

Context 1. You have applied the MICROSERVICE ARCHITECTURE PATTERN [32], or your are building a large-scale distributed networked system, that consists of loosely-coupled software components or services.

Context 2. You are in *context 1* and you have also applied the PHYSICAL MESH PATTERN to establish dependable communication channels on the physical network level; and/or you have applied the LOGICAL MESH PATTERN on top of the PHYSICAL MESH PATTERN.

7.2 Problem

In distributed (dependable) networked systems it is necessary to implement the distributed system logic and distributed system functions on the service level, which generally requires the implementation of various cross-cutting concerns, including:

- service discovery, allowing to transparently provide the availability information of service instances, functions, and devices;
- health checking, which enables always to know the state of service instances and devices, whether they are ready to accept network traffic or not;

[3] https://www.openstack.org/
[4] https://cloudstack.apache.org/
[5] https://www.bluetooth.com/specifications/mesh-specifications/

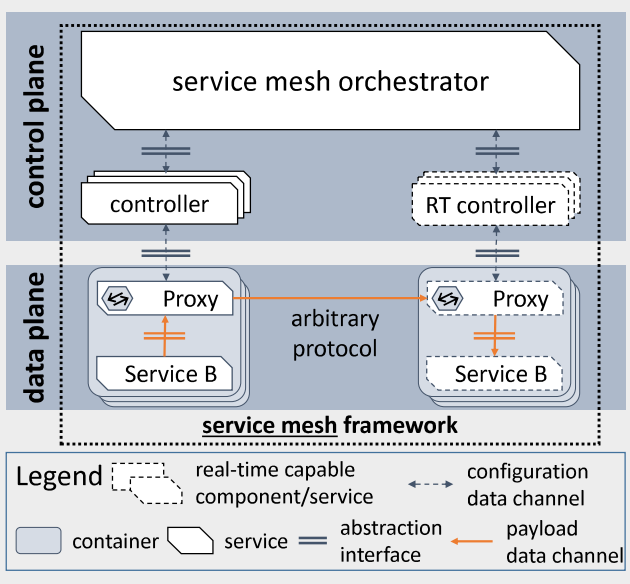

Figure 8: Concept of the service mesh pattern.

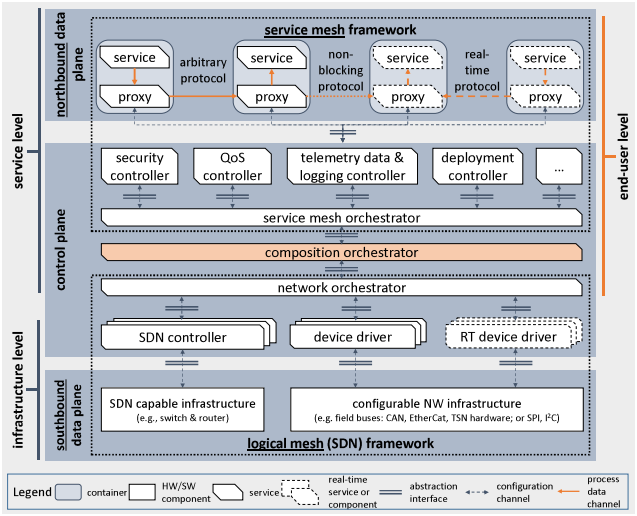

Figure 9: The infrastructure-as-a-service (IaaS) framework representing the solution for context 2.

- authentication and authorization, allowing to cryptographically attested if a caller/service/device is allowed to invoke a specific requested service or function;
- observability, providing detailed statistics, logging, and distributed tracing data to enable distributed debugging features and advanced network traffic analysis;
- routing and load balancing, to enable clustering of services and devices as well as the transparent and efficient transport of data through the network;
- externalized configuration, to allow the decoupling of application and device configuration data from the actual deployment in order to ease or automate the management of security and QoS policies throughout the whole system lifecycle, including system development, testing, startup, and operation.

7.2.1 *Forces.*

- **Loose coupling**: Cross-cutting concern should not yield a tight coupling of system services and functions – a so-called monolith.
- **Graceful system evolution**: Cross-cutting concerns should not restrict the graceful system evolution.
- **Interoperability**: Cross-cutting concerns should not limit the system interoperability.
- **Dependability**: The system dependability must remain unaffected by cross-cutting concerns, meaning that the introduction of common cause failures, as well as single and multi-point failures, must be prevented, when introducing cross-cutting concerns.
- **Scalability**: The coordination and management of cross-cutting concerns should not limit the system scalability.
- **Transparency**: The residence of services should not influence the system logic, the system functions, and the establishment of service communication channels.
- **Holistic system view**: The discovery of the system and service topology, as well as the aggregation of system state information (e.g., health-state, resource usage), should be supported in an efficient manner.
- **Policy management**: The management and configuration of security and QoS policies should be supported in an efficient manner.
- **Development focus**: The implementation of cross-cutting concerns should not be the focus of system developers; instead, system developers should be able to focus on developing system and service functions.
- **System complexity and maintenance**: The implementation of cross-cutting concerns should not increase the maintenance effort and system complexity.

7.3 Solution

To minimize the effects of cross-cutting concerns and to obtain a loose coupling between the actual service implementations and their management logic, apply the SERVICE-ORIENTED CONTROL/DATA PLANE PATLET to separated the service communication layer(s) into two parts:

- **the data plane, which isolates the service-functionality from the service-communication logic by routing the entire in-coming and out-going service traffic through associated proxies; and into**
- **the control plane, which mediates the service-communication traffic on the data plane by configuring, monitoring, and controlling the behavior of the associated service-proxies.**

Solution for context 1. Figure 8 illustrates the solution for *context 1*, where the data plane consists of several distributed service

instances, all communicating via associated proxies. These proxies are configured, monitored, and controlled by the controllers (i.e., distributed control-services) residing on the control plane. In general, an optional orchestrator service is deployed (i.e., the service mesh orchestrator) to coordinate the distributed controllers on the control plane, and to obtain a holistic view of the entire service level (i.e., the data plane services and the data plane service communication).

Solution for context 2. Figure 9 illustrates the solution for *context 2*, where all patterns presented in this paper are applied simultaneously. The resulting system architecture is structured into three main planes:

- The **southbound data plane**, comprised of all network devices on the physical-level – representing *the network infrastructure of the system*;
- the **northbound data plane**, comprised of all software/service functions required to fulfill the purpose of the system; and
- the **control plane**, comprised of all hardware and software controllers that are required to orchestrate and configure the components and systems residing on the northbound and southbound data planes.

The **composition orchestrator**, at the control plane's core, aggregates all monitoring, management, and orchestration mechanism and therefore, provides a holistic view of the entire distributed system including the network nodes on the physical and logical level, as well as all services on the service level. This holistic view is obtained by aggregating the information about the physical level, the logical level, and the service level, which is captured by the controllers on the control plane. Apart from acting as an information source, the **composition orchestrator** can also be used to provide access to specific software, hardware, and network functions to applications on the **end-user level**. These end-user applications can use the holistic view to implement the advanced deployment, interoperability, and resilience mechanisms on top of it, by dynamically (i.e., at run-time) composing the software, hardware, and network functions. Summing up, applying all three pattern results in a framework that enables the provisioning of hardware and software functions via service-interfaces, which effectively implements the *infrastructure-as-a-service (IaaS) paradigm* [12]. In the article *Towards Cyber-Physical Infrastructure as-a-Service in the Era of Industry 4.0* [10], the presented concepts are discussed in more detail.

7.3.1 *Consequences.*

+ **Loose coupling**: A loose coupling is achieved by two means. First, the communication logic is encapsulated from the service implementation via proxies, which allows changing the service function or communication method without the need to update the other mechanism. Second, the isolation into a control plane and a data plane allows exchanging one plane, without the need to change the other plane, as long as the interface between the planes remains constant.
+ **Security**: System security is on the one hand increased by easing the service isolation via communication means, and on the other hand, the system security increases, because the service mesh allows changing communication policies by configuring the connection settings of a proxy. Additionally, service mesh implementations support setting up a public-key-infrastructure (PKI)-based authentication and authorization mechanisms on the proxy level, which can be monitored and managed by the control plane.
+ **Development focus**: The loose coupling and the encapsulation of the communication logic within the proxies allows developers to focus on the service and system function implementation, instead of writing communication and networking code.
+ **Interoperability**: The interoperability of the system increases, since the proxy-service architecture eases the integration of third-party services by supporting the efficient implementation of the API-Gateway Pattern[6]; since an API-Gateway can be simply implemented in the form of *[Proxy]* connects to *[API-Gateway-Service]* connects to *[third-party service]*, which allows to let an initially incompatible third-party service to join the service mesh.
+ **Graceful system evolution**: The loose coupling and the increased interoperability also facilitate graceful system evolution.
+ **Dependability**: System dependability can be increased by deploying redundant services (on different network nodes), which can be easily interconnected and monitored via observing the communication that runs across the proxies. In the case of a connection loss to the control plane, the data plane can still continue operating by establishing (redundant) peer-to-peer connections between various proxies residing on different devices. This requires the service discovery mechanisms (i.e., access to the control plane) only once during the connection setup, and in the case of a failure, the proxy can automatically trigger or execute the fail-over handling by switching to a spare connection.
+ **Dependability and fault-tolerance**: The routing of the traffic through the proxies allows to inject communication faults for simulating network failures, and it also supports load-testing by injecting additional messages on the data plane according to configured patterns dictated by the control plane.
+ **Transparency**: The entire communication is routed through the proxies, which isolates cross-cutting concerns, such as service discovery, communication establishment and management, and message routing, from the service logic.
+ **Holistic system view**: Since each service uses a proxy to join the service mesh, it is possible to create a holistic system view by aggregating all the information within, e.g., the composition orchestrator.
+ **Policy management**: The management and configuration of security and QoS policies on service level can be fully controlled and configured by the control plane via instrumenting the proxies associated to the services on the data plane.

[6] https://microservices.io/patterns/apigateway.html

+ **QoS orchestration**: The composition orchestrator (see Figure 9) can configure all hardware devices and software service via the connected controllers, which enables not only the monitoring of the entire network traffic but also the orchestration of network hardware for establishing dedicated e2e communication channels that provide a guaranteed QoS level.
+ **Scalability**: On the one hand, the scalability is increased, since services meshes facilitate fast system evolution by enabling the easy integration of new services into the service mesh, as well as by supporting load-balancing and traffic routing.
- **Scalability**: On the other hand, the scalability of the system is limited by the additional communication overhead and the increased resource usage, since every service gets assigned an associated proxy.
- **System complexity and maintenance**: Adding additional features, of course, increases the system complexity and the maintenance effort. Furthermore, setting up, configuring, and operating a fully-featured service mesh in a production environment, which also implements the entire list of cross-cutting features mentioned in the problem section, requires much initial effort and know-how. However, it should be worth the effort, once the service mesh is established and CI/CD pipelines are set up.

7.4 Known Uses

The trend in modern large-scale system development is heading towards distributed and loosely-coupled service-oriented architectures, so-called microservice architectures. This distributed SoA-style has its origin in the cloud-computing domain, where it became prominent due to its high flexibility, which enables the co-existence of various operations technology and heterogeneous development environments and teams, all at the same time. While this SoA-trend can also be observed in industrial environments [8, 10, 27, 41, 42], it is most prominent in modern cloud-computing [11, 12, 16, 37]. To efficiently manage and make use of the great flexibility provided by (micro)service architectures, today's systems rely on novel technologies and design patterns like containerization, serverless architectures, CI, and CD. An essential feature, all these approaches require, is the efficient implementation of several cross-cutting concerns, which resulted in the development of various **service mesh implementations** like:

- Istio[7] - open source project by Google Inc.
- Linkerd[8] - open source project
- Consul[9] - open source project by HashiCorp
- Kong[10] - open source project by KongHQ

Figure 10 shows the architecture of the Istio framework, which is a Google Inc. open source project. Several components of this architecture can be mapped to the components shown in Figure 9, such as the **mixer** component can be mapped to the **telemetry data & and logging controller**. The **service mesh orchestrator**, for example, represents the entire Istio framework mostly, since Istio is a generic service mesh implementation allowing the integration of various controllers via adapters. A more detailed discussion and explanation of the mentioned service mesh implementation can be found on their official web site and in several blog posts[11,12,13].

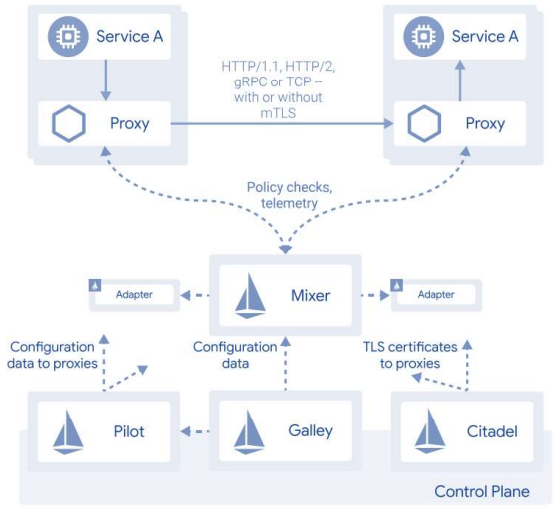

Figure 10: Service mesh architecture of Istio (adapted from Istio).

8 CONCLUSION

In this work, we explained the importance of mesh networking in the context of dependable system design and outlined the need for a holistic system-wide design and engineering approaches. To guide engineers in dependable networked system design, we presented three mesh networking patterns for building dependable mixed-criticality networked systems at all scales. Each of the presented patterns addresses specific design challenges on different technical levels and further provides the context for the application of related patterns.

Although this work focuses explicitly on dependable mesh networked systems, the presented patterns are applicable in various networking contexts, which are more general and where system dependability is subsidiary, and for example, system availability and interoperability are the primary quality criteria instead.

9 ACKNOWLEDGEMENTS

The authors would like to thank their shepherd Christopher Preschern for providing helpful and valuable feedback throughout the shepherding process. We also want to thank the workshop group for giving us constructive feedback about style, content, and readability of the paper.

[7] https://istio.io/
[8] https://linkerd.io/2/architecture/
[9] https://www.consul.io/
[10] https://konghq.com/solutions/service-mesh/
[11] kublr.com: How to implement a Service Mesh with Istio
[12] blog.envoyproxy.io: Service mesh data plane vs. control plane
[13] kubedex.com: Service mesh

REFERENCES

[1] Ashraf Armoush. 2010. Design Patterns for Safety-Critical Embedded Systems. (2010), 384 pages. https://doi.org/10.4236/jsea.2009.21001 arXiv:arXiv:1011.1669v3

[2] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, et al. 2001. Fundamental concepts of dependability. (2001).

[3] Algirdas Avižienis, Jean Claude Laprie, Brian Randell, and Carl Landwehr. 2004. Basic concepts and taxonomy of dependable and secure computing. (2004). https://doi.org/10.1109/TDSC.2004.2

[4] Gedare Bloom, Bassma Alsulami, Ebelechukwu Nwafor, and Ivan Cibrario Bertolotti. 2018. Design patterns for the industrial Internet of Things. (2018). https://doi.org/10.1109/WFCS.2018.8402353

[5] Tiago Boldt Sousa, Ademar Aguiar, Filipe Figueiredo Correia, and Hugo Sereno Ferreira. 2016. Engineering Software for the Cloud - Patterns and Sequences. (2016), 8 pages.

[6] Tiago Boldt Sousa, Filipe Figueiredo Correia, and Hugo Sereno Ferreira. 2015. Patterns for Software Orchestration on the Cloud. (2015).

[7] Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt. 2007. Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing. (2007).

[8] Jürgen Dobaj, Johannes Iber, Michael Krisper, and Christian Kreiner. 2018. A Microservice Architecture for the Industrial Internet-Of-Things. (2018). https://doi.org/10.1145/3282308.3282320

[9] Jürgen Dobaj, Johannes Iber, Michael Krisper, and Christian Kreiner. 2018. Towards Executable Dependability Properties. (2018).

[10] Jürgen Dobaj, Michael Krisper, and Georg Macher. 2019. Towards Cyber-Physical Infrastructure as-a-Service (CPIaaS) in the Era of Industry 4.0. (2019), 310–321 pages. https://doi.org/10.1007/978-3-030-28005-5_24

[11] Qiang Duan, Yuhong Yan, and Athanasios V. Vasilakos. 2012. A survey on service-oriented network virtualization toward convergence of networking and cloud computing. *IEEE Transactions on Network and Service Management* 9, 4 (2012), 373–392. https://doi.org/10.1109/TNSM.2012.113012.120310

[12] Yucong Duan, Guohua Fu, Nianjun Zhou, Xiaobing Sun, Nanjangud C. Narendra, and Bo Hu. 2015. Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends. (2015). https://doi.org/10.1109/CLOUD.2015.88

[13] Diego Dujovne, Thomas Watteyne, Xavier Vilajosana, and Pascal Thubert. 2014. 6TiSCH: Deterministic IP-enabled industrial internet (of things). *IEEE Communications Magazine* 52, 12 (2014), 36–41. https://doi.org/10.1109/MCOM.2014.6979984

[14] Adel S Elmaghraby and Michael M Losavio. 2014. Cyber security challenges in Smart Cities: Safety, security and privacy. (2014).

[15] European Commission. 2016. *A European strategy on Cooperative Intelligent Transport Systems, a milestone towards cooperative, connected and automated mobility.* Technical Report. European Commission.

[16] Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann De Meer, and Xavier Hesselbach. 2013. Virtual network embedding: A survey. *IEEE Communications Surveys and Tutorials* 15, 4 (2013), 1888–1906. https://doi.org/10.1109/SURV.2013.013013.00155

[17] Information Security Forum. 2015. Threat Horizon 2017: Dangers accelerate. (2015). https://www.securityforum.org/research/threat-horizon-2017-dangers-accelerate/

[18] Johannes Iber, Michael Krisper, Jürgen Dobaj, and Christian Kreiner. 2018. Separation of processing and coordination in computer systems. (2018).

[19] Johannes Iber, Tobias Rauter, and Christian Kreiner. 2018. A Self-Adaptive Software System for Increasing the Reliability and Security of Cyber-Physical Systems. (2018), 223–246 pages. https://doi.org/10.4018/978-1-5225-2845-6.ch009

[20] Johannes Iber, Tobias Rauter, Michael Krisper, and Christian Kreiner. 2017. The Potential of Self-Adaptive Software Systems in Industrial Control Systems. In *Communications in Computer and Information Science*. Springer, Cham, Ostrava, Czech Republic, 150–161. https://doi.org/10.1007/978-3-319-64218-5_12

[21] Information Security Forum. 2017. Threat Horizon 2019: Disruption. Distortion. Deterioration. (2017). https://www.securityforum.org/research/threat-horizon-2on-deterioration/

[22] Michael Kircher and Prashant Jain. 2013. Pattern-Oriented Software Architecture Volume 3: Patterns for Resource Management. (2013).

[23] Jay Lee, Hung An Kao, and Shanhu Yang. 2014. Service innovation and smart analytics for Industry 4.0 and big data environment. *Procedia CIRP* 16 (2014), 3–8. https://doi.org/10.1016/j.procir.2014.02.001

[24] Fei Li, Joachim Fröhlich, Daniel Schall, Markus Lachenmayr, Christoph Stückjürgen, Sebastian Meixner, and Frank Buschmann. 2018. Microservice Patterns for the Life Cycle of Industrial Edge Software. (2018). https://doi.org/10.1145/3282308.3282313

[25] Jie Lin, Wei Yu, Nan Zhang, Xinyu Yang, Hanlin Zhang, and Wei Zhao. 2017. A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications. *IEEE Internet of Things Journal* 4, 5 (2017), 1125–1142. https://doi.org/10.1109/JIOT.2017.2683200 arXiv:1305.0982

[26] Matthew Liotine. 2003. Mission-critical Network Planning. (2003), 414 pages.

[27] Yongkui Liu, Lihui Wang, and Xi Vincent Wang. 2018. Cloud manufacturing: latest advancements and future trends. *Procedia Manufacturing* 25 (2018), 62–73. https://doi.org/10.1016/j.promfg.2018.06.058

[28] Timothy Mattson, Beverly Sanders, and Berna Massingill. 2004. Patterns for Parallel Programming. (2004).

[29] Shahid Mumtaz, Ahmed Alsohaily, Zhibo Pang, Ammar Rayes, Kim Fung Tsang, and Jonathan Rodriguez. 2017. Massive Internet of Things for Industrial Applications. (2017), 28–33 pages. https://doi.org/10.13140/RG.2.1.3901.9609

[30] Christopher Preschern. 2014. Pattern-Based Development of Embedded Systems for Safety and Security. (2014).

[31] Soheil Qanbari, Samim Pezeshki, Rozita Raisi, Samira Mahdizadeh, Rabee Rahimzadeh, Negar Behinaein, Fada Mahmoudi, Shiva Ayoubzadeh, Parham Fazlali, Keyvan Roshani, Azalia Yaghini, Mozhdeh Amiri, Ashkan Farivarmoheb, Arash Zamani, and Schahram Dustdar. 2016. IoT Design Patterns: Computational Constructs to Design, Build and Engineer Edge Applications. (apr 2016), 277–282 pages. https://doi.org/10.1109/IoTDI.2015.18

[32] Chris Richardson. [n. d.]. A pattern language for microservices. ([n. d.]). https://microservices.io/patterns/index.html

[33] Thilo Sauter, Stefan Soucek, Wolfgang Kastner, and Dietmar Dietrich. 2011. The evolution of factory and building automation. *IEEE Industrial Electronics Magazine* 5, 3 (2011), 35–48. https://doi.org/10.1109/MIE.2011.942175

[34] Douglas Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. 2000. Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects, Volume 2. (2000).

[35] Markus Schuß, Johannes Iber, Jürgen Dobaj, Christian Kreiner, Carlo Alberto Boano, and Kay Römer. 2018. IoT Device Security the Hard(ware) way. (2018).

[36] Jianping Song, Song Han, Aloysius K. Mok, Deji Chen, Mike Lucas, Mark Nixon, and Wally Pratt. 2008. WirelessHART: Applying wireless technology in real-time industrial process control. (2008), 377–386 pages. https://doi.org/10.1109/RTAS.2008.15

[37] Bhisham Sonkar, Devendra Chaphekar, and Anurag Seetha. 2015. Service Driven Approach towards Future Internet. (2015).

[38] Tiago Boldt Sousa, Hugo Sereno Ferreira, Filipe Figueiredo Correia, and Ademar Aguiar. 2017. Engineering Software for the Cloud. (2017). https://doi.org/10.1145/3147704.3147720

[39] Tiago Boldt Sousa, Hugo Sereno Ferreira, Filipe Figueiredo Correia, and Ademar Aguiar. 2017. Engineering Software for the Cloud: External Monitoring and Failure Injection. (2017). https://doi.org/10.1145/3147704.3147720

[40] Tiago Boldt Sousa, Hugo Sereno Ferreira, Filipe Figueiredo Correia, and Ademar Aguiar. 2019. Engineering Software for the Cloud. (2019). https://doi.org/10.1145/3282308.3282315

[41] Fei Tao, Jiangfeng Cheng, Qinglin Qi, Meng Zhang, He Zhang, and Fangyuan Sui. 2018. Digital twin-driven product design, manufacturing and service with big data. *International Journal of Advanced Manufacturing Technology* 94, 9-12 (2018), 3563–3576. https://doi.org/10.1007/s00170-017-0233-1

[42] F. Tao, L. Zhang, V. C. Venkatesh, Y. Luo, and Y. Cheng. 2011. Cloud manufacturing: A computing and service-oriented manufacturing model. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 225, 10 (2011), 1969–1976. https://doi.org/10.1177/0954405411405575

[43] Danny Weyns, Bradley Schmerl, Vincenzo Grassi, Sam Malek, Raffaela Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese, and Karl M. Göschka. 2013. On patterns for decentralized control in self-adaptive systems. (2013). https://doi.org/10.1007/978-3-642-35813-5_4

[44] Martin Wollschlaeger, Thilo Sauter, and Juergen Jasperneite. 2017. The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. *IEEE Industrial Electronics Magazine* 11, 1 (2017), 17–27. https://doi.org/10.1109/MIE.2017.2649104

[45] Li Da Xu, Wu He, and Shancang Li. 2014. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics* 10, 4 (2014), 2233–2243. https://doi.org/10.1109/TII.2014.2300753 arXiv:arXiv:1011.1669v3

[46] Li Da Xu, Eric L Xu, and Ling Li. 2018. Industry 4.0: state of the art and future trends. (2018).

[47] Uwe Zdun, Michael Kircher, and M. Volter. 2004. Remoting patterns: design reuse of distributed object middleware solutions. *IEEE Internet Computing* 8, 6 (nov 2004), 60–68. https://doi.org/10.1109/MIC.2004.70